

# MySQL Information Schema

---

# MySQL Information Schema

## Abstract

This is the MySQL Replication extract from the MySQL 6.0 Reference Manual.

Document generated on: 2009-06-02 (revision: 15161)

Copyright © 1997-2008 MySQL AB, 2009 Sun Microsystems, Inc. All rights reserved. U.S. Government Rights - Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements. Use is subject to license terms. Sun, Sun Microsystems, the Sun logo, Java, Solaris, StarOffice, MySQL Enterprise Monitor 2.0, MySQL logo™ and MySQL™ are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Copyright © 1997-2008 MySQL AB, 2009 Sun Microsystems, Inc. Tous droits réservés. L'utilisation est soumise aux termes du contrat de licence. Sun, Sun Microsystems, le logo Sun, Java, Solaris, StarOffice, MySQL Enterprise Monitor 2.0, MySQL logo™ et MySQL™ sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

This documentation is NOT distributed under a GPL license. Use of this documentation is subject to the following terms: You may create a printed copy of this documentation solely for your own personal use. Conversion to other formats is allowed as long as the actual content is not altered or edited in any way. You shall not publish or distribute this documentation in any form or on any media, except if you distribute the documentation in a manner similar to how Sun disseminates it (that is, electronically for download on a Web site with the software) or on a CD-ROM or similar medium, provided however that the documentation is disseminated together with the software on the same medium. Any other use, such as any dissemination of printed copies or use of this documentation, in whole or in part, in another publication, requires the prior written consent from an authorized representative of Sun Microsystems, Inc. Sun Microsystems, Inc. and MySQL AB reserve any and all rights to this documentation not expressly granted above.

For more information on the terms of this license, for details on how the MySQL documentation is built and produced, or if you are interested in doing a translation, please contact the [Documentation Team](#).

For additional licensing information, including licenses for libraries used by MySQL, see [Preface, Notes, Licenses](#).

If you want help with using MySQL, please visit either the [MySQL Forums](#) or [MySQL Mailing Lists](#) where you can discuss your issues with other MySQL users.

For additional documentation on MySQL products, including translations of the documentation into other languages, and downloadable versions in variety of formats, including HTML, CHM, and PDF formats, see [MySQL Documentation Library](#).

---

---

---

---

# INFORMATION\_SCHEMA Tables

`INFORMATION_SCHEMA` provides access to database metadata.

*Metadata* is data about the data, such as the name of a database or table, the data type of a column, or access privileges. Other terms that sometimes are used for this information are *data dictionary* and *system catalog*.

`INFORMATION_SCHEMA` is the information database, the place that stores information about all the other databases that the MySQL server maintains. Inside `INFORMATION_SCHEMA` there are several read-only tables. They are actually views, not base tables, so there are no files associated with them.

In effect, we have a database named `INFORMATION_SCHEMA`, although the server does not create a database directory with that name. It is possible to select `INFORMATION_SCHEMA` as the default database with a `USE` statement, but it is possible only to read the contents of tables. You cannot insert into them, update them, or delete from them.

Here is an example of a statement that retrieves information from `INFORMATION_SCHEMA`:

```
mysql> SELECT table_name, table_type, engine
-> FROM information_schema.tables
-> WHERE table_schema = 'db5'
-> ORDER BY table_name DESC;
```

table_name	table_type	engine
v56	VIEW	NULL
v3	VIEW	NULL
v2	VIEW	NULL
v	VIEW	NULL
tables	BASE TABLE	MyISAM
t7	BASE TABLE	MyISAM
t3	BASE TABLE	MyISAM
t2	BASE TABLE	MyISAM
t	BASE TABLE	MyISAM
pk	BASE TABLE	InnoDB
loop	BASE TABLE	MyISAM
kurs	BASE TABLE	MyISAM
k	BASE TABLE	MyISAM
into	BASE TABLE	MyISAM
goto	BASE TABLE	MyISAM
fk2	BASE TABLE	InnoDB
fk	BASE TABLE	InnoDB

17 rows in set (0.01 sec)

Explanation: The statement requests a list of all the tables in database `db5`, in reverse alphabetical order, showing just three pieces of information: the name of the table, its type, and its storage engine.

Each MySQL user has the right to access these tables, but can see only the rows in the tables that correspond to objects for which the user has the proper access privileges. In some cases (for example, the `ROUTINE_DEFINITION` column in the `INFORMATION_SCHEMA.ROUTINES` table), users who have insufficient privileges will see `NULL`.

The `SELECT . . . FROM INFORMATION_SCHEMA` statement is intended as a more consistent way to provide access to the information provided by the various `SHOW` statements that MySQL supports (`SHOW DATABASES`, `SHOW TABLES`, and so forth). Using `SELECT` has these advantages, compared to `SHOW`:

- It conforms to Codd's rules. That is, all access is done on tables.
- Nobody needs to learn a new statement syntax. Because they already know how `SELECT` works, they only need to learn the object names.
- The implementor need not worry about adding keywords.
- There are millions of possible output variations, instead of just one. This provides more flexibility for applications that have varying requirements about what metadata they need.
- Migration is easier because every other DBMS does it this way.

However, because `SHOW` is popular with MySQL employees and users, and because it might be confusing were it to disappear, the advantages of conventional syntax are not a sufficient reason to eliminate `SHOW`. In fact, along with the implementation of `INFORMATION_SCHEMA`, there are enhancements to `SHOW` as well. These are described in [Chapter 27, Extensions to SHOW Statements](#).

There is no difference between the privileges required for `SHOW` statements and those required to select information from `INFORMATION_SCHEMA`. In either case, you have to have some privilege on an object in order to see information about it.

The implementation for the `INFORMATION_SCHEMA` table structures in MySQL follows the ANSI/ISO SQL:2003 standard Part 11 *Schemata*. Our intent is approximate compliance with SQL:2003 core feature F021 *Basic information schema*.

Users of SQL Server 2000 (which also follows the standard) may notice a strong similarity. However, MySQL has omitted many columns that are not relevant for our implementation, and added columns that are MySQL-specific. One such column is the `ENGINE` column in the `INFORMATION_SCHEMA.TABLES` table.

Although other DBMSs use a variety of names, like `syscat` or `system`, the standard name is `INFORMATION_SCHEMA`.

The following sections describe each of the tables and columns that are in `INFORMATION_SCHEMA`. For each column, there are three pieces of information:

- “`INFORMATION_SCHEMA Name`” indicates the name for the column in the `INFORMATION_SCHEMA` table. This corresponds to the standard SQL name unless the “Remarks” field says “MySQL extension.”
- “`SHOW Name`” indicates the equivalent field name in the closest `SHOW` statement, if there is one.
- “Remarks” provides additional information where applicable. If this field is `NULL`, it means that the value of the column is always `NULL`. If this field says “MySQL extension,” the column is a MySQL extension to standard SQL.

To avoid using any name that is reserved in the standard or in DB2, SQL Server, or Oracle, we changed the names of some columns marked “MySQL extension”. (For example, we changed `COLLATION` to `TABLE_COLLATION` in the `TABLES` table.) See the list of reserved words near the end of this article: [http://web.archive.org/web/20070409075643rn\\_1/www.dbazine.com/db2/db2-disarticles/gulutzan5](http://web.archive.org/web/20070409075643rn_1/www.dbazine.com/db2/db2-disarticles/gulutzan5).

The definition for character columns (for example, `TABLES.TABLE_NAME`) is generally `VARCHAR(N) CHARACTER SET utf8` where `N` is at least 64. MySQL uses the default collation for this character set (`utf8_general_ci`) for all searches, sorts, comparisons, and other string operations on such columns. If the default collation is not correct for your needs, you can force a suitable collation with a `COLLATE` clause ([Using COLLATE in SQL Statements](#)).

Each section indicates what `SHOW` statement is equivalent to a `SELECT` that retrieves information from `INFORMATION_SCHEMA`, if there is such a statement. For `SHOW` statements that display information for the current database if you omit a `FROM db_name` clause, you can often select information for the current database by adding an `AND TABLE_SCHEMA = CURRENT_SCHEMA()` condition to the `WHERE` clause of a query that retrieves information from an `INFORMATION_SCHEMA` table.

#### Note

At present, there are some missing columns and some columns out of order. We are working on this and updating the documentation as changes are made.

For answers to questions that are often asked concerning the `INFORMATION_SCHEMA` database, see [Chapter 28, MySQL 5.0 FAQ — INFORMATION\\_SCHEMA](#).

