Oracle® SQL Developer

Supplementary Information for MySQL Migrations

Release 3.0

E18461-03

March 2011

This document contains information for migrating from MySQL to Oracle. It supplements the information about migration in *Oracle SQL Developer User's Guide*.

ORACLE

Oracle SQL Developer Supplementary Information for MySQL Migrations, Release 3.0

E18461-03

Primary Author: Chuck Murray

Contents

Preface

Oracle SQL Developer Supplementary Information for MySQL Migrations describes several differences between MySQL and Oracle. It also outlines how those differences are dealt with by SQL Developer during the migration process.

Audience

This guide is intended for anyone who is involved in converting a MySQL database to Oracle using SQL Developer.

You should be familiar with relational database concepts and with the operating system environments under which you are running Oracle and MySQL.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/us/corporate/accessibility/index.html`.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/support/contact.html` or visit `http://www.oracle.com/accessibility/support.html` if you are hearing impaired.

Related Documents

For information about using Oracle SQL Developer, see and the SQL Developer online help.

For information about installing Oracle SQL Developer, see *Oracle SQL Developer Installation Guide*.

Oracle error message documentation is only available in HTML. If you only have access to the Oracle Documentation CD, you can browse the error messages by range. Once you find the specific range, use your browser's "find in page" feature to locate the specific message. When connected to the Internet, you can search for a specific error message using the error message search feature of the Oracle online documentation.

To download free release notes, installation documentation, white papers, or other collateral, go to the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

http://www.oracle.com/technology/membership

If you already have a user name and password for OTN, then you can go directly to the documentation section of the OTN Web site at

http://www.oracle.com/technology/documentation

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 Introduction

This document provides supplementary reference information for migrating MySQL databases to Oracle using the Oracle SQL Developer tool. It includes information to help you plan for the migration and to solve any problems that might occur during or after the migration.

You should already know how to use the SQL Developer tool, including its migration capabilities. SQL Developer is described in *Oracle SQL Developer User's Guide* and in the online help.

The following supplementary reference information is available:

- Chapter 2, "Oracle and MySQL Compared"
- Chapter 3, "Triggers and Stored Procedures"
- Chapter 4, "Troubleshooting"

2

Oracle and MySQL Compared

This chapter compares the MySQL database and the Oracle database. It includes the following sections:

- Database Security
- Schema Migration
- Data Types
- Data Storage Concepts

2.1 Database Security

This section includes information about security issues with MySQL databases and Oracle databases.

As with Oracle, MySQL users are maintained by the database. MySQL uses a set of grant tables to keep track of users and the privileges that they can have. MySQL uses these grant tables when performing authentication, authorization and access control for users.

2.1.1 Database Authentication

Unlike Oracle (when set up to use database authentication) and most other databases that use only the user name and password to authenticate a user, MySQL uses an additional `location` parameter when authenticating a user. This `location` parameter is usually the host name, IP address, or a wildcard ("%"). With this additional parameter, MySQL may further restrict a user access to the database to a particular host or hosts in a domain. Moreover, this also allows a different password and set of privileges to be enforced for a user depending on the host from which the connection is made. Thus, user scott, who logs on from abc.com may or may not the same as user scott who logs on from xyz.com.

2.1.2 Privileges

The MySQL privilege system is a hierarchical system that works through inheritance. Privileges granted at a higher level are implicitly passed down to all lower levels and may be overridden by the same privileges set at lower levels. MySQL allows privileges to be granted at five different levels, in descending order of the scope of the privileges:

- Global
- Per-host basis
- Database-level

- Table-specific
- Column-specific (single column in a single table

Each level has a corresponding grant table in the database. When performing a privilege check, MySQL checks each of the tables in descending order of the scope of the privileges, and the privileges granted at a lower level take precedence over the same privileges granted at a higher level.

The privileges supported by MySQL are grouped into two types: administrative privileges and per-object privileges. The administrative privileges are global privileges that have server-wide effects and are concerned with the functioning of MySQL. These administrative privileges include the FILE, PROCESS, REPLICATION, SHUTDOWN and SUPER privilege. The per-object privileges affect database objects such tables, columns, indexes, and stored procedures, and can be granted with a different scope. These per-object privileges are named after the SQL queries that trigger their checks.

Unlike in Oracle, there is no concept of role in MySQL. Thus, in order to grant a group of users the same set of privileges, the privileges have to be granted to each user separately. Alternately, though less satisfactory for auditing, users performing tasks as a role may all share a single user account that is designated for the "role" and with the required privileges granted.

2.2 Schema Migration

The schema contains the definitions of the tables, views, indexes, users, constraints, stored procedures, triggers, and other database-specific objects. Most relational databases work with similar objects.

This section contains the following:

2.2.1 Schema Object Similarities

There are many similarities between schema objects in Oracle and MySQL. However, some schema objects differ between these databases. For more information about schema objects, see *Oracle SQL Reference*.

Table 2–1 shows the differences between Oracle and MySQL.

Table 2–1 Schema Objects in Oracle and MySQL

Oracle	MySQL
AFTER trigger	trigger
BEFORE trigger	trigger
Check constraint	Check constraint
Column default	Column default
Database	Database
Foreign key	Foreign key

Table 2–1 (Cont.) Schema Objects in Oracle and MySQL

Oracle	MySQL
Index	Index
Package	N/A
PL/SQL function	Routine
PL/SQL procedure	Routine
Primary key	Primary key
Role	N/A
Schema	Schema
Sequence	AUTO_INCREMENT for a column
Snapshot	N/A
Synonym	N/A
Table	Table
Tablespace	N/A
Temporary table	Temporary table
Trigger for each row	Trigger for each row
Unique key	Unique key
User	User
View	View

2.2.2 Schema Object Names

Oracle is case insensitive to object names, and Oracle schema object names are stored as uppercase.

