存储对象

This chapter discusses stored database objects that are defined in terms of SQL code that is stored on the server for later execution.

Stored objects include these object types:

* Stored procedure: An object created with [**CREATE PROCEDURE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-procedure) and invoked using the [**CALL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#call) statement. A procedure does not have a return value but can modify its parameters for later inspection by the caller. It can also generate result sets to be returned to the client program.
* Stored function: An object created with [**CREATE FUNCTION**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-function) and used much like a built-in function. You invoke it in an expression and it returns a value during expression evaluation.
* Trigger: An object created with [**CREATE TRIGGER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-trigger) that is associated with a table. A trigger is activated when a particular event occurs for the table, such as an insert or update.
* Event: An object created with [**CREATE EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-event) and invoked by the server according to schedule.
* View: An object created with [**CREATE VIEW**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-view) that when referenced produces a result set. A view acts as a virtual table.

Terminology used in this document reflects the stored object hierarchy:

* Stored routines include stored procedures and functions.
* Stored programs include stored routines, triggers, and events.
* Stored objects include stored programs and views.

This chapter describes how to use stored objects. The following sections provide additional information about SQL syntax for statements related to these objects, and about object processing:

* For each object type, there are **CREATE**, **ALTER**, and **DROP** statements that control which objects exist and how they are defined. See [Section 13.1, “Data Definition Statements”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#sql-data-definition-statements).
* The [**CALL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#call) statement is used to invoke stored procedures. See [Section 13.2.1, “CALL Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#call).
* Stored program definitions include a body that may use compound statements, loops, conditionals, and declared variables. See [Section 13.6, “Compound Statement Syntax”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#sql-compound-statements).
* Metadata changes to objects referred to by stored programs are detected and cause automatic reparsing of the affected statements when the program is next executed. For more information, see [Section 8.10.3, “Caching of Prepared Statements and Stored Programs”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\optimization.html#statement-caching).

## 25.1 Defining Stored Programs

Each stored program contains a body that consists of an SQL statement. This statement may be a compound statement made up of several statements separated by semicolon (**;**) characters. For example, the following stored procedure has a body made up of a [**BEGIN ... END**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#begin-end) block that contains a [**SET**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#set-variable) statement and a [**REPEAT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#repeat) loop that itself contains another [**SET**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#set-variable) statement:

CREATE PROCEDURE dorepeat(p1 INT)

BEGIN

SET @x = 0;

REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;

END;

If you use the **[mysql](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\programs.html" \l "mysql" \o "4.5.1 mysql — The MySQL Command-Line Client)** client program to define a stored program containing semicolon characters, a problem arises. By default, **[mysql](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\programs.html" \l "mysql" \o "4.5.1 mysql — The MySQL Command-Line Client)** itself recognizes the semicolon as a statement delimiter, so you must redefine the delimiter temporarily to cause **[mysql](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\programs.html" \l "mysql" \o "4.5.1 mysql — The MySQL Command-Line Client)** to pass the entire stored program definition to the server.

To redefine the **[mysql](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\programs.html" \l "mysql" \o "4.5.1 mysql — The MySQL Command-Line Client)** delimiter, use the **delimiter** command. The following example shows how to do this for the **dorepeat()** procedure just shown. The delimiter is changed to **//** to enable the entire definition to be passed to the server as a single statement, and then restored to **;** before invoking the procedure. This enables the **;** delimiter used in the procedure body to be passed through to the server rather than being interpreted by **[mysql](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\programs.html" \l "mysql" \o "4.5.1 mysql — The MySQL Command-Line Client)** itself.

mysql> **delimiter //**

mysql> **CREATE PROCEDURE dorepeat(p1 INT)**

-> **BEGIN**

-> **SET @x = 0;**

-> **REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;**

-> **END**

-> **//**

Query OK, 0 rows affected (0.00 sec)

mysql> **delimiter ;**

mysql> **CALL dorepeat(1000);**

Query OK, 0 rows affected (0.00 sec)

mysql> **SELECT @x;**

+------+

| @x |

+------+

| 1001 |

+------+

1 row in set (0.00 sec)

You can redefine the delimiter to a string other than **//**, and the delimiter can consist of a single character or multiple characters. You should avoid the use of the backslash (**\**) character because that is the escape character for MySQL.