---

# Chapter 1. The INFORMATION\_SCHEMA SCHEMATA Table

A schema is a database, so the SCHEMATA table provides information about databases.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
CATALOG_NAME		def
SCHEMA_NAME		Database
DEFAULT_CHARACTER_SET_NAME		
DEFAULT_COLLATION_NAME		
SQL_PATH		NULL

The following statements are equivalent:

```
SELECT SCHEMA_NAME AS `Database`  
FROM INFORMATION_SCHEMA.SCHEMATA  
[WHERE SCHEMA_NAME LIKE 'wild']  
SHOW DATABASES  
[LIKE 'wild']
```

---

## Chapter 2. The `INFORMATION_SCHEMA TABLES` Table

The `TABLES` table provides information about tables in databases.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>TABLE_CATALOG</code>		<code>def</code>
<code>TABLE_SCHEMA</code>	<code>Table_...</code>	
<code>TABLE_NAME</code>	<code>Table_...</code>	
<code>TABLE_TYPE</code>		
<code>ENGINE</code>	<code>Engine</code>	MySQL extension
<code>VERSION</code>	<code>Version</code>	The version number of the table's <code>.frm</code> file, MySQL extension
<code>ROW_FORMAT</code>	<code>Row_format</code>	MySQL extension
<code>TABLE_ROWS</code>	<code>Rows</code>	MySQL extension
<code>AVG_ROW_LENGTH</code>	<code>Avg_row_length</code>	MySQL extension
<code>DATA_LENGTH</code>	<code>Data_length</code>	MySQL extension
<code>MAX_DATA_LENGTH</code>	<code>Max_data_length</code>	MySQL extension
<code>INDEX_LENGTH</code>	<code>Index_length</code>	MySQL extension
<code>DATA_FREE</code>	<code>Data_free</code>	MySQL extension
<code>AUTO_INCREMENT</code>	<code>Auto_increment</code>	MySQL extension
<code>CREATE_TIME</code>	<code>Create_time</code>	MySQL extension
<code>UPDATE_TIME</code>	<code>Update_time</code>	MySQL extension
<code>CHECK_TIME</code>	<code>Check_time</code>	MySQL extension
<code>TABLE_COLLATION</code>	<code>Collation</code>	MySQL extension
<code>CHECKSUM</code>	<code>Checksum</code>	MySQL extension
<code>CREATE_OPTIONS</code>	<code>Create_options</code>	MySQL extension
<code>TABLE_COMMENT</code>	<code>Comment</code>	MySQL extension

### Notes:

- `TABLE_SCHEMA` and `TABLE_NAME` are a single field in a `SHOW` display, for example `Table_in_db1`.
- `TABLE_TYPE` should be `BASE TABLE` or `VIEW`. Currently, the `TABLES` table does not list `TEMPORARY` tables.
- For partitioned tables, the `ENGINE` column shows the name of the storage engine used by all partitions. (Previously, this column showed `PARTITION` for such tables.)
- The `TABLE_ROWS` column is `NULL` if the table is in the `INFORMATION_SCHEMA` database.  
For `InnoDB` tables, the row count is only a rough estimate used in SQL optimization. (This is also true if the `InnoDB` table is partitioned.)
- Beginning with MySQL 6.0.6, the `DATA_FREE` column shows the free space in bytes for `InnoDB` tables.
- We have nothing for the table's default character set. `TABLE_COLLATION` is close, because collation names begin with a character set name.
- The `CREATE_OPTIONS` column shows `partitioned` if the table is partitioned.

The following statements are equivalent:

```
SELECT table_name FROM INFORMATION_SCHEMA.TABLES
WHERE table_schema = 'db_name'
[AND table_name LIKE 'wild']
SHOW TABLES
FROM db_name
[LIKE 'wild']
```

---

## Chapter 3. The `INFORMATION_SCHEMA.COLUMNS` Table

The `COLUMNS` table provides information about columns in tables.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>TABLE_CATALOG</code>		def
<code>TABLE_SCHEMA</code>		
<code>TABLE_NAME</code>		
<code>COLUMN_NAME</code>	Field	
<code>ORDINAL_POSITION</code>		see notes
<code>COLUMN_DEFAULT</code>	Default	
<code>IS_NULLABLE</code>	Null	
<code>DATA_TYPE</code>	Type	
<code>CHARACTER_MAXIMUM_LENGTH</code>	Type	
<code>CHARACTER_OCTET_LENGTH</code>		
<code>NUMERIC_PRECISION</code>	Type	
<code>NUMERIC_SCALE</code>	Type	
<code>CHARACTER_SET_NAME</code>		
<code>COLLATION_NAME</code>	Collation	
<code>COLUMN_TYPE</code>	Type	MySQL extension
<code>COLUMN_KEY</code>	Key	MySQL extension
<code>EXTRA</code>	Extra	MySQL extension
<code>PRIVILEGES</code>	Privileges	MySQL extension
<code>COLUMN_COMMENT</code>	Comment	MySQL extension
<code>STORAGE</code>	Column storage type	MySQL extension
<code>FORMAT</code>	Column storage format	MySQL extension

### Notes:

- In `SHOW`, the `Type` display includes values from several different `COLUMNS` columns.
- `ORDINAL_POSITION` is necessary because you might want to say `ORDER BY ORDINAL_POSITION`. Unlike `SHOW`, `SELECT` does not have automatic ordering.
- `CHARACTER_OCTET_LENGTH` should be the same as `CHARACTER_MAXIMUM_LENGTH`, except for multi-byte character sets.
- `CHARACTER_SET_NAME` can be derived from `Collation`. For example, if you say `SHOW FULL COLUMNS FROM t`, and you see in the `Collation` column a value of `latin1_swedish_ci`, the character set is what is before the first underscore: `latin1`.
- `STORAGE` and `FORMAT` apply to `NDB` tables. `STORAGE` indicates whether a column is stored on disk or memory, and `FORMAT` indicates the column storage format (`FIXED`, `DYNAMIC`, or `DEFAULT`).

The following statements are nearly equivalent:

```
SELECT COLUMN_NAME, DATA_TYPE, IS_NULLABLE, COLUMN_DEFAULT
FROM INFORMATION_SCHEMA.COLUMNS
WHERE table_name = 'tbl_name'
[AND table_schema = 'db_name']
[AND column_name LIKE 'wild']
SHOW COLUMNS
FROM tbl_name
[FROM db_name]
[LIKE 'wild']
```

---

## Chapter 4. The INFORMATION\_SCHEMA STATISTICS Table

The `STATISTICS` table provides information about table indexes.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
TABLE_CATALOG		def
TABLE_SCHEMA		= Database
TABLE_NAME	Table	
NON_UNIQUE	Non_unique	
INDEX_SCHEMA		= Database
INDEX_NAME	Key_name	
SEQ_IN_INDEX	Seq_in_index	
COLUMN_NAME	Column_name	
COLLATION	Collation	
CARDINALITY	Cardinality	
SUB_PART	Sub_part	MySQL extension
PACKED	Packed	MySQL extension
NULLABLE	Null	MySQL extension
INDEX_TYPE	Index_type	MySQL extension
COMMENT	Comment	MySQL extension
INDEX_COMMENT	Comment	MySQL extension

### Notes:

- There is no standard table for indexes. The preceding list is similar to what SQL Server 2000 returns for `sp_statistics`, except that we replaced the name `QUALIFIER` with `CATALOG` and we replaced the name `OWNER` with `SCHEMA`.

Clearly, the preceding table and the output from `SHOW INDEX` are derived from the same parent. So the correlation is already close.

- The `INDEX_COMMENT` column indicates any comment provided for the index with a `COMMENT` attribute when the index was created.

The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.STATISTICS
  WHERE table_name = 'tbl_name'
     AND table_schema = 'db_name'
SHOW INDEX
  FROM tbl_name
  FROM db_name
```

---

## Chapter 5. The `INFORMATION_SCHEMA` `USER_PRIVILEGES` Table

The `USER_PRIVILEGES` table provides information about global privileges. This information comes from the `mysql.user` grant table.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>GRANTEE</code>		' <i>user_name</i> '@' <i>host_name</i> ' value, MySQL extension
<code>TABLE_CATALOG</code>		<code>def</code> , MySQL extension
<code>PRIVILEGE_TYPE</code>		MySQL extension
<code>IS_GRANTABLE</code>		MySQL extension

### Notes:

- This is a non-standard table. It takes its values from the `mysql.user` table.

---

## Chapter 6. The INFORMATION\_SCHEMA SCHEMA\_PRIVILEGES Table

The SCHEMA\_PRIVILEGES table provides information about schema (database) privileges. This information comes from the `mysql.db` grant table.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
GRANTEE		' <i>user_name</i> '@' <i>host_name</i> ' value, MySQL extension
TABLE_CATALOG		<code>def</code> , MySQL extension
TABLE_SCHEMA		MySQL extension
PRIVILEGE_TYPE		MySQL extension
IS_GRANTABLE		MySQL extension

### Notes:

- This is a non-standard table. It takes its values from the `mysql.db` table.

---

## Chapter 7. The `INFORMATION_SCHEMA` `TABLE_PRIVILEGES` Table

The `TABLE_PRIVILEGES` table provides information about table privileges. This information comes from the `mysql.tables_priv` grant table.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>GRANTEE</code>		' <i>user_name</i> '@' <i>host_name</i> ' value
<code>TABLE_CATALOG</code>		<code>def</code>
<code>TABLE_SCHEMA</code>		
<code>TABLE_NAME</code>		
<code>PRIVILEGE_TYPE</code>		
<code>IS_GRANTABLE</code>		

### Notes:

- `PRIVILEGE_TYPE` can contain one (and only one) of these values: `SELECT`, `INSERT`, `UPDATE`, `REFERENCES`, `ALTER`, `INDEX`, `DROP`, `CREATE VIEW`.

The following statements are *not* equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.TABLE_PRIVILEGES
SHOW GRANTS ...
```