As in Oracle, column, index, stored procedure, and trigger names as well as column aliases in MySQL are case insensitive on all platforms. However, the case sensitivity of database and tables names for MySQL differs from Oracle. In MySQL, databases correspond to directories within the data directory, and tables correspond to one or more files within the database directory. As such, the case sensitivity of the database and table names is determined by the case sensitivity of the underlying operating systems. This means that database and table names are not case-sensitive in Windows and are case-sensitive in most varieties of Unix. However, MySQL allows users to determine how the database and table names are stored on disk and in their use in MySQL through the `lower_case_table_names` system variable. Table aliases are case-sensitive in releases before MySQL 4.1.1.

Both Oracle and MySQL let you use reserved words as object names by representing the name with a quoted identifier. However, MySQL allows some reserved words such as DATE and TIMESTAMP to be used as unquoted identifier for object names, although this is not allowed in Oracle. SQL Developer appends an underscore (_) to the name of a MySQL object that is an Oracle reserved word.

MySQL and Oracle have some minor differences in their definition of an identifier. In MySQL, an unquoted identifier may begin with a digit, and double quotation marks are allowed in a quoted identifier; however, neither of these is allowed in an Oracle identifier. In MySQL, the quote character is the backtick (`). If the SQL mode ANSI_QUOTES is set, double quotes can also be used to quote the identifiers. In Oracle, identifiers are quoted using double quotation marks.

You should choose a schema object name that is unique by case and by at least one other characteristic, and ensure that the object name is not a reserved word from either database.

2.2.3 Table Design Considerations

This section discusses table design issues that you need to consider when converting MySQL databases to Oracle. This section includes the following:

- Character Data Types
- Column Default Value

2.2.3.1 Character Data Types

MySQL and Oracle have some differences in the character types that they support and in the way they store and retrieve the character type values.

MySQL supports the CHAR and VARCHAR type for character type with a length that is less than 65,535 bytes. The CHAR type can have a maximum length of 255 bytes, and as of MySQL 3.23 it may also be declared with a length of 0 byte. Before MySQL 5.0.3, the length specification for the VARCHAR type is the same as the CHAR type. From MySQL 5.0.3 on, the maximum length for the VARCHAR type is 65,535 bytes. Oracle supports four character types: CHAR, NCHAR, NVARCHAR2 and VARCHAR2. The minimum length that can be declared for all Oracle character types is 1 byte. The maximum size allowed for CHAR and NCHAR is 2,000 bytes, and for NVARCHAR2 and VARCHAR2 it is 4,000 bytes.

MySQL CHAR values are right-padded with spaces to the specified length when they are stored, and trailing spaces are removed when the values are retrieved. On the other hand, VARCHAR values are stored using as many characters as are given, but before MySQL 5.0.3 trailing spaces are removed when the values are stored and retrieved. Oracle blank-pads the value for its CHAR and NCHAR type to the column length if the value is shorter than the column length, and trailing spaces are not removed on retrieval. For NVARCHAR2 and VARVHAR2 data type columns, Oracle stores and retrieves the value exactly as is given, including trailing spaces.

If a value is assigned to a character type column that exceeds its specified length, MySQL truncates the value and does not generate an error unless the STRICT SQL mode is set. Oracle generates an error if the value assigned to a character type column exceeds its specified length.

In MySQL, every character type (CHAR, VARCHAR, and TEXT) column has a column character set and collation. If the character set or collation is not explicitly defined in the column definition, the table character set or collation is implied if specified; otherwise, the database character or collation is chosen. In Oracle, the character set for CHAR and VARCHAR2 types is defined by the database character set, and for the character set for NCHAR and NVARCHAR types is defined the national character set.

When declaring a CHAR or VARCHAR type in MySQL, the default length semantics is characters instead of bytes for MySQL 4.1 and later. In Oracle, the default length semantics is bytes for CHAR and VARCHAR2 types and characters for NCHAR and NVARCHAR2 types.

SQL Developer will map MySQL CHAR and VARCHAR types to Oracle CHAR and VARCHAR2 types, respectively. SQL Developer will determine the maximum number of bytes for the Oracle CHAR and VARCHAR2 data type columns from the number of bytes required to hold the maximum length specified for the corresponding MySQL CHAR and VARCHAR data type columns. If the MySQL VARCHAR2 column is such

that the data exceeds 4000 bytes, convert the column to an Oracle CLOB data type column.

2.2.3.2 Column Default Value

MySQL differs from Oracle in the way it handles default value for a column that does not allow NULL value.

In MySQL, for a column that does not allow NULL value and for which no data is provided for the column when data is inserted into the table, MySQL determines a default value for the column. This default value is the implicit default value for the column data type. However, if the strict mode is enabled, MySQL generates errors, and for transactional tables it rolls back the insert statement.

In Oracle, when data is inserted into a table, data must be provided for all columns that do not allow NULL value. Oracle does not generate a default value for columns that have the NOT NULL constraint.

2.2.4 Migrating Multiple Databases

SQL Developer supports the migration of multiple MySQL databases if they are on the same MySQL database server.

2.2.5 Schema Migration Considerations for MySQL

Schema migration considerations for MySQL apply in the following areas:

- Databases
- Mapping MySQL Global and Database-Level Privileges to Oracle System Privileges
- Temporary Tables
- Owner of Schema Objects

2.2.5.1 Databases

When migrating MySQL databases to Oracle, SQL Developer maps each MySQL database to a tablespace in Oracle. Database objects, such as tables, indexes and views are stored in the respective tablespaces and are referenced from the Oracle schema for the user that owns them.