The following is an example of a function that takes a parameter, performs an operation using an SQL function, and returns the result. In this case, it is unnecessary to use **delimiter** because the function definition contains no internal **;** statement delimiters:

mysql> **CREATE FUNCTION hello (s CHAR(20))**

mysql> **RETURNS CHAR(50) DETERMINISTIC**

-> **RETURN CONCAT('Hello, ',s,'!');**

Query OK, 0 rows affected (0.00 sec)

mysql> **SELECT hello('world');**

+----------------+

| hello('world') |

+----------------+

| Hello, world! |

+----------------+

1 row in set (0.00 sec)

## 25.2 Using Stored Routines

[25.2.1 Stored Routine Syntax](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-routines-syntax)

[25.2.2 Stored Routines and MySQL Privileges](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-routines-privileges)

[25.2.3 Stored Routine Metadata](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-routines-metadata)

[25.2.4 Stored Procedures, Functions, Triggers, and LAST\_INSERT\_ID()](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-routines-last-insert-id)

MySQL supports stored routines (procedures and functions). A stored routine is a set of SQL statements that can be stored in the server. Once this has been done, clients don't need to keep reissuing the individual statements but can refer to the stored routine instead.

Stored routines can be particularly useful in certain situations:

* When multiple client applications are written in different languages or work on different platforms, but need to perform the same database operations.
* When security is paramount. Banks, for example, use stored procedures and functions for all common operations. This provides a consistent and secure environment, and routines can ensure that each operation is properly logged. In such a setup, applications and users would have no access to the database tables directly, but can only execute specific stored routines.

Stored routines can provide improved performance because less information needs to be sent between the server and the client. The tradeoff is that this does increase the load on the database server because more of the work is done on the server side and less is done on the client (application) side. Consider this if many client machines (such as Web servers) are serviced by only one or a few database servers.

Stored routines also enable you to have libraries of functions in the database server. This is a feature shared by modern application languages that enable such design internally (for example, by using classes). Using these client application language features is beneficial for the programmer even outside the scope of database use.

MySQL follows the SQL:2003 syntax for stored routines, which is also used by IBM's DB2. All syntax described here is supported and any limitations and extensions are documented where appropriate.

### Additional Resources

* You may find the [Stored Procedures User Forum](https://forums.mysql.com/list.php?98) of use when working with stored procedures and functions.
* For answers to some commonly asked questions regarding stored routines in MySQL, see [Section A.4, “MySQL 8.0 FAQ: Stored Procedures and Functions”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\faqs.html#faqs-stored-procs).
* There are some restrictions on the use of stored routines. See [Section 25.8, “Restrictions on Stored Programs”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-program-restrictions).
* Binary logging for stored routines takes place as described in [Section 25.7, “Stored Program Binary Logging”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-programs-logging).

### 25.2.1 Stored Routine Syntax

A stored routine is either a procedure or a function. Stored routines are created with the [**CREATE PROCEDURE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-procedure) and [**CREATE FUNCTION**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-function) statements (see [Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Statements”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-procedure)). A procedure is invoked using a [**CALL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#call) statement (see [Section 13.2.1, “CALL Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#call)), and can only pass back values using output variables. A function can be called from inside a statement just like any other function (that is, by invoking the function's name), and can return a scalar value. The body of a stored routine can use compound statements (see [Section 13.6, “Compound Statement Syntax”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#sql-compound-statements)).

Stored routines can be dropped with the [**DROP PROCEDURE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#drop-procedure) and [**DROP FUNCTION**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#drop-function) statements (see [Section 13.1.29, “DROP PROCEDURE and DROP FUNCTION Statements”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#drop-procedure)), and altered with the [**ALTER PROCEDURE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-procedure) and [**ALTER FUNCTION**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-function) statements (see [Section 13.1.7, “ALTER PROCEDURE Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-procedure)).

A stored procedure or function is associated with a particular database. This has several implications:

* When the routine is invoked, an implicit **USE *db\_name*** is performed (and undone when the routine terminates). [**USE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#use) statements within stored routines are not permitted.
* You can qualify routine names with the database name. This can be used to refer to a routine that is not in the current database. For example, to invoke a stored procedure **p** or function **f** that is associated with the **test** database, you can say **CALL test.p()** or **test.f()**.
* When a database is dropped, all stored routines associated with it are dropped as well.

Stored functions cannot be recursive.

Recursion in stored procedures is permitted but disabled by default. To enable recursion, set the **[max\_sp\_recursion\_depth](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_max_sp_recursion_depth)** server system variable to a value greater than zero. Stored procedure recursion increases the demand on thread stack space. If you increase the value of **[max\_sp\_recursion\_depth](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_max_sp_recursion_depth)**, it may be necessary to increase thread stack size by increasing the value of **[thread\_stack](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_thread_stack)** at server startup. See [Section 5.1.8, “Server System Variables”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#server-system-variables), for more information.