---

## Chapter 8. The `INFORMATION_SCHEMA.COLUMN_PRIVILEGES` Table

The `COLUMN_PRIVILEGES` table provides information about column privileges. This information comes from the `mysql.columns_priv` grant table.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>GRANTEE</code>		' <i>user_name</i> '@' <i>host_name</i> ' value
<code>TABLE_CATALOG</code>		<code>def</code>
<code>TABLE_SCHEMA</code>		
<code>TABLE_NAME</code>		
<code>COLUMN_NAME</code>		
<code>PRIVILEGE_TYPE</code>		
<code>IS_GRANTABLE</code>		

### Notes:

- In the output from `SHOW FULL COLUMNS`, the privileges are all in one field and in lowercase, for example, `select,insert,update,references`. In `COLUMN_PRIVILEGES`, there is one privilege per row, in uppercase.
- `PRIVILEGE_TYPE` can contain one (and only one) of these values: `SELECT, INSERT, UPDATE, REFERENCES`.
- If the user has `GRANT OPTION` privilege, `IS_GRANTABLE` should be `YES`. Otherwise, `IS_GRANTABLE` should be `NO`. The output does not list `GRANT OPTION` as a separate privilege.

The following statements are *not* equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.COLUMN_PRIVILEGES
SHOW GRANTS ...
```

---

## Chapter 9. The `INFORMATION_SCHEMA.CHARACTER_SETS` Table

The `CHARACTER_SETS` table provides information about available character sets.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>CHARACTER_SET_NAME</code>	Charset	
<code>DEFAULT_COLLATE_NAME</code>	Default collation	
<code>DESCRIPTION</code>	Description	MySQL extension
<code>MAXLEN</code>	Maxlen	MySQL extension

The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.CHARACTER_SETS
  [WHERE name LIKE 'wild']
SHOW CHARACTER SET
  [LIKE 'wild']
```

---

## Chapter 10. The `INFORMATION_SCHEMA COLLATIONS` Table

The `COLLATIONS` table provides information about collations for each character set.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>COLLATION_NAME</code>	<code>Collation</code>	
<code>CHARACTER_SET_NAME</code>	<code>Charset</code>	MySQL extension
<code>ID</code>	<code>Id</code>	MySQL extension
<code>IS_DEFAULT</code>	<code>Default</code>	MySQL extension
<code>IS_COMPILED</code>	<code>Compiled</code>	MySQL extension
<code>SORTLEN</code>	<code>Sortlen</code>	MySQL extension

The following statements are equivalent:

```
SELECT COLLATION_NAME FROM INFORMATION_SCHEMA.COLLATIONS
  [WHERE collation_name LIKE 'wild']
SHOW COLLATION
  [LIKE 'wild']
```

---

## Chapter 11. The INFORMATION\_SCHEMA COLLATION\_CHARACTER\_SET\_APPLICABILITY Table

The `COLLATION_CHARACTER_SET_APPLICABILITY` table indicates what character set is applicable for what collation. The columns are equivalent to the first two display fields that we get from `SHOW COLLATION`.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>COLLATION_NAME</code>	Collation	
<code>CHARACTER_SET_NAME</code>	Charset	

---

## Chapter 12. The INFORMATION\_SCHEMA TABLE\_CONSTRAINTS Table

The TABLE\_CONSTRAINTS table describes which tables have constraints.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
CONSTRAINT_CATALOG		def
CONSTRAINT_SCHEMA		
CONSTRAINT_NAME		
TABLE_SCHEMA		
TABLE_NAME		
CONSTRAINT_TYPE		

### Notes:

- The CONSTRAINT\_TYPE value can be UNIQUE, PRIMARY KEY, or FOREIGN KEY.
- The UNIQUE and PRIMARY KEY information is about the same as what you get from the Key\_name field in the output from SHOW INDEX when the Non\_unique field is 0.
- The CONSTRAINT\_TYPE column can contain one of these values: UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK. This is a CHAR (not ENUM) column. The CHECK value is not available until we support CHECK.

---

## Chapter 13. The INFORMATION\_SCHEMA KEY\_COLUMN\_USAGE Table

The `KEY_COLUMN_USAGE` table describes which key columns have constraints.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
<code>CONSTRAINT_CATALOG</code>		def
<code>CONSTRAINT_SCHEMA</code>		
<code>CONSTRAINT_NAME</code>		
<code>TABLE_CATALOG</code>		def
<code>TABLE_SCHEMA</code>		
<code>TABLE_NAME</code>		
<code>COLUMN_NAME</code>		
<code>ORDINAL_POSITION</code>		
<code>POSITION_IN_UNIQUE_CONSTRAINT</code>		
<code>REFERENCED_TABLE_SCHEMA</code>		
<code>REFERENCED_TABLE_NAME</code>		
<code>REFERENCED_COLUMN_NAME</code>		

### Notes:

- If the constraint is a foreign key, then this is the column of the foreign key, not the column that the foreign key references.
- The value of `ORDINAL_POSITION` is the column's position within the constraint, not the column's position within the table. Column positions are numbered beginning with 1.
- The value of `POSITION_IN_UNIQUE_CONSTRAINT` is `NULL` for unique and primary-key constraints. For foreign-key constraints, it is the ordinal position in key of the table that is being referenced.

For example, suppose that there are two tables name `t1` and `t3` that have the following definitions:

```
CREATE TABLE t1
(
  s1 INT,
  s2 INT,
  s3 INT,
  PRIMARY KEY(s3)
) ENGINE=InnoDB;
CREATE TABLE t3
(
  s1 INT,
  s2 INT,
  s3 INT,
  KEY(s1),
  CONSTRAINT CO FOREIGN KEY (s2) REFERENCES t1(s3)
) ENGINE=InnoDB;
```

For those two tables, the `KEY_COLUMN_USAGE` table has two rows:

- One row with `CONSTRAINT_NAME = 'PRIMARY'`, `TABLE_NAME = 't1'`, `COLUMN_NAME = 's3'`, `ORDINAL_POSITION = 1`, `POSITION_IN_UNIQUE_CONSTRAINT = NULL`.
- One row with `CONSTRAINT_NAME = 'CO'`, `TABLE_NAME = 't3'`, `COLUMN_NAME = 's2'`, `ORDINAL_POSITION = 1`, `POSITION_IN_UNIQUE_CONSTRAINT = 1`.

## Chapter 14. The INFORMATION\_SCHEMA ROUTINES Table

The `ROUTINES` table provides information about stored routines (both procedures and functions). The `ROUTINES` table does not include user-defined functions (UDFs) at this time.

The column named “`mysql.proc` name” indicates the `mysql.proc` table column that corresponds to the `INFORMATION_SCHEMA.ROUTINES` table column, if any.