2.2.5.2 Mapping MySQL Global and Database-Level Privileges to Oracle System Privileges

SQL Developer does not process all the administrative privileges on MySQL, except the SUPER privilege. Table 2–2 shows the mappings for MySQL per-object privileges granted at the different levels as well as the SUPER privilege granted at the global level.

Table 2–2 MySQL Privileges and Oracle System Privileges

Level	Privilege	System Privilege(s) on Oracle
Global	ALTER	ALTER ANY TABLE, ALTER ANY SEQUENCE, ALTER ANY CUSTER, COMMENT ANY TABLE
Global	ALTER ROUTINE	ALTER ANY PROCEDURE, DROP ANY PROCEDURE

Table 2–2 (Cont.) MySQL Privileges and Oracle System Privileges

Level	Privilege	System Privilege(s) on Oracle
Global	CREATE	CREATE ANY TABLE, CREATE ANY SEQUENCE, CREATE ANY CLUSTER, CREATE DATABASE LINK, COMMENT ANY TABLE
Global	CREATE ROUTINE	CREATE ANY PROCEDURE
Global	CREATE USER	CREATE USER, GRANT ANY PRIVILEGE
Global	CREATE VIEW	CREATE ANY VIEW
Global	DELETE	ALTER ANY TABLE, DROP USER, DELETE ANY TABLE
Global	DROP	DROP ANY TABLE, DROP ANY SEQUENCE, DROP ANY CLUSTER, DROP ANY VIEW
Global	EXECUTE	EXECUTE ANY PROCEDURE
Global	INDEX	CREATE ANY INDEX, ALTER ANY INDEX, DROP ANY INDEX
Global	INSERT	INSERT ANY TABLE
Global	LOCK TABLES	LOCK ANY TABLE
Global	SELECT	SELECT ANY TABLE
Global	SUPER	CREATE ANY TRIGGER, DROP ANY TRIGGER
Global	UPDATE	UPDATE ANY TABLE
Global	USAGE	CREATE SESSION, ALTER SESSION, UNLIMITED TABLESPACE
Database	CREATE	CREATE CLUSTER, CREATE DATABASE LINK, CREATE SEQUENCE, CREATE TABLE
Database	CREATE ROUTINE	CREATE PROCEDURE
Database	CREATE VIEW	CREATE VIEW
Table	CREATE	CREATE TABLE
Table	CREATE VIEW	CREATE VIEW

2.2.5.3 Temporary Tables

SQL Developer does not support the migration of temporary tables.

In MySQL, temporary tables are database objects that are visible only to the current user session and are automatically dropped when the user session ends.

The definition of temporary tables in Oracle differs slightly from MySQL, in that the temporary tables, once created, exist until they are explicitly dropped and they are visible to all sessions with appropriate privileges. However, the data in the temporary tables is visible only to the user session that inserts the data into the table, and the data may persist for the duration of a transaction or a user session.

2.2.5.4 Owner of Schema Objects

SQL Developer creates an Oracle schema for the MySQL root user that owns, for all databases to be migrated, all database objects except stored procedures. For stored procedures, the MySQL users that created them remain the owner. SQL Developer creates an Oracle schema for each MySQL user that is migrated.

2.3 Data Types

This section describes the data types used within Oracle. It shows the MySQL data types and the Oracle equivalent. It includes information about the following:

- Supported Oracle Data Types
- Default Data Type Mappings
- Comparing Data Types

2.3.1 Supported Oracle Data Types

Table 2–3 describes the Oracle data types supported by Oracle SQL Developer.

Table 2–3 Supported Oracle Data Types

Data Type	Description
BLOB	A binary large object. Maximum size is 4 gigabytes.
CHAR (SIZE)	Fixed-length character data of length size bytes. Maximum size is 2000 bytes. Default and minimum size is 1 byte.
CLOB	A character large object containing single-byte characters. Both fixed-width and variable-width character sets are supported, both using the CHAR database character set. Maximum size is 4 gigabytes.
DATE	The DATE data type stores date and time information. Although date and time information can be represented in both CHAR and NUMBER data types, the DATE data type has special associated properties. For each DATE value, Oracle stores the following information: century, year, month, day, hour, minute, and second.
FLOAT	Specifies a floating-point number with decimal precision 38, or binary precision 126.
LONG (SIZE)	Character data of variable length up to 2 gigabytes, or 2^31 -1 bytes.
LONG RAW	Raw binary data of variable length up to 2 gigabytes.
NCHAR (SIZE)	Fixed-length character data of length size characters or bytes, depending on the choice of national character set. Maximum size is determined by the number of bytes required to store each character, with an upper limit of 2000 bytes. Default and minimum size is 1 character or 1 byte, depending on the character set.
NCLOB	A character large object containing multibyte characters. Both fixed-width and variable-width character sets are supported, both using the NCHAR database character set. Maximum size is 4 gigabytes. Stores national character set data.
NUMBER	Number having precision p and scale s. The precision p can range from 1 to 38. The scale s can range from -84 to 127.
NVARCHAR2 (SIZE)	Variable-length character string having maximum length size characters or bytes, depending on the choice of national character set. Maximum size is determined by the number of bytes required to store each character, with an upper limit of 4000 bytes. You must specify size for NVARCHAR2.
RAW (SIZE)	Raw binary data of length size bytes. Maximum size is 2000 bytes. You must specify size for a RAW value.

Table 2–3 (Cont.) Supported Oracle Data Types

Data Type	Description
VARCHAR (SIZE)	The VARCHAR data type is currently synonymous with the VARCHAR2 data type. Oracle recommends that you use VARCHAR2 rather than VARCHAR. In the future, VARCHAR might be defined as a separate data type used for variable-length character strings compared with different comparison semantics. The maximum size is 4000 and the minimum of 1 is the default.
BINARY_DOUBLE	A 64-bit, double-precision floating-point number data type.
BINARY_FLOAT	A 32-bit, single-precision floating-point number data type.