MySQL supports a very useful extension that enables the use of regular [**SELECT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#select) statements (that is, without using cursors or local variables) inside a stored procedure. The result set of such a query is simply sent directly to the client. Multiple [**SELECT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#select) statements generate multiple result sets, so the client must use a MySQL client library that supports multiple result sets. This means the client must use a client library from a version of MySQL at least as recent as 4.1. The client should also specify the **CLIENT\_MULTI\_RESULTS** option when it connects. For C programs, this can be done with the **[mysql\_real\_connect()](https://dev.mysql.com/doc/c-api/8.0/en/mysql-real-connect.html" \t "_top)** C API function. See [mysql\_real\_connect()](https://dev.mysql.com/doc/c-api/8.0/en/mysql-real-connect.html" \t "_top), and [Multiple Statement Execution Support](https://dev.mysql.com/doc/c-api/8.0/en/c-api-multiple-queries.html).

In MySQL 8.0.22 and later, a user variable referenced by a statement in a stored procedure has its type determined the first time the procedure is invoked, and retains this type each time the procedure is invoked thereafter.

### 25.2.2 Stored Routines and MySQL Privileges

The MySQL grant system takes stored routines into account as follows:

* The [**CREATE ROUTINE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_create-routine) privilege is needed to create stored routines.
* The [**ALTER ROUTINE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_alter-routine) privilege is needed to alter or drop stored routines. This privilege is granted automatically to the creator of a routine if necessary, and dropped from the creator when the routine is dropped.
* The [**EXECUTE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_execute) privilege is required to execute stored routines. However, this privilege is granted automatically to the creator of a routine if necessary (and dropped from the creator when the routine is dropped). Also, the default **SQL SECURITY** characteristic for a routine is **DEFINER**, which enables users who have access to the database with which the routine is associated to execute the routine.
* If the **[automatic\_sp\_privileges](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_automatic_sp_privileges)** system variable is 0, the [**EXECUTE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_execute) and [**ALTER ROUTINE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_alter-routine) privileges are not automatically granted to and dropped from the routine creator.
* The creator of a routine is the account used to execute the **CREATE** statement for it. This might not be the same as the account named as the **DEFINER** in the routine definition.
* The account named as a routine **DEFINER** can see all routine properties, including its definition. The account thus has full access to the routine output as produced by:
  + The contents of the [**INFORMATION\_SCHEMA.ROUTINES**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\information-schema.html#information-schema-routines-table) table.
  + The [**SHOW CREATE FUNCTION**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-create-function) and [**SHOW CREATE PROCEDURE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-create-procedure) statements.
  + The [**SHOW FUNCTION CODE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-function-code) and [**SHOW PROCEDURE CODE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-procedure-code) statements.
  + The [**SHOW FUNCTION STATUS**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-function-status) and [**SHOW PROCEDURE STATUS**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-procedure-status) statements.
* For an account other than the account named as the routine **DEFINER**, access to routine properties depends on the privileges granted to the account:
  + With the [**SHOW\_ROUTINE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_show-routine) privilege or the global [**SELECT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_select) privilege, the account can see all routine properties, including its definition.
  + With the [**CREATE ROUTINE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_create-routine), [**ALTER ROUTINE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_alter-routine) or [**EXECUTE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_execute) privilege granted at a scope that includes the routine, the account can see all routine properties except its definition.

### 25.2.3 Stored Routine Metadata

To obtain metadata about stored routines:

* Query the [**ROUTINES**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\information-schema.html#information-schema-routines-table) table of the **INFORMATION\_SCHEMA** database. See [Section 26.3.30, “The INFORMATION\_SCHEMA ROUTINES Table”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\information-schema.html#information-schema-routines-table).
* Use the [**SHOW CREATE PROCEDURE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-create-procedure) and [**SHOW CREATE FUNCTION**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-create-function) statements to see routine definitions. See [Section 13.7.7.9, “SHOW CREATE PROCEDURE Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-create-procedure).
* Use the [**SHOW PROCEDURE STATUS**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-procedure-status) and [**SHOW FUNCTION STATUS**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-function-status) statements to see routine characteristics. See [Section 13.7.7.28, “SHOW PROCEDURE STATUS Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-procedure-status).
* Use the [**SHOW PROCEDURE CODE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-procedure-code) and [**SHOW FUNCTION CODE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-function-code) statements to see a representation of the internal implementation of the routine. See [Section 13.7.7.27, “SHOW PROCEDURE CODE Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-procedure-code).