INFORMATION_SCHEMA Name	mysql.proc Name	Remarks
<code>SPECIFIC_NAME</code>	<code>specific_name</code>	
<code>ROUTINE_CATALOG</code>		<code>def</code>
<code>ROUTINE_SCHEMA</code>	<code>db</code>	
<code>ROUTINE_NAME</code>	<code>name</code>	
<code>ROUTINE_TYPE</code>	<code>type</code>	{ <code>PROCEDURE</code>   <code>FUNCTION</code> }
<code>DATA_TYPE</code>		same as for <code>COLUMNS</code> table
<code>CHARACTER_MAXIMUM_LENGTH</code>		same as for <code>COLUMNS</code> table
<code>CHARACTER_OCTET_LENGTH</code>		same as for <code>COLUMNS</code> table
<code>NUMERIC_PRECISION</code>		same as for <code>COLUMNS</code> table
<code>NUMERIC_SCALE</code>		same as for <code>COLUMNS</code> table
<code>CHARACTER_SET_NAME</code>		same as for <code>COLUMNS</code> table
<code>COLLATION_NAME</code>		same as for <code>COLUMNS</code> table
<code>DTD_IDENTIFIER</code>		data type descriptor
<code>ROUTINE_BODY</code>		<code>SQL</code>
<code>ROUTINE_DEFINITION</code>	<code>body</code>	
<code>EXTERNAL_NAME</code>		<code>NULL</code>
<code>EXTERNAL_LANGUAGE</code>	<code>language</code>	<code>NULL</code>
<code>PARAMETER_STYLE</code>		<code>SQL</code>
<code>IS_DETERMINISTIC</code>	<code>is_deterministic</code>	
<code>SQL_DATA_ACCESS</code>	<code>sql_data_access</code>	
<code>SQL_PATH</code>		<code>NULL</code>
<code>SECURITY_TYPE</code>	<code>security_type</code>	
<code>CREATED</code>	<code>created</code>	
<code>LAST_ALTERED</code>	<code>modified</code>	
<code>SQL_MODE</code>	<code>sql_mode</code>	MySQL extension
<code>ROUTINE_COMMENT</code>	<code>comment</code>	MySQL extension
<code>DEFINER</code>	<code>definer</code>	MySQL extension
<code>CHARACTER_SET_CLIENT</code>		MySQL extension
<code>COLLATION_CONNECTION</code>		MySQL extension
<code>DATABASE_COLLATION</code>		MySQL extension

### Notes:

- MySQL calculates `EXTERNAL_LANGUAGE` thus:
  - If `mysql.proc.language = 'SQL'`, `EXTERNAL_LANGUAGE` is `NULL`
  - Otherwise, `EXTERNAL_LANGUAGE` is what is in `mysql.proc.language`. However, we do not have external languages yet, so it is always `NULL`.
- `CHARACTER_SET_CLIENT` is the session value of the `character_set_client` system variable when the routine was created. `COLLATION_CONNECTION` is the session value of the `collation_connection` system variable when the routine was created. `DATABASE_COLLATION` is the collation of the database with which the routine is associated.
- The `DATA_TYPE`, `CHARACTER_MAXIMUM_LENGTH`, `CHARACTER_OCTET_LENGTH`, `NUMERIC_PRECISION`, `NUMERIC_SCALE`, `CHARACTER_SET_NAME`, and `COLLATION_NAME` columns are the same as for the `COLUMNS` table.

`IC_SCALE`, `CHARACTER_SET_NAME`, and `COLLATION_NAME` columns provide information about the data type for the `RETURNS` clause of stored functions. If a stored routine is a stored procedure, these columns all are `NULL`. These columns were added in MySQL 5.2.6.

Information about stored function `RETURNS` data types is also available in the `PARAMETERS` table. The return value data type row for a function can be identified as the row that has an `ORDINAL_POSITION` value of 0.

## Chapter 15. The INFORMATION\_SCHEMA VIEWS Table

The `VIEWS` table provides information about views in databases. You must have the `SHOW VIEW` privilege to access this table.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
TABLE_CATALOG		def
TABLE_SCHEMA		
TABLE_NAME		
VIEW_DEFINITION		
CHECK_OPTION		
IS_UPDATABLE		
DEFINER		
SECURITY_TYPE		
CHARACTER_SET_CLIENT		MySQL extension
COLLATION_CONNECTION		MySQL extension

### Notes:

- The `VIEW_DEFINITION` column has most of what you see in the `Create Table` field that `SHOW CREATE VIEW` produces. Skip the words before `SELECT` and skip the words `WITH CHECK OPTION`. Suppose that the original statement was:

```
CREATE VIEW v AS
  SELECT s2,s1 FROM t
  WHERE s1 > 5
  ORDER BY s1
  WITH CHECK OPTION;
```

Then the view definition looks like this:

```
SELECT s2,s1 FROM t WHERE s1 > 5 ORDER BY s1
```

- The `CHECK_OPTION` column has a value of `NONE`, `CASCADE`, or `LOCAL`.
- MySQL sets a flag, called the view updatability flag, at `CREATE VIEW` time. The flag is set to `YES` (true) if `UPDATE` and `DELETE` (and similar operations) are legal for the view. Otherwise, the flag is set to `NO` (false). The `IS_UPDATABLE` column in the `VIEWS` table displays the status of this flag. It means that the server always knows whether a view is updatable. If the view is not updatable, statements such `UPDATE`, `DELETE`, and `INSERT` are illegal and will be rejected. (Note that even if a view is updatable, it might not be possible to insert into it; for details, refer to [CREATE VIEW Syntax](#).)
- The `DEFINER` column indicates who defined the view. `SECURITY_TYPE` has a value of `DEFINER` or `INVOKER`.
- `CHARACTER_SET_CLIENT` is the session value of the `character_set_client` system variable when the view was created. `COLLATION_CONNECTION` is the session value of the `collation_connection` system variable when the view was created.

MySQL lets you use different `sql_mode` settings to tell the server the type of SQL syntax to support. For example, you might use the `ANSI` SQL mode to ensure MySQL correctly interprets the standard SQL concatenation operator, the double bar (`||`), in your queries. If you then create a view that concatenates items, you might worry that changing the `sql_mode` setting to a value different from `ANSI` could cause the view to become invalid. But this is not the case. No matter how you write out a view definition, MySQL always stores it the same way, in a canonical form. Here is an example that shows how the server changes a double bar concatenation operator to a `CONCAT()` function:

```
mysql> SET sql_mode = 'ANSI';
Query OK, 0 rows affected (0.00 sec)
mysql> CREATE VIEW test.v AS SELECT 'a' || 'b' as coll;
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT VIEW_DEFINITION FROM INFORMATION_SCHEMA.VIEWS
-> WHERE TABLE_SCHEMA = 'test' AND TABLE_NAME = 'v';
+-----+
| VIEW_DEFINITION |
+-----+
| select concat('a','b') AS `coll` |
+-----+
1 row in set (0.00 sec)
```

The advantage of storing a view definition in canonical form is that changes made later to the value of `sql_mode` will not affect the results from the view. However an additional consequence is that comments prior to `SELECT` are stripped from the definition by the server.

---

## Chapter 16. The INFORMATION\_SCHEMA TRIGGERS Table

The `TRIGGERS` table provides information about triggers. You must have the `TRIGGER` privilege to access this table.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
<code>TRIGGER_CATALOG</code>		def
<code>TRIGGER_SCHEMA</code>		
<code>TRIGGER_NAME</code>	Trigger	
<code>EVENT_MANIPULATION</code>	Event	
<code>EVENT_OBJECT_CATALOG</code>		def
<code>EVENT_OBJECT_SCHEMA</code>		
<code>EVENT_OBJECT_TABLE</code>	Table	
<code>ACTION_ORDER</code>		0
<code>ACTION_CONDITION</code>		NULL
<code>ACTION_STATEMENT</code>	Statement	
<code>ACTION_ORIENTATION</code>		ROW
<code>ACTION_TIMING</code>	Timing	
<code>ACTION_REFERENCE_OLD_TABLE</code>		NULL
<code>ACTION_REFERENCE_NEW_TABLE</code>		NULL
<code>ACTION_REFERENCE_OLD_ROW</code>		OLD
<code>ACTION_REFERENCE_NEW_ROW</code>		NEW
<code>CREATED</code>		NULL (0)
<code>SQL_MODE</code>		MySQL extension
<code>DEFINER</code>		MySQL extension
<code>CHARACTER_SET_CLIENT</code>		MySQL extension
<code>COLLATION_CONNECTION</code>		MySQL extension
<code>DATABASE_COLLATION</code>		MySQL extension

### Notes:

- The `TRIGGER_SCHEMA` and `TRIGGER_NAME` columns contain the name of the database in which the trigger occurs and the trigger name, respectively.
- The `EVENT_MANIPULATION` column contains one of the values 'INSERT', 'DELETE', or 'UPDATE'.
- As noted in [Using Triggers](#), every trigger is associated with exactly one table. The `EVENT_OBJECT_SCHEMA` and `EVENT_OBJECT_TABLE` columns contain the database in which this table occurs, and the table's name.
- The `ACTION_ORDER` statement contains the ordinal position of the trigger's action within the list of all similar triggers on the same table. Currently, this value is always 0, because it is not possible to have more than one trigger with the same `EVENT_MANIPULATION` and `ACTION_TIMING` on the same table.
- The `ACTION_STATEMENT` column contains the statement to be executed when the trigger is invoked. This is the same as the text displayed in the `Statement` column of the output from `SHOW TRIGGERS`. Note that this text uses UTF-8 encoding.
- The `ACTION_ORIENTATION` column always contains the value 'ROW'.
- The `ACTION_TIMING` column contains one of the two values 'BEFORE' or 'AFTER'.
- The columns `ACTION_REFERENCE_OLD_ROW` and `ACTION_REFERENCE_NEW_ROW` contain the old and new column identifiers, respectively. This means that `ACTION_REFERENCE_OLD_ROW` always contains the value 'OLD' and `ACTION_REFERENCE_NEW_ROW` always contains the value 'NEW'.
- The `SQL_MODE` column shows the server SQL mode that was in effect at the time when the trigger was created (and thus which remains in effect for this trigger whenever it is invoked, *regardless of the current server SQL mode*). The possible range of values for this column is the same as that of the `sql_mode` system variable. See [Server SQL Modes](#).