For more information about Oracle data types, see *Oracle Database SQL Language Reference*.

2.3.2 Default Data Type Mappings

Table 2–4 shows the default settings used by SQL Developer to convert data types from MySQL to Oracle. SQL Developer enables you to change the default setting for certain data types by specifying an alternative type. For information about changing the default data type mappings, see the SQL Developer online help.

Table 2–4 Default Data Type Mappings Used by Oracle SQL Developer

MySQL Data Type	Oracle Data Type
BIGINT	NUMBER(19, 0)
BIT	RAW
BLOB	BLOB, RAW
CHAR	CHAR
DATE	DATE
DATETIME	DATE
DECIMAL	FLOAT (24)
DOUBLE	FLOAT (24)
DOUBLE PRECISION	FLOAT (24)
ENUM	VARCHAR2
FLOAT	FLOAT
INT	NUMBER(10, 0)
INTEGER	NUMBER(10, 0)
LONGBLOB	BLOB, RAW
LONGTEXT	CLOB, RAW
MEDIUMBLOB	BLOB, RAW
MEDIUMINT	NUMBER(7, 0)
MEDIUMTEXT	CLOB, RAW
NUMERIC	NUMBER
REAL	FLOAT (24)
SET	VARCHAR2

Table 2–4 (Cont.) Default Data Type Mappings Used by Oracle SQL Developer

MySQL Data Type	Oracle Data Type
SMALLINT	NUMBER(5, 0)
TEXT	VARCHAR2, CLOB
TIME	INTERVAL DAY TO SECOND
TIMESTAMP	DATE
TINYBLOB	RAW
TINYINT	NUMBER(3, 0)
TINYTEXT	VARCHAR2
VARCHAR	VARCHAR2, CLOB
YEAR	NUMBER

Note: The ENUM and SET data types have no direct mapping in Oracle. SQL Developer maps ENUM columns in MySQL to VARCHAR2 columns in Oracle. It then adds a constraint and a trigger to those columns to ensure that only values that were allowed by the ENUM data type are allowed in the column it was mapped to in Oracle.

2.3.3 Comparing Data Types

This section lists the difference between MySQL and Oracle data types. For some MySQL data types there is more than one alternative Oracle data type. The tables include information about the following:

- Numeric Types
- Date and Time Types
- String Types

2.3.3.1 Numeric Types

When mapping MySQL data types to numeric data types in Oracle, the following conditions apply:

- If there is no precision or scale defined for the destination Oracle data type, precision and scale are taken from the MySQL source data type.
- If there is a precision or scale defined for the destination data type, these values are compared to the equivalent values of the source data type and the maximum value is selected.

The following table compares the numeric types of MySQL to Oracle:

MySQL	Size	Oracle
BIGINT	8 Bytes	NUMBER (19,0)
BIT	approximately (M+7)/8 Bytes	RAW
DECIMAL(M,D)	M+2 bytes if D > 0, M+1 bytes if D = 0 (D+2, if M < D)	FLOAT(24), BINARY_FLOAT

MySQL	Size	Oracle
DOUBLE	8 Bytes	FLOAT(24), BINARY_FLOAT, BINARY_DOUBLE
DOUBLE PRECION	8 Bytes	FLOAT(24), BINARY_DOUBLE
FLOAT(25<=X <=53)	8 Bytes	FLOAT(24), BINARY_FLOAT
FLOAT(X<=24)	4 Bytes	FLOAT, BINARY_FLOAT
INT	4 Bytes	NUMBER (10,0)
INTEGER	4 Bytes	NUMBER (10,0)
MEDIUMINT	3 Bytes	NUMBER (7,0)
NUMERIC	M+2 bytes if D > 0, M+1 bytes if D = 0 (D+2, if M < D)	NUMBER
REAL	8 Bytes	FLOAT(24), BINARY_FLOAT
SMALLINT	2 Bytes	NUMBER(5,0)
TINYINT	1 Byte	NUMBER(3,0)

2.3.3.2 Date and Time Types

The following table compares the date and time types of MySQL to Oracle:

MySQL	Size	Oracle
DATE	3 Bytes	DATE
DATETIME	8 Bytes	DATE
TIMESTAMP	4 Bytes	DATE
TIME	3 Bytes	DATE
YEAR	1 Byte	NUMBER

2.3.3.3 String Types

When mapping MySQL data types to character data types in Oracle, the following conditions apply:

- If there is no length defined for the destination data type, the length is taken from the source data type.
- If there is a length defined for the destination data type, the maximum value of the two lengths is taken.

The following table compares the string types of MySQL to Oracle:

> **Note:** Reference to M indicates the maximum display size. The maximum legal display size is 255. A reference to L applies to a floating point types and indicates the number of digits following the decimal point.

MySQL	Size	Oracle
BLOB	L + 2 Bytes whereas L<2^16	RAW, BLOB
CHAR(m)	M Bytes, 0<=M<=255	CHAR

MySQL	Size	Oracle
ENUM (VALUE1, VALUE2, ...)	1 or 2 Bytes depending on the number of enum. values (65535 values max)	
LONGBLOB	L + 4 Bytes whereas L < 2 ^ 32	RAW, BLOB
LONGTEXT	L + 4 Bytes whereas L < 2 ^ 32	RAW, CLOB
MEDIUMBLOB	L + 3 Bytes whereas L < 2^ 24	RAW, BLOB
MEDIUMTEXT	L + 3 Bytes whereas L < 2^ 24	RAW, CLOB
SET (VALUE1, VALUE2, ...)	1, 2, 3, 4 or 8 Bytes depending on the number of set members (64 members maximum)	
TEXT	L + 2 Bytes whereas L<2^16	VARCHAR2, CLOB
TINYBLOB	L + 1 Bytes whereas L<2 ^8	RAW, BLOB
TINYTEXT	L + 1 Bytes whereas L<2 ^8	VARCHAR2
VARCHAR(m)	L+1 Bytes whereas L<=M and0<=M<=255 before MySQL 5.0.3 (0 <= M <= 65535 in MySQL 5.0.3 and later; effective maximum length is 65,532 bytes)	VARCHAR2, CLOB

2.4 Data Storage Concepts

This section provide a description of the conceptual differences and similarities in data storage for MySQL and Oracle databases.