### 25.2.4 Stored Procedures, Functions, Triggers, and LAST\_INSERT\_ID()

Within the body of a stored routine (procedure or function) or a trigger, the value of [**LAST\_INSERT\_ID()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_last-insert-id) changes the same way as for statements executed outside the body of these kinds of objects (see [Section 12.16, “Information Functions”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#information-functions)). The effect of a stored routine or trigger upon the value of [**LAST\_INSERT\_ID()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_last-insert-id) that is seen by following statements depends on the kind of routine:

* If a stored procedure executes statements that change the value of [**LAST\_INSERT\_ID()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_last-insert-id), the changed value is seen by statements that follow the procedure call.
* For stored functions and triggers that change the value, the value is restored when the function or trigger ends, so following statements do not see a changed value.

## 25.3 Using Triggers

[25.3.1 Trigger Syntax and Examples](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#trigger-syntax)

[25.3.2 Trigger Metadata](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#trigger-metadata)

A trigger is a named database object that is associated with a table, and that activates when a particular event occurs for the table. Some uses for triggers are to perform checks of values to be inserted into a table or to perform calculations on values involved in an update.

A trigger is defined to activate when a statement inserts, updates, or deletes rows in the associated table. These row operations are trigger events. For example, rows can be inserted by [**INSERT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#insert) or [**LOAD DATA**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#load-data) statements, and an insert trigger activates for each inserted row. A trigger can be set to activate either before or after the trigger event. For example, you can have a trigger activate before each row that is inserted into a table or after each row that is updated.

**Important**

MySQL triggers activate only for changes made to tables by SQL statements. This includes changes to base tables that underlie updatable views. Triggers do not activate for changes to tables made by APIs that do not transmit SQL statements to the MySQL Server. This means that triggers are not activated by updates made using the [**NDB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\mysql-cluster.html) API.

Triggers are not activated by changes in **INFORMATION\_SCHEMA** or **performance\_schema** tables. Those tables are actually views and triggers are not permitted on views.

The following sections describe the syntax for creating and dropping triggers, show some examples of how to use them, and indicate how to obtain trigger metadata.

### Additional Resources

* You may find the [Triggers User Forum](https://forums.mysql.com/list.php?100) of use when working with triggers.
* For answers to commonly asked questions regarding triggers in MySQL, see [Section A.5, “MySQL 8.0 FAQ: Triggers”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\faqs.html#faqs-triggers).
* There are some restrictions on the use of triggers; see [Section 25.8, “Restrictions on Stored Programs”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-program-restrictions).
* Binary logging for triggers takes place as described in [Section 25.7, “Stored Program Binary Logging”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-programs-logging).

### 25.3.1 Trigger Syntax and Examples

To create a trigger or drop a trigger, use the [**CREATE TRIGGER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-trigger) or [**DROP TRIGGER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#drop-trigger) statement, described in [Section 13.1.22, “CREATE TRIGGER Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-trigger), and [Section 13.1.34, “DROP TRIGGER Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#drop-trigger).

Here is a simple example that associates a trigger with a table, to activate for [**INSERT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#insert) operations. The trigger acts as an accumulator, summing the values inserted into one of the columns of the table.

mysql> **CREATE TABLE account (acct\_num INT, amount DECIMAL(10,2));**

Query OK, 0 rows affected (0.03 sec)

mysql> **CREATE TRIGGER ins\_sum BEFORE INSERT ON account**

**FOR EACH ROW SET @sum = @sum + NEW.amount;**

Query OK, 0 rows affected (0.01 sec)

The [**CREATE TRIGGER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-trigger) statement creates a trigger named **ins\_sum** that is associated with the **account** table. It also includes clauses that specify the trigger action time, the triggering event, and what to do when the trigger activates:

* The keyword **BEFORE** indicates the trigger action time. In this case, the trigger activates before each row inserted into the table. The other permitted keyword here is **AFTER**.
* The keyword **INSERT** indicates the trigger event; that is, the type of operation that activates the trigger. In the example, [**INSERT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#insert) operations cause trigger activation. You can also create triggers for [**DELETE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#delete) and [**UPDATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#update) operations.
* The statement following **FOR EACH ROW** defines the trigger body; that is, the statement to execute each time the trigger activates, which occurs once for each row affected by the triggering event. In the example, the trigger body is a simple [**SET**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#set-variable) that accumulates into a user variable the values inserted into the **amount** column. The statement refers to the column as **NEW.amount** which means “the value of the **amount** column to be inserted into the new row.”