- The `DEFINER` column indicates who defined the trigger.
- `CHARACTER_SET_CLIENT` is the session value of the `character_set_client` system variable when the trigger was created. `COLLATION_CONNECTION` is the session value of the `collation_connection` system variable when the trigger was created. `DATABASE_COLLATION` is the collation of the database with which the trigger is associated.
- The following columns currently always contain `NULL`: `ACTION_CONDITION`, `ACTION_REFERENCE_OLD_TABLE`, `ACTION_REFERENCE_NEW_TABLE`, and `CREATED`.

Example, using the `ins_sum` trigger defined in [Using Triggers](#):

```
mysql> SELECT * FROM INFORMATION_SCHEMA.TRIGGERS\G
***** 1. row *****
      TRIGGER_CATALOG: def
      TRIGGER_SCHEMA: test
      TRIGGER_NAME: ins_sum
      EVENT_MANIPULATION: INSERT
      EVENT_OBJECT_CATALOG: def
      EVENT_OBJECT_SCHEMA: test
      EVENT_OBJECT_TABLE: account
      ACTION_ORDER: 0
      ACTION_CONDITION: NULL
      ACTION_STATEMENT: SET @sum = @sum + NEW.amount
      ACTION_ORIENTATION: ROW
      ACTION_TIMING: BEFORE
      ACTION_REFERENCE_OLD_TABLE: NULL
      ACTION_REFERENCE_NEW_TABLE: NULL
      ACTION_REFERENCE_OLD_ROW: OLD
      ACTION_REFERENCE_NEW_ROW: NEW
      CREATED: NULL
      SQL_MODE:
      DEFINER: me@localhost
```

See also [SHOW TRIGGERS Syntax](#).

---

## Chapter 17. The `INFORMATION_SCHEMA PLUGINS` Table

The `PLUGINS` table provides information about server plugins.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>PLUGIN_NAME</code>	Name	MySQL extension
<code>PLUGIN_VERSION</code>		MySQL extension
<code>PLUGIN_STATUS</code>	Status	MySQL extension
<code>PLUGIN_TYPE</code>	Type	MySQL extension
<code>PLUGIN_TYPE_VERSION</code>		MySQL extension
<code>PLUGIN_LIBRARY</code>	Library	MySQL extension
<code>PLUGIN_LIBRARY_VERSION</code>		MySQL extension
<code>PLUGIN_AUTHOR</code>		MySQL extension
<code>PLUGIN_DESCRIPTION</code>		MySQL extension
<code>PLUGIN_LICENSE</code>		MySQL extension

### Notes:

- The `PLUGINS` table is a non-standard table.

See also `SHOW PLUGINS Syntax`.

---

## Chapter 18. The `INFORMATION_SCHEMA ENGINES` Table

The `PLUGINS` table provides information about storage engines.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>ENGINE</code>	<code>Engine</code>	MySQL extension
<code>SUPPORT</code>	<code>Support</code>	MySQL extension
<code>COMMENT</code>	<code>Comment</code>	MySQL extension
<code>TRANSACTIONS</code>	<code>Transactions</code>	MySQL extension
<code>XA</code>	<code>XA</code>	MySQL extension
<code>SAVEPOINTS</code>	<code>Savepoints</code>	MySQL extension

### Notes:

- The `ENGINES` table is a non-standard table.

See also `SHOW ENGINES Syntax`.

---

## Chapter 19. The `INFORMATION_SCHEMA PARTITIONS` Table

The `PARTITIONS` table provides information about table partitions. See [Partitioning](#), for more information about partitioning tables.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>TABLE_CATALOG</code>		MySQL extension
<code>TABLE_SCHEMA</code>		MySQL extension
<code>TABLE_NAME</code>		MySQL extension
<code>PARTITION_NAME</code>		MySQL extension
<code>SUBPARTITION_NAME</code>		MySQL extension
<code>PARTITION_ORDINAL_POSITION</code>		MySQL extension
<code>SUBPARTITION_ORDINAL_POSITION</code>		MySQL extension
<code>PARTITION_METHOD</code>		MySQL extension
<code>SUBPARTITION_METHOD</code>		MySQL extension
<code>PARTITION_EXPRESSION</code>		MySQL extension
<code>SUBPARTITION_EXPRESSION</code>		MySQL extension
<code>PARTITION_DESCRIPTION</code>		MySQL extension
<code>TABLE_ROWS</code>		MySQL extension
<code>AVG_ROW_LENGTH</code>		MySQL extension
<code>DATA_LENGTH</code>		MySQL extension
<code>MAX_DATA_LENGTH</code>		MySQL extension
<code>INDEX_LENGTH</code>		MySQL extension
<code>DATA_FREE</code>		MySQL extension
<code>CREATE_TIME</code>		MySQL extension
<code>UPDATE_TIME</code>		MySQL extension
<code>CHECK_TIME</code>		MySQL extension
<code>CHECKSUM</code>		MySQL extension
<code>PARTITION_COMMENT</code>		MySQL extension
<code>NODEGROUP</code>		MySQL extension
<code>TABLESPACE_NAME</code>		MySQL extension

### Notes:

- The `PARTITIONS` table is a non-standard table.  
Each record in this table corresponds to an individual partition or subpartition of a partitioned table.
- `TABLE_CATALOG`: This column is always `def`.
- `TABLE_SCHEMA`: This column contains the name of the database to which the table belongs.
- `TABLE_NAME`: This column contains the name of the table containing the partition.
- `PARTITION_NAME`: The name of the partition.
- `SUBPARTITION_NAME`: If the `PARTITIONS` table record represents a subpartition, then this column contains the name of subpartition; otherwise it is `NULL`.
- `PARTITION_ORDINAL_POSITION`: All partitions are indexed in the same order as they are defined, with `1` being the number assigned to the first partition. The indexing can change as partitions are added, dropped, and reorganized; the number shown in this column reflects the current order, taking into account any indexing changes.
- `SUBPARTITION_ORDINAL_POSITION`: Subpartitions within a given partition are also indexed and reindexed in the same manner as partitions are indexed within a table.

- **PARTITION\_METHOD**: One of the values `RANGE`, `LIST`, `HASH`, `LINEAR HASH`, `KEY`, or `LINEAR KEY`; that is, one of the available partitioning types as discussed in [Partition Types](#).
- **SUBPARTITION\_METHOD**: One of the values `HASH`, `LINEAR HASH`, `KEY`, or `LINEAR KEY`; that is, one of the available subpartitioning types as discussed in [Subpartitioning](#).
- **PARTITION\_EXPRESSION**: This is the expression for the partitioning function used in the `CREATE TABLE` or `ALTER TABLE` statement that created the table's current partitioning scheme.

For example, consider a partitioned table created in the `test` database using this statement:

```
CREATE TABLE tp (
  c1 INT,
  c2 INT,
  c3 VARCHAR(25)
)
PARTITION BY HASH(c1 + c2)
PARTITIONS 4;
```

The **PARTITION\_EXPRESSION** column in a PARTITIONS table record for a partition from this table displays `c1 + c2`, as shown here:

```
mysql> SELECT DISTINCT PARTITION_EXPRESSION
> FROM INFORMATION_SCHEMA.PARTITIONS
> WHERE TABLE_NAME='tp' AND TABLE_SCHEMA='test';
+-----+
| PARTITION_EXPRESSION |
+-----+
| c1 + c2              |
+-----+
1 row in set (0.09 sec)
```

- **SUBPARTITION\_EXPRESSION**: This works in the same fashion for the subpartitioning expression that defines the subpartitioning for a table as **PARTITION\_EXPRESSION** does for the partitioning expression used to define a table's partitioning.

If the table has no subpartitions, then this column is `NULL`.

- **PARTITION\_DESCRIPTION**: This column is used for `RANGE` and `LIST` partitions. For a `RANGE` partition, it contains the value set in the partition's `VALUES LESS THAN` clause, which can be either an integer or `MAXVALUE`. For a `LIST` partition, this column contains the values defined in the partition's `VALUES IN` clause, which is a comma-separated list of integer values.

For partitions whose **PARTITION\_METHOD** is other than `RANGE` or `LIST`, this column is always `NULL`.

- **TABLE\_ROWS**: The number of table rows in the partition.

For partitioned `InnoDB` tables, the row count given in the **TABLE\_ROWS** column is only an estimated value used in SQL optimization, and may not always be exact.

- **AVG\_ROW\_LENGTH**: The average length of the rows stored in this partition or subpartition, in bytes.