Data storage is an aspect of MySQL that sets it apart for nearly every database, including Oracle. In MySQL, databases correspond to directories within the data directory of the server. Tables within a database correspond to one or more files within the database directory, depending on the storage engine used for the tables.

A database can contain a mix of tables of different storage engines. A storage engine is responsible for the storage and retrieval of the data for a table.

MySQL offers a variety of storage engines (formerly called table types) to meet the different requirements of the user's environment. Table 2–5 shows the storage engines supported by MySQL.

Table 2–5 Storage Engines Supported by MySQL

Storage Engine	Description
MyISAM	The default non-transactional storage engine that provides full-text indexing and is highly portable
MERGE	A non-transactional storage engine that allows a collection of MyISAM tables with identical column and index information to be used as one
MEMORY (HEAP)	A non-transactional storage engine that stores data in memory
BDB (Berkeley DB)	The first transactional-safe storage engine
InnoDB	A transactional-safe storage engine designed for maximum performance when processing large volume of data and that provides row-level locking
FEDERATED	A storage engine that accesses data in tables of remote databases rather than in local tables

Table 2–5 (Cont.) Storage Engines Supported by MySQL

Storage Engine	Description
ARCHIVE	A storage engine that can store large amount of data without indexes in very small footprint
CSV	A storage engine that stores data in text file using comma-separated-values format
BLACKHOLE	A storage engine that acts as a "black hole" that accepts data but throws it away and does not store it
EXAMPLE	A "stub" engine that does nothing. Its purpose is to serve as an example that illustrates how to begin writing new engines.
ISAM	The original MySQL storage engine that has been deprecated in favor of the MyISAM storage engine as of version 5.0

Each storage engine has its benefits and drawbacks. Some of features that differentiate the storage engines are transaction, locking, concurrency and portability. The following table summarizes the features for four of the commonly used storage engines.

Table 2–6 Feature Comparison for Common Storage Engines

Feature	MyISAM	Heap	BDB	InnoDB
Transactional	No	No	Yes	Yes
Lock granularity	Table	Table	Page	Row
Storage	A data file (.MYD) and an index file (.MYI) for each table	In-memory	A single data and index file (.db) for each table	A set of data files for all the tables
Portable	Yes	N/A	No	Yes

An Oracle database consists of one or more tablespaces. Tablespaces provide logical storage space that link a database to the physical disks that hold the data. A tablespace is created from one or more data files. Data files are files in the file system or an area of disk space specified by a raw device. A tablespace can be enlarged by adding more data files.

An Oracle database consists of a least a SYSTEM tablespace, where the Oracle tables are stored. It can also consist of user defined tablespaces. A tablespace is the logical storage location for database objects. For example, you can specify where a particular table or index gets created in the tablespace.

3

Triggers and Stored Procedures

This chapter compares MySQL and Oracle triggers and stored procedures. (The information in this chapter applies only to MySQL release 5, not to earlier releases.) For more information about Oracle triggers and stored procedures, see *Oracle Database PL/SQL Language Reference*. This chapter includes the following sections:

3.1 Triggers

Triggers are named database objects that are implicitly fired when a triggering event occurs. The trigger action can be run before or after the triggering event. Triggers are similar to stored procedures but differ in the way that they are invoked.

Support for triggers in MySQL is only included beginning with release 5.0.2. A trigger can only be associated with a table and defined to fire when an INSERT, DELETE or UPDATE statement is performed on the table. MySQL does not permit two triggers with the same trigger timing (BEFORE or AFTER) and trigger event or statement (INSERT, DELETE, or UPDATE) to be defined on a table. For example, you cannot define two BEFORE INSERT or two AFTER UPDATE triggers for a table. All triggers defined on MySQL are row triggers, which means that the action defined for the triggers is executed for each row affected by the triggering statement.

Error handling during trigger execution for transactional tables ensures that either both the triggering statement and trigger action is completed successfully or neither the trigger statement nor the trigger action is executed, that is all changes made are rollback on failure. For non-transactional tables, all changes made prior to the point of error remains in effect.

The following is the syntax to create a trigger in MySQL:

```
CREATE TRIGGER <trigger name>
  { BEFORE | AFTER }
  { INSERT | UPDATE | DELETE }
  ON <table name>
  FOR EACH ROW
  <triggered action>
```

In Oracle, triggers can be fired when one of the following operations occurs:

- DML statements (INSERT, DELETE or UPDATE) that modify data on a table or view
- DDL statements

- User events such as logon and logoff
- System events such as startup, shutdown, and error messages

Oracle allows multiple triggers with the same trigger timing and trigger event to be defined on a table; however, these triggers are not guaranteed to execute in any specific order. Triggers can be defined as row triggers or statement triggers. Statement triggers are fired once for each triggering statement regardless of the number of rows in a table affected by the triggering statement. For example if a DELETE statement deletes several rows from a table, a statement trigger is only fired once.

The execution model for Oracle triggers is transactional. All actions performed as a result of the triggering statement, including the actions performed by fired triggers, must all succeed; otherwise, they are rolled back.