This is the same as **DATA\_LENGTH** divided by **TABLE\_ROWS**.

- **DATA\_LENGTH**: The total length of all rows stored in this partition or subpartition, in bytes — that is, the total number of bytes stored in the partition or subpartition.
- **MAX\_DATA\_LENGTH**: The maximum number of bytes that can be stored in this partition or subpartition.
- **INDEX\_LENGTH**: The length of the index file for this partition or subpartition, in bytes.
- **DATA\_FREE**: The number of bytes allocated to the partition or subpartition but not used.
- **CREATE\_TIME**: The time of the partition's or subpartition's creation.
- **UPDATE\_TIME**: The time that the partition or subpartition was last modified.
- **CHECK\_TIME**: The last time that the table to which this partition or subpartition belongs was checked.

**Note**

Some storage engines do not update this time; for tables using these storage engines, this value is always `NULL`.

- **CHECKSUM**: The checksum value, if any; otherwise, this column is `NULL`.

- `PARTITION_COMMENT`: This column contains the text of any comment made for the partition.

The default value for this column is an empty string.

- `NODEGROUP`: This is the nodegroup to which the partition belongs. This is relevant only to MySQL Cluster tables; otherwise the value of this column is always `0`.

#### Note

The `NDBCLUSTER` storage engine is currently not supported in MySQL 6.0. If you are interested in using MySQL Cluster, see [MySQL Cluster NDB 6.X/7.X](#), which provides information about MySQL Cluster NDB 6.2 and 6.3 (based on MySQL 5.1 but containing the latest improvements and fixes for `NDBCLUSTER`).

- `TABLESPACE_NAME`: This column contains the name of tablespace to which the partition belongs. In MySQL 6.0, the value of this column is always `DEFAULT`.
- A non-partitioned table has one record in `INFORMATION_SCHEMA.PARTITIONS`; however, the values of the `PARTITION_NAME`, `SUBPARTITION_NAME`, `PARTITION_ORDINAL_POSITION`, `SUBPARTITION_ORDINAL_POSITION`, `PARTITION_METHOD`, `SUBPARTITION_METHOD`, `PARTITION_EXPRESSION`, `SUBPARTITION_EXPRESSION`, and `PARTITION_DESCRIPTION` columns are all `NULL`. (The `PARTITION_COMMENT` column in this case is blank.)

---

## Chapter 20. The `INFORMATION_SCHEMA` `EVENTS` Table

The `EVENTS` table provides information about scheduled events, which are discussed in [Using the Event Scheduler](#).

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>EVENT_CATALOG</code>		<code>def</code> , MySQL extension
<code>EVENT_SCHEMA</code>	<code>Db</code>	MySQL extension
<code>EVENT_NAME</code>	<code>Name</code>	MySQL extension
<code>DEFINER</code>	<code>Definer</code>	MySQL extension
<code>TIME_ZONE</code>	<code>Time zone</code>	MySQL extension
<code>EVENT_BODY</code>		MySQL extension
<code>EVENT_DEFINITION</code>		MySQL extension
<code>EVENT_TYPE</code>	<code>Type</code>	MySQL extension
<code>EXECUTE_AT</code>	<code>Execute at</code>	MySQL extension
<code>INTERVAL_VALUE</code>	<code>Interval value</code>	MySQL extension
<code>INTERVAL_FIELD</code>	<code>Interval field</code>	MySQL extension
<code>SQL_MODE</code>		MySQL extension
<code>STARTS</code>	<code>Starts</code>	MySQL extension
<code>ENDS</code>	<code>Ends</code>	MySQL extension
<code>STATUS</code>	<code>Status</code>	MySQL extension
<code>ON_COMPLETION</code>		MySQL extension
<code>CREATED</code>		MySQL extension
<code>LAST_ALTERED</code>		MySQL extension
<code>LAST_EXECUTED</code>		MySQL extension
<code>EVENT_COMMENT</code>		MySQL extension
<code>ORIGINATOR</code>	<code>Originator</code>	MySQL extension
<code>CHARACTER_SET_CLIENT</code>		MySQL extension
<code>COLLATION_CONNECTION</code>		MySQL extension
<code>DATABASE_COLLATION</code>		MySQL extension

### Notes:

- The `EVENTS` table is a non-standard table.
- `EVENT_CATALOG`: The value of this column is always `def`.
- `EVENT_SCHEMA`: The name of the schema (database) to which this event belongs.
- `EVENT_NAME`: The name of the event.
- `DEFINER`: The user who created the event. Always displayed in `'user_name'@'host_name'` format.
- `TIME_ZONE`: The time zone in effect when schedule for the event was last modified. If the event's schedule has not been modified since the event was created, then this is the time zone that was in effect at the event's creation. The default value is `SYS-TEM`.
- `EVENT_BODY`: The language used for the statements in the event's `DO` clause; in MySQL 6.0, this is always `SQL`.  
  
This column is not to be confused with the column of the same name (now named `EVENT_DEFINITION`) that existed in earlier MySQL versions.
- `EVENT_DEFINITION`: The text of the SQL statement making up the event's `DO` clause; in other words, the statement executed by this event.
- `EVENT_TYPE`: One of the two values `ONE TIME` or `RECURRING`.
- `EXECUTE_AT`: For a one-time event, this is the `DATETIME` value specified in the `AT` clause of the `CREATE EVENT` statement

used to create the event, or of the last `ALTER EVENT` statement that modified the event. The value shown in this column reflects the addition or subtraction of any `INTERVAL` value included in the event's `AT` clause. For example, if an event is created using `ON SCHEDULE AT CURRENT_TIMESTAMP + '1:6' DAY_HOUR`, and the event was created at 2006-02-09 14:05:30, the value shown in this column would be `'2006-02-10 20:05:30'`.

If the event's timing is determined by an `EVERY` clause instead of an `AT` clause (that is, if the event is recurring), the value of this column is `NULL`.

- **INTERVAL\_VALUE:** For recurring events, this column contains the numeric portion of the event's `EVERY` clause.

For a one-time event (that is, an event whose timing is determined by an `AT` clause), this column's value is `NULL`.

- **INTERVAL\_FIELD:** For recurring events, this column contains the units portion of the `EVERY` clause governing the timing of the event. Thus, this column contains a value such as `'YEAR'`, `'QUARTER'`, `'DAY'`, and so on.

For a one-time event (that is, an event whose timing is determined by an `AT` clause), this column's value is `NULL`.

- **SQL\_MODE:** The SQL mode in effect at the time the event was created or altered.
- **STARTS:** For a recurring event whose definition includes a `STARTS` clause, this column contains the corresponding `DATE-TIME` value. As with the `EXECUTE_AT` column, this value resolves any expressions used.

If there is no `STARTS` clause affecting the timing of the event, this column is empty.

- **ENDS:** For a recurring event whose definition includes a `ENDS` clause, this column contains the corresponding `DATETIME` value. As with the `EXECUTE_AT` column (see previous example), this value resolves any expressions used.

If there is no `ENDS` clause affecting the timing of the event, this column contains `NULL`.

- **STATUS:** One of the three values `ENABLED`, `DISABLED`, or `SLAVESIDE_DISABLED`.

`SLAVESIDE_DISABLED` indicates that the creation of the event occurred on another MySQL server acting as a replication master and was replicated to the current MySQL server which is acting as a slave, but the event is not presently being executed on the slave. See [Replication of Invoked Features](#), for more information.

- **ON\_COMPLETION:** One of the two values `PRESERVE` or `NOT PRESERVE`.
- **CREATED:** The date and time when the event was created. This is a `DATETIME` value.
- **LAST\_ALTERED:** The date and time when the event was last modified. This is a `DATETIME` value. If the event has not been modified since its creation, this column holds the same value as the `CREATED` column.
- **LAST\_EXECUTED:** The date and time when the event last executed. A `DATETIME` value. If the event has never executed, this column's value is `NULL`.

Before MySQL 6.0.5, `LAST_EXECUTED` indicates when event finished executing. As of 6.0.5, `LAST_EXECUTED` instead indicates when the event started. As a result, the `ENDS` column is never less than `LAST_EXECUTED`.

- **EVENT\_COMMENT:** The text of a comment, if the event has one. If there is no comment, the value of this column is an empty string.
- **ORIGINATOR:** The server ID of the MySQL server on which the event was created; used in replication. The default value is 0.
- **CHARACTER\_SET\_CLIENT** is the session value of the `character_set_client` system variable when the event was created. **COLLATION\_CONNECTION** is the session value of the `collation_connection` system variable when the event was created. **DATABASE\_COLLATION** is the collation of the database with which the event is associated.