3.2 Stored Procedures

Stored procedures provide a powerful way to code application logic that can be stored on the server. MySQL and Oracle both use stored procedures and functions. Stored functions are similar to procedures, except that a function returns a value to the environment in which it is called. In MySQL, stored procedures and functions are collectively called routines.

The following sections compare stored procedures in MySQL and Oracle:

- Individual SQL Statements
- Variables in Stored Procedures
- Error Handling in Stored Procedures

3.2.1 Individual SQL Statements

This section describes considerations related to the following statements or constructs:

- REPLACE Statement
- DO Statement
- Compound DECLARE Statement
- Compound SET Statement

3.2.1.1 REPLACE Statement

The REPLACE statement in MySQL is a dual-purpose statement. It works like the INSERT statement when there is no record in the table that has the same value as the new record for a primary key or a unique index, and otherwise it works like the UPDATE statement.

Oracle does not have any built-in SQL statements that supports the purposes of the MySQL REPLACE statement. To convert this statement to Oracle, an emulated function using both the INSERT and UPDATE statements has to be created. An attempt is first made to place the data into the table using the INSERT statement; and if this fails, the data in the table is then updated using the UPDATE statement.

3.2.1.2 DO Statement

As its name implies, the DO statement in MySQL does something but does not return anything; specifically, it executes the comma-delimited list of expressions specified as its parameters. The DO statement is converted to a `SELECT expr1 [, expr2,…] INTO … FROM DUAL` statement in Oracle.

3.2.1.3 Compound DECLARE Statement

MySQL uses the DECLARE statement to declare local variables in stored procedures. PL/SQL does not allow multiple declarations; each declaration must be made separately. To convert compound DECLARE statements into functionally equivalent PL/SQL code, each MySQL multiple declaration statement should be converted into logically equivalent separate statements, one for each declaration.

For example, consider the following MySQL simple declaration and multiple declaration statements:

```
/* Simple declaration */
DECLARE a INT;

/* Compound declaration */
DECLARE a, b INT DEFAULT 5;
```

The PL/SQL functionally equivalent statements are:

```
/* Simple declaration */
a INT;

/* Multiple declarations */
a INT := 5;
b INT := 5;
```

In this example, the two original MySQL DECLARE statements are converted into three logically equivalent PL/SQL declaration statements, with one PL/SQL declaration statement for every declaration used within the MySQL DECLARE statements.

3.2.1.4 Compound SET Statement

MySQL uses the SET statement to assign values to variables (user variables or system variables). MySQL allows compound statements that assign values to two or more variables within the same statement. PL/SQL allows only simple assignments that assign a single value to a single variable. To convert compound SET statements into functionally equivalent PL/SQL code, split each MySQL multiple assignment statement into logically equivalent simple assignment statements.

For example, consider the following MySQL simple assignment and multiple assignment statements:

```
/* Simple statement */
SET a:=1;

/* Compound statement*/
SET x:=1, y:=0;
```

The PL/SQL functionally equivalent statements are:

```
/* Simple statement */
a:=1;

/* Multiple statements */
x:=1;
y:=0;
```

In this example, the two original MySQL SET statements are converted into three logically equivalent PL/SQL assignment statements, with one PL/SQL assignment statement for every declaration used within the MySQL SET statements.

3.2.2 Variables in Stored Procedures

MySQL supports three types of variables in stored procedures: local variables, user variables, and system variables.

Local variables are declared within stored procedures and are only valid within the BEGIN…END block where they are declared. Local variables must be declared within a BEGIN…END block before they can be referenced in other statements in the block, including any nested BEGIN…END blocks. If a local variable declared within a nested BEGIN…END block has the same name as a local variable declared in its enclosing BEGIN…END block, the local variable in the nested block takes precedence wherever the local variable is referenced in the nested BEGIN…END block. Local variables can have any SQL data type. The following example shows the use of local variables in a stored procedure.

```
CREATE PROCEDURE p1()
BEGIN
   /* declare local variables */
   DECLARE x INT DEFAULT 0;
   DECLARE y, z INT;

   /* using the local variables */
   SET x := x + 100;
   SET y := 2;
   SET z := x + y;

   BEGIN
      /* local variable in nested block */
      DECLARE z INT;

      SET z := 5;

      /* local variable z takes precedence over the one of the
         same name declared in the enclosing block. */
      SELECT x, y, z;
   END;

   SELECT x, y, z;
END;

mysql> call p1();
+-----+---+---+
| x   | y | z |
+-----+---+---+
| 100 | 2 | 5 |
+-----+---+---+
1 row in set (0.00 sec)

+-----+---+-----+
| x   | y | z   |
+-----+---+-----+
| 100 | 2 | 102 |
+-----+---+-----+
1 row in set (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
```

User variables are specific to a user session and cannot be seen or used by other users. They are valid only for the duration of a user session and are automatically freed when the user session ends. User variables have a session-scope; thus, all references to a user variable of the same name within a session refer to the same variable. In MySQL