**Example:** Suppose the user `jon@ghidora` creates an event named `e_daily`, and then modifies it a few minutes later using an `ALTER EVENT` statement, as shown here:

```
DELIMITER |
CREATE EVENT e_daily
ON SCHEDULE
EVERY 1 DAY
COMMENT 'Saves total number of sessions then clears the table each day'
DO
BEGIN
INSERT INTO site_activity.totals (time, total)
SELECT CURRENT_TIMESTAMP, COUNT(*)
FROM site_activity.sessions;
DELETE FROM site_activity.sessions;
END |
DELIMITER ;
ALTER EVENT e_daily
```

---

ENABLED;

(Note that comments can span multiple lines.)

This user can then run the following `SELECT` statement, and obtain the output shown:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.EVENTS
> WHERE EVENT_NAME = 'e_daily'
> AND EVENT_SCHEMA = 'myschema'\G
***** 1. row *****
EVENT_CATALOG: def
EVENT_SCHEMA: test
EVENT_NAME: e_daily
DEFINER: paul@localhost
TIME_ZONE: SYSTEM
EVENT_BODY: SQL
EVENT_DEFINITION: BEGIN
INSERT INTO site_activity.totals (time, total)
SELECT CURRENT_TIMESTAMP, COUNT(*)
FROM site_activity.sessions;
DELETE FROM site_activity.sessions;
END
EVENT_TYPE: RECURRING
EXECUTE_AT: NULL
INTERVAL_VALUE: 1
INTERVAL_FIELD: DAY
SQL_MODE:
STARTS: 2008-09-03 12:13:39
ENDS: NULL
STATUS: ENABLED
ON_COMPLETION: NOT PRESERVE
CREATED: 2008-09-03 12:13:39
LAST_ALTERED: 2008-09-03 12:13:39
LAST_EXECUTED: NULL
EVENT_COMMENT: Saves total number of sessions then clears the
table each day
ORIGINATOR: 1
CHARACTER_SET_CLIENT: latin1
COLLATION_CONNECTION: latin1_swedish_ci
DATABASE_COLLATION: latin1_swedish_ci
```

These times are all given in terms of local time as determined by the MySQL server's `time_zone` setting. (The same is true of the `starts`, `ends`, and `last_executed` columns of the `mysql.event` table as well as the `Starts` and `Ends` columns in the output of `SHOW [FULL] EVENTS`.)

The `CREATED` and `LAST_ALTERED` columns use the server time zone (as do the `created` and `last_altered` columns of the `mysql.event` table).

See also `SHOW EVENTS Syntax`.

---

## Chapter 21. The INFORMATION\_SCHEMA FILES Table

The `FILES` table provides information about the files in which MySQL `Falcon` tablespace data is stored. The table was added in MySQL 6.0.8.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
<code>FILE_ID</code>		MySQL extension
<code>FILE_NAME</code>		MySQL extension
<code>FILE_TYPE</code>		MySQL extension
<code>TABLESPACE_NAME</code>		MySQL extension
<code>TABLE_CATALOG</code>		MySQL extension
<code>TABLE_SCHEMA</code>		MySQL extension
<code>TABLE_NAME</code>		MySQL extension
<code>LOGFILE_GROUP_NAME</code>		MySQL extension
<code>LOGFILE_GROUP_NUMBER</code>		MySQL extension
<code>ENGINE</code>		MySQL extension
<code>FULLTEXT_KEYS</code>		MySQL extension
<code>DELETED_ROWS</code>		MySQL extension
<code>UPDATE_COUNT</code>		MySQL extension
<code>FREE_EXTENTS</code>		MySQL extension
<code>TOTAL_EXTENTS</code>		MySQL extension
<code>EXTENT_SIZE</code>		MySQL extension
<code>INITIAL_SIZE</code>		MySQL extension
<code>MAXIMUM_SIZE</code>		MySQL extension
<code>AUTOEXTEND_SIZE</code>		MySQL extension
<code>CREATION_TIME</code>		MySQL extension
<code>LAST_UPDATE_TIME</code>		MySQL extension
<code>LAST_ACCESS_TIME</code>		MySQL extension
<code>RECOVER_TIME</code>		MySQL extension
<code>TRANSACTION_COUNTER</code>		MySQL extension
<code>VERSION</code>		MySQL extension
<code>ROW_FORMAT</code>		MySQL extension
<code>TABLE_ROWS</code>		MySQL extension
<code>AVG_ROW_LENGTH</code>		MySQL extension
<code>DATA_LENGTH</code>		MySQL extension
<code>MAX_DATA_LENGTH</code>		MySQL extension
<code>INDEX_LENGTH</code>		MySQL extension
<code>DATA_FREE</code>		MySQL extension
<code>CREATE_TIME</code>		MySQL extension
<code>UPDATE_TIME</code>		MySQL extension
<code>CHECK_TIME</code>		MySQL extension
<code>CHECKSUM</code>		MySQL extension
<code>STATUS</code>		MySQL extension
<code>EXTRA</code>		MySQL extension

### Notes:

- `FILE_ID` column values are auto-generated.
- `FILE_NAME` is the name of a data file created by `CREATE TABLESPACE` or `ALTER TABLESPACE`.

- `FILE_TYPE` is `SYSTEM DATAFILE` for the `FALCON_MASTER`, `FALCON_USER` and `FALCON_TEMPORARY` tablespace files, or `USER DATAFILE` for any user-created tablespace files.
- `TABLESPACE_NAME` is the name of the tablespace with which the file is associated.
- Currently, the value of the `TABLESPACE_CATALOG` column is always `NULL`.
- `TABLE_NAME` is the name of the table with which the file is associated, if any.
- The `EXTENT_SIZE` is always 0.
- For MySQL Falcon tablespace files, the following columns are always `NULL`:
  - `TABLESPACE_CATALOG`
  - `TABLE_SCHEMA`
  - `TABLE_NAME`
  - `LOGFILE_GROUP_NAME`
  - `LOGFILE_GROUP_NUMBER`
  - `FULLTEXT_KEYS`
  - `DELETED_ROWS`
  - `UPDATE_COUNT`
  - `FREE_EXTENTS`
  - `TOTAL_EXTENTS`
  - `INITIAL_SIZE`
  - `MAXIMUM_SIZE`
  - `AUTOEXTEND_SIZE`
  - `CREATION_TIME`
  - `LAST_UPDATE_TIME`
  - `LAST_ACCESS_TIME`
  - `RECOVER_TIME`
  - `TRANSACTION_COUNTER`
  - `VERSION`
  - `ROW_FORMAT`
  - `TABLE_ROWS`
  - `AVG_ROW_LENGTH`
  - `DATA_LENGTH`
  - `MAX_DATA_LENGTH`
  - `INDEX_LENGTH`
  - `DATA_FREE`
  - `CREATE_TIME`
  - `UPDATE_TIME`
  - `CHECK_TIME`
  - `CHECKSUM`

- For `Falcon` tablespaces, the value of the `STATUS` column is always `NORMAL`.
- There are no `SHOW` commands associated with the `FILES` table.

---

## Chapter 22. The `INFORMATION_SCHEMA.PROCESSLIST` Table

The `PROCESSLIST` table provides information about which threads are running.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>ID</code>	<code>Id</code>	MySQL extension
<code>USER</code>	<code>User</code>	MySQL extension
<code>HOST</code>	<code>Host</code>	MySQL extension
<code>DB</code>	<code>db</code>	MySQL extension
<code>COMMAND</code>	<code>Command</code>	MySQL extension
<code>TIME</code>	<code>Time</code>	MySQL extension
<code>STATE</code>	<code>State</code>	MySQL extension
<code>INFO</code>	<code>Info</code>	MySQL extension

For an extensive description of the table columns, see [SHOW PROCESSLIST Syntax](#).

### Notes:

- The `PROCESSLIST` table is a non-standard table.
- Like the output from the corresponding `SHOW` statement, the `PROCESSLIST` table will only show information about your own threads, unless you have the `PROCESS` privilege, in which case you will see information about other threads, too. As an anonymous user, you cannot see any rows at all.
- If an SQL statement refers to `INFORMATION_SCHEMA.PROCESSLIST`, then MySQL will populate the entire table once, when statement execution begins, so there is read consistency during the statement. There is no read consistency for a multi-statement transaction, though.

The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST
SHOW FULL PROCESSLIST
```

---

## Chapter 23. The INFORMATION\_SCHEMA REFERENTIAL\_CONSTRAINTS Table

The REFERENTIAL\_CONSTRAINTS table provides information about foreign keys.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
CONSTRAINT_CATALOG		def
CONSTRAINT_SCHEMA		
CONSTRAINT_NAME		
UNIQUE_CONSTRAINT_CATALOG		def
UNIQUE_CONSTRAINT_SCHEMA		
UNIQUE_CONSTRAINT_NAME		
MATCH_OPTION		
UPDATE_RULE		
DELETE_RULE		
TABLE_NAME		
REFERENCED_TABLE_NAME		

### Notes:

- TABLE\_NAME has the same value as TABLE\_NAME in INFORMATION\_SCHEMA.TABLE\_CONSTRAINTS.
- CONSTRAINT\_SCHEMA and CONSTRAINT\_NAME identify the foreign key.
- UNIQUE\_CONSTRAINT\_SCHEMA, UNIQUE\_CONSTRAINT\_NAME, and REFERENCED\_TABLE\_NAME identify the referenced key.
- The only valid value at this time for MATCH\_OPTION is NONE.
- The possible values for UPDATE\_RULE or DELETE\_RULE are CASCADE, SET NULL, SET DEFAULT, RESTRICT, NO ACTION.