stored procedures, user variables are referenced with an ampersand (@) prefixed to the user variable name (for example, `@x` and `@y`). The following example shows the use of user variables in two stored procedures.

```
CREATE PROCEDURE p2()
BEGIN
   SET @a = 5;
      SET @b = 5;
      SELECT @a, @b;
   END;

CREATE PROCEDURE p3()
BEGIN
   SET @a = @a + 10;
      SET @b = @b - 5;
      SELECT @a, @b;
   END;

mysql> call p2();
+------+------+
| @a   | @b   |
+------+------+
| 5    | 5    |
+------+------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> call p3();
+------+------+
| @a   | @b   |
+------+------+
| 15   | 0    |
+------+------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

In the second procedure (p3) in the preceding example, the use of the user variables `a` and `b` on the right-hand side of the assignment statement assumed that the variables have previously been initialized to some value by the first procedure. If the variables have not been initialized by the first procedure, they would have null values, and the result for the assignment would also be a null value.

System variables can also be referenced in MySQL stored procedures. There are two kinds of system variable in MySQL: global system variables and session system variables. Global system variables affect the operation of the overall server. Session system variables affect the individual user session. System variables that are dynamic can also be changed in MySQL stored procedures.

In a SET statement, variables name preceded by `GLOBAL` or `@@global.` are global variables, and session variables name can optionally be preceded by `SESSION`, `@@session.`, `LOCAL`, or `@@local.`. System variables can be referenced in SELECT statement using the `@@[global.|session.|local.]var_name` syntax. If `global.`, `session.`, or `local.` is not present, MySQL returns the SESSION variable if it exists or the GLOBAL value otherwise. The following example shows some of these syntax options.

```
CREATE PROCEDURE p4()
BEGIN
```

```
    /* setting the value of a (dynamic) global variable */
    SET GLOBAL sort_buffer_size := 10000;

    /* retrieving the value of a global variable */
    SELECT @@global.sort_buffer_size;

    /* setting the value of a (dynamic) session variable */
    SET max_join_size := DEFAULT;

    /* retrieving the value of a session variable, shown using
       different syntax */
    SELECT @@session.max_join_size;
    SELECT @@local.max_join_size;
    SELECT @@max_join_size;
END;
```

Oracle PL/SQL also allows variables to be declared and used in stored procedures. As in MySQL, variables in PL/SQL must be declared in the declarative part of a PL/SQL block before they are referenced in any other statements in the block.

Local variables in PL/SQL have same scope as local variables in MySQL stored procedures. They are valid within the PL/SQL block where they are declared, including nested PL/SQL blocks. A variable of the same name in a nested PL/SQL block takes precedence over the variable in the enclosing PL/SQL block.

As with local variables in MySQL, variables in PL/SQL can have any SQL data type, such as NUMBER or VARCHAR2. In addition, variables in PL/SQL can also have PL/SQL data types, such as BOOLEAN or PLS_INTEGER, or be declared to hold table columns or table rows using the special qualifiers `%TYPE` and `%ROWTYPE`.

The following example shows some variable declarations in a PL/SQL block.

```
DECLARE
  /* variables of SQL data-type */
  wages NUMBER;
  hours_worked NUMBER := 40;
  hourly_salary NUMBER := 22.50;
  bonus NUMBER := 150;
  country VARCHAR2(128);
  counter NUMBER := 0;

  /* variables of PL/SQL data-types */
  done BOOLEAN;
  valid_id BOOLEAN;

  /* variables declared to hold table rows */
  emp_rec1 employees%ROWTYPE;
  emp_rec2 employees%ROWTYPE;
BEGIN
  wages := (hours_worked * hourly_salary) + bonus;
  country := 'France';
  country := UPPER('Canada');
  done := (counter > 100);
  valid_id := TRUE;
  emp_rec1.first_name := 'Antonio';
  emp_rec1.last_name := 'Ortiz';
  emp_rec1 := emp_rec2;
END;
```

Oracle PL/SQL also allows constants to be declared in stored procedures. Like variables, constants must be declared in the declarative part of a PL/SQL block before

they can be referenced in other statements in the PL/SQL block, including nested PL/SQL blocks. A constant is declared with the CONSTANT keyword. A constant must be initialized in its declaration, and no further assignments to the constant are allowed. The following example declares a constant in PL/SQL.

```
credit_limit CONSTANT NUMBER := 5000.00;
```

User variables in MySQL stored procedures can be emulated in Oracle by defining the variables in a package. The package specification emulates the per-session MySQL user variables.Variables defined in a package are available to the users of the package. For an example of a MySQL stored procedure and the converted equivalent in Oracle, consider the following MySQL stored procedure:

```
CREATE PROCEDURE p2()
BEGIN
  SET @a = 5;
  SET @b = 5;
  SELECT @a, @b;
END;
```

For this example, the Oracle equivalent statements are:

```
CREATE OR REPLACE PACKAGE root.globalPkg AS
   a NUMBER;
   b NUMBER;
END globalPkg;

CREATE OR REPLACE PROCEDURE root.p2 AS
BEGIN
   globalPkg.a := 5;
   globalPkg.b := 5;

   DBMS_OUTPUT.PUT_LINE(globalPkg.a || ',' || globalPkg.b);
END p2;

CREATE OR REPLACE PROCEDURE root.p3 AS
BEGIN
   globalPkg.a := globalPkg.a + 10;
   globalPkg.b := globalPkg.b - 5;