---

## Chapter 24. The `INFORMATION_SCHEMA` `GLOBAL_STATUS` and `SESSION_STATUS` Tables

The `GLOBAL_STATUS` and `SESSION_STATUS` tables provide information about server status variables. Their contents correspond to the information produced by the `SHOW GLOBAL STATUS` and `SHOW SESSION STATUS` statements (see [SHOW STATUS Syntax](#)).

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>VARIABLE_NAME</code>	Variable_name	
<code>VARIABLE_VALUE</code>	Value	

### Notes:

- The `VARIABLE_VALUE` column for each of these tables is defined as `VARCHAR(20480)`.

---

## Chapter 25. The `INFORMATION_SCHEMA` `GLOBAL_VARIABLES` and `SESSION_VARIABLES` Tables

The `GLOBAL_VARIABLES` and `SESSION_VARIABLES` tables provide information about server status variables. Their contents correspond to the information produced by the `SHOW GLOBAL VARIABLES` and `SHOW SESSION VARIABLES` statements (see `SHOW VARIABLES Syntax`).

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>VARIABLE_NAME</code>	Variable_name	
<code>VARIABLE_VALUE</code>	Value	

### Notes:

- The `VARIABLE_VALUE` column for each of these tables is defined as `VARCHAR(20480)`.

---

## Chapter 26. Other `INFORMATION_SCHEMA` Tables

We intend to implement additional `INFORMATION_SCHEMA` tables.

## Chapter 27. Extensions to SHOW Statements

Some extensions to `SHOW` statements accompany the implementation of `INFORMATION_SCHEMA`:

- `SHOW` can be used to get information about the structure of `INFORMATION_SCHEMA` itself.
- Several `SHOW` statements accept a `WHERE` clause that provides more flexibility in specifying which rows to display.

`INFORMATION_SCHEMA` is an information database, so its name is included in the output from `SHOW DATABASES`. Similarly, `SHOW TABLES` can be used with `INFORMATION_SCHEMA` to obtain a list of its tables:

```
mysql> SHOW TABLES FROM INFORMATION_SCHEMA;
+-----+
| Tables_in_INFORMATION_SCHEMA |
+-----+
| CHARACTER_SETS                |
| COLLATIONS                    |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| COLUMNS                      |
| COLUMN_PRIVILEGES             |
| ENGINES                      |
| EVENTS                        |
| FILES                         |
| GLOBAL_STATUS                 |
| GLOBAL_VARIABLES              |
| KEY_COLUMN_USAGE              |
| PARTITIONS                    |
| PLUGINS                       |
| PROCESSLIST                   |
| REFERENTIAL_CONSTRAINTS      |
| ROUTINES                      |
| SCHEMATA                      |
| SCHEMA_PRIVILEGES             |
| SESSION_STATUS                |
| SESSION_VARIABLES             |
| STATISTICS                    |
| TABLES                       |
| TABLE_CONSTRAINTS           |
| TABLE_PRIVILEGES             |
| TRIGGERS                      |
| USER_PRIVILEGES               |
| VIEWS                         |
+-----+
27 rows in set (0.00 sec)
```

`SHOW COLUMNS` and `DESCRIBE` can display information about the columns in individual `INFORMATION_SCHEMA` tables.

`SHOW` statements that accept a `LIKE` clause to limit the rows displayed also allow a `WHERE` clause that enables specification of more general conditions that selected rows must satisfy:

```
SHOW CHARACTER SET
SHOW COLLATION
SHOW COLUMNS
SHOW DATABASES
SHOW FUNCTION STATUS
SHOW INDEX
SHOW OPEN TABLES
SHOW PROCEDURE STATUS
SHOW STATUS
SHOW TABLE STATUS
SHOW TABLES
SHOW TRIGGERS
SHOW VARIABLES
```

The `WHERE` clause, if present, is evaluated against the column names displayed by the `SHOW` statement. For example, the `SHOW CHARACTER SET` statement produces these output columns:

```
mysql> SHOW CHARACTER SET;
+-----+-----+-----+-----+
| Charset | Description | Default collation | Maxlen |
+-----+-----+-----+-----+
| big5    | Big5 Traditional Chinese | big5_chinese_ci | 2 |
| dec8   | DEC West European | dec8_swedish_ci | 1 |
| cp850  | DOS West European | cp850_general_ci | 1 |
| hp8    | HP West European | hp8_english_ci | 1 |
| koi8r  | KOI8-R Relcom Russian | koi8r_general_ci | 1 |
| latin1 | cp1252 West European | latin1_swedish_ci | 1 |
| latin2 | ISO 8859-2 Central European | latin2_general_ci | 1 |
| ...    | ... | ... | ... |
+-----+-----+-----+-----+
```

To use a `WHERE` clause with `SHOW CHARACTER SET`, you would refer to those column names. As an example, the following

statement displays information about character sets for which the default collation contains the string `'japanese'`:

```
mysql> SHOW CHARACTER SET WHERE `Default collation` LIKE '%japanese%';
```

Charset	Description	Default collation	Maxlen
ujis	EUC-JP Japanese	ujis_japanese_ci	3
sjis	Shift-JIS Japanese	sjis_japanese_ci	2
cp932	SJIS for Windows Japanese	cp932_japanese_ci	2
eucjpms	UJIS for Windows Japanese	eucjpms_japanese_ci	3

This statement displays the multi-byte character sets:

```
mysql> SHOW CHARACTER SET WHERE Maxlen > 1;
```

Charset	Description	Default collation	Maxlen
big5	Big5 Traditional Chinese	big5_chinese_ci	2
ujis	EUC-JP Japanese	ujis_japanese_ci	3
sjis	Shift-JIS Japanese	sjis_japanese_ci	2
euckr	EUC-KR Korean	euckr_korean_ci	2
gb2312	GB2312 Simplified Chinese	gb2312_chinese_ci	2
gbk	GBK Simplified Chinese	gbk_chinese_ci	2
utf8	UTF-8 Unicode	utf8_general_ci	3
ucs2	UCS-2 Unicode	ucs2_general_ci	2
cp932	SJIS for Windows Japanese	cp932_japanese_ci	2
eucjpms	UJIS for Windows Japanese	eucjpms_japanese_ci	3

---

## Chapter 28. MySQL 5.0 FAQ — INFORMATION\_SCHEMA

### Questions

- [28.1](#): Where can I find documentation for the MySQL `INFORMATION_SCHEMA` database?
- [28.2](#): Is there a discussion forum for `INFORMATION_SCHEMA`?
- [28.3](#): Where can I find the ANSI SQL 2003 specification for `INFORMATION_SCHEMA`?
- [28.4](#): What is the difference between the Oracle Data Dictionary and MySQL's `INFORMATION_SCHEMA`?
- [28.5](#): Can I add to or otherwise modify the tables found in the `INFORMATION_SCHEMA` database?

### Questions and Answers

#### **28.1: Where can I find documentation for the MySQL `INFORMATION_SCHEMA` database?**

See [INFORMATION\\_SCHEMA Tables](#)

#### **28.2: Is there a discussion forum for `INFORMATION_SCHEMA`?**

See <http://forums.mysql.com/list.php?101>.

#### **28.3: Where can I find the ANSI SQL 2003 specification for `INFORMATION_SCHEMA`?**

Unfortunately, the official specifications are not freely available. (ANSI makes them available for purchase.) However, there are books available — such as *SQL-99 Complete, Really* by Peter Gultzan and Trudy Pelzer — which give a comprehensive overview of the standard, including `INFORMATION_SCHEMA`.

#### **28.4: What is the difference between the Oracle Data Dictionary and MySQL's `INFORMATION_SCHEMA`?**

Both Oracle and MySQL provide metadata in tables. However, Oracle and MySQL use different table names and column names. MySQL's implementation is more similar to those found in DB2 and SQL Server, which also support `INFORMATION_SCHEMA` as defined in the SQL standard.

#### **28.5: Can I add to or otherwise modify the tables found in the `INFORMATION_SCHEMA` database?**

No. Since applications may rely on a certain standard structure, this should not be modified. For this reason, *we cannot support bugs or other issues which result from modifying `INFORMATION_SCHEMA` tables or data.*