   DBMS_OUTPUT.PUT_LINE(globalPkg.a || ',' || globalPkg.b);
END p3;
```

3.2.3 Error Handling in Stored Procedures

Both Oracle PL/SQL and MySQL implement an error handling mechanism for their stored procedures. Each SQL statement in the stored procedure is checked for errors before the next statement is processed. If an error occurs, control immediately is passed to an error handler. For example, if a SELECT statement does not find any rows in the database, an error is raised, and the code to deal with this error is executed.

In MySQL stored procedures, handlers can be defined to deal with errors or warnings that occurs from executing a SQL statement within a stored procedure. MySQL allows two types of handlers: CONTINUE handlers and EXIT handlers. The two types of handlers differ from their next point of execution in the stored procedure after the handler is run. For a CONTINUE handler, execution continue at the next statement after the statement that raised the error. For an EXIT handler, execution of the current compound statement, enclosed by a pair of BEGIN and END statements, is terminated and execution continues at the next statement (if any) after the compound statement.

Handlers are defined to deal with one or more conditions. A condition may be a SQLSTATE value, a MySQL error code, or a predefined condition. There are three predefined conditions: SQLWARNING (warning or note), NOT FOUND (no more rows) and SQLEXCEPTION (error). A condition may be defined separately with a name and subsequently referenced in the handler statement. All the handler definitions are made at the start of a compound statement block.

In Oracle PL/SQL stored procedures, an error condition is called an exception. Exceptions may be internally defined (by the runtime system) or user-defined. Some internal exceptions have predefined name, such as ZERO_DIVIDE or NO_DATA_FOUND. Internal exceptions are implicitly (automatically) raised by the runtime system. User-defined exceptions must be given names and must be raised explicitly by RAISE statements in the stored procedures. Exception handlers handle exceptions that are raised.

Exception handlers can be declared for a PL/SQL block. Such exception handlers are enclosed between BEGIN and END statements, and they handle exceptions that might be raised by statements in the PL/SQL block, including sub-blocks. A PL/SQL block is similar to a MySQL compound statement block. Exceptions can be declared only in the declarative part of a PL/SQL block, and they are local to that block and global to all of its sub-blocks. Thus, the enclosing block cannot handle exceptions raised in a sub-block if they are exceptions local to the sub-block. Exceptions raised in a sub-block are not propagated to the enclosing block if exception handlers defined for sub-block handle them and if are not raised again in the exception handlers. After an exception handler runs, the current block stops executing and execution resumes at the next statement in the enclosing block.

For an example of using the error handling mechanism in MySQL and Oracle stored procedures, consider the following MySQL stored procedure:

```
CREATE PROCEDURE adjust_emp_salary ()
BEGIN
  DECLARE job_id INT;
  DECLARE employee_id INT DEFAULT 115;
  DECLARE sal_raise DECIMAL(3,2);
  DECLARE EXIT HANDLER FOR 1339;

  SELECT job_id INTO jobid FROM employees WHERE employee_id = empid;
  CASE
    WHEN jobid = 'PU_CLERK' THEN
      SET sal_raise := .09;
    WHEN jobid = 'SH_CLERK' THEN
      SET sal_raise := .08;
    WHEN jobid = 'ST_CLERK' THEN
      SET sal_raise := .07;
  END CASE;
END;
```

The following is the Oracle PL/SQL equivalent.

```
CREATE OR REPLACE PROCEDURE adjust_emp_salary ()
AS
  jobid employees.job_id%TYPE;
  empid employees.employee_id%TYPE := 115;
  sal_raise NUMBER(3,2);
BEGIN
  SELECT job_id INTO jobid from employees
    WHERE employee_id = empid;

  CASE
```

```
    WHEN jobid = 'PU_CLERK' THEN
      sal_raise := .09;
    WHEN jobid = 'SH_CLERK' THEN
      sal_raise := .08;
    WHEN jobid = 'ST_CLERK' THEN
      sal_raise := .07;
  END CASE;
  EXCEPTION
    WHEN CASE_NOT_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Employee salary not adjusted.');
END;
```

4 Troubleshooting

This chapter provides information about optimizing the command line options and avoiding issues connecting SQL Developer to the MySQL database server. It includes information about:

- Defining the User Account
- Dumping MySQL Data
- Optimizing Command Line Options

4.1 Defining the User Account

When you migrate to Oracle, SQL Developer attempts to connect to the MySQL database as MySQL user associated with the current database connection. (SQL Developer does *not* use the MySQL `root` user.) SQL Developer can capture only those MySQL database objects that the specified MySQL user has privileges to access.

Therefore, when you create the database connection that you intend to use for MySQL migrations, be sure to specify a MySQL user that has sufficient privileges to access all objects and data that you plan to migrate to Oracle.

4.2 Dumping MySQL Data

If you are having difficulty migrating from MySQL to Oracle, you can check the SQL Developer discussion forum on the Oracle Technology Network.

If you have a support contract, the SQL Developer support team may ask you to provide a dump of the MySQL database. This helps the team track the problem and provide a swift solution. By using the `mysqldump` method to create a copy of the MySQL database you generate text files that are portable to other systems, even those with different hardware architecture. The SQL Developer support team can regenerate the output into another database.

The following table provides an explanation of the code used to dump the MySQL data and to regenerate the database from the `mysqldump` output text file:

Command	Description
mysqldump	Allows you to extract the schema and data in a MySQL database to a file.
mysql	Loads MySQL so you can carry out the command.
`-u` *user name*	MySQL user name.
`-p` *password*	Password for the specified user.

Command	Description
`--opt`	Optimizes table dumping speed and writes a dump file for reloading speed. For a list of definitions of the options enabled by `--opt`, see "Optimizing Command Line Options".
`database_name`	Name of the database containing the information you want to dump to an output text file.
`>`	Symbol used for re-directing the input in UNIX and NT.
`file_name.sql`	File name containing the MySQL database information.

To dump the MySQL data, use the following command:

```
% mysqldump -u user name -ppassword --opt database_name > file_name.sql
```

To regenerate the database from the `mysqldump` output text file into a database, use the following command:

```
% mysql -u user name -ppassword database_name < file_name.sql
```

4.3 Optimizing Command Line Options

You automatically switch on options within the `mysqldump` command line by using `--opt`. The following table lists the commands encompassed by the `-opt` command:

Command	Description
`--add-drop-table`	Adds a `DROP TABLE IF EXISTS` statement before each `CREATE TABLE` statement.
`--all`	Includes all of the MySQL specific create options.
`--extended-insert`	Writes multiple row `INSERT` statements.
`--quick`	Dumps tables directly to the standard output without buffering the query. If you suspend `mysqldump` while using this option, you may interfere with other clients because it could cause the server to stop responding.
`--lock-tables`	Locks all tables as read only.

Index