To use the trigger, set the accumulator variable to zero, execute an [**INSERT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#insert) statement, and then see what value the variable has afterward:

mysql> **SET @sum = 0;**

mysql> **INSERT INTO account VALUES(137,14.98),(141,1937.50),(97,-100.00);**

mysql> **SELECT @sum AS 'Total amount inserted';**

+-----------------------+

| Total amount inserted |

+-----------------------+

| 1852.48 |

+-----------------------+

In this case, the value of **@sum** after the [**INSERT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#insert) statement has executed is **14.98 + 1937.50 - 100**, or **1852.48**.

To destroy the trigger, use a [**DROP TRIGGER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#drop-trigger) statement. You must specify the schema name if the trigger is not in the default schema:

mysql> **DROP TRIGGER test.ins\_sum;**

If you drop a table, any triggers for the table are also dropped.

Trigger names exist in the schema namespace, meaning that all triggers must have unique names within a schema. Triggers in different schemas can have the same name.

It is possible to define multiple triggers for a given table that have the same trigger event and action time. For example, you can have two **BEFORE UPDATE** triggers for a table. By default, triggers that have the same trigger event and action time activate in the order they were created. To affect trigger order, specify a clause after **FOR EACH ROW** that indicates **FOLLOWS** or **PRECEDES** and the name of an existing trigger that also has the same trigger event and action time. With **FOLLOWS**, the new trigger activates after the existing trigger. With **PRECEDES**, the new trigger activates before the existing trigger.

For example, the following trigger definition defines another **BEFORE INSERT** trigger for the **account** table:

mysql> **CREATE TRIGGER ins\_transaction BEFORE INSERT ON account**

**FOR EACH ROW PRECEDES ins\_sum**

**SET**

**@deposits = @deposits + IF(NEW.amount>0,NEW.amount,0),**

**@withdrawals = @withdrawals + IF(NEW.amount<0,-NEW.amount,0);**

Query OK, 0 rows affected (0.01 sec)

This trigger, **ins\_transaction**, is similar to **ins\_sum** but accumulates deposits and withdrawals separately. It has a **PRECEDES** clause that causes it to activate before **ins\_sum**; without that clause, it would activate after **ins\_sum** because it is created after **ins\_sum**.

Within the trigger body, the **OLD** and **NEW** keywords enable you to access columns in the rows affected by a trigger. **OLD** and **NEW** are MySQL extensions to triggers; they are not case-sensitive.

In an **INSERT** trigger, only **NEW.*col\_name*** can be used; there is no old row. In a **DELETE** trigger, only **OLD.*col\_name*** can be used; there is no new row. In an **UPDATE** trigger, you can use **OLD.*col\_name*** to refer to the columns of a row before it is updated and **NEW.*col\_name*** to refer to the columns of the row after it is updated.

A column named with **OLD** is read only. You can refer to it (if you have the [**SELECT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#select) privilege), but not modify it. You can refer to a column named with **NEW** if you have the [**SELECT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_select) privilege for it. In a **BEFORE** trigger, you can also change its value with **SET NEW.*col\_name* = *value*** if you have the [**UPDATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_update) privilege for it. This means you can use a trigger to modify the values to be inserted into a new row or used to update a row. (Such a **SET** statement has no effect in an **AFTER** trigger because the row change has already occurred.)

In a **BEFORE** trigger, the **NEW** value for an **AUTO\_INCREMENT** column is 0, not the sequence number that is generated automatically when the new row actually is inserted.

By using the [**BEGIN ... END**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#begin-end) construct, you can define a trigger that executes multiple statements. Within the **BEGIN** block, you also can use other syntax that is permitted within stored routines such as conditionals and loops. However, just as for stored routines, if you use the **[mysql](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\programs.html" \l "mysql" \o "4.5.1 mysql — The MySQL Command-Line Client)** program to define a trigger that executes multiple statements, it is necessary to redefine the **[mysql](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\programs.html" \l "mysql" \o "4.5.1 mysql — The MySQL Command-Line Client)** statement delimiter so that you can use the **;** statement delimiter within the trigger definition. The following example illustrates these points. It defines an **UPDATE** trigger that checks the new value to be used for updating each row, and modifies the value to be within the range from 0 to 100. This must be a **BEFORE** trigger because the value must be checked before it is used to update the row:

mysql> **delimiter //**

mysql> **CREATE TRIGGER upd\_check BEFORE UPDATE ON account**

**FOR EACH ROW**

**BEGIN**

**IF NEW.amount < 0 THEN**

**SET NEW.amount = 0;**

**ELSEIF NEW.amount > 100 THEN**

**SET NEW.amount = 100;**

**END IF;**

**END;//**

mysql> **delimiter ;**

It can be easier to define a stored procedure separately and then invoke it from the trigger using a simple [**CALL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#call) statement. This is also advantageous if you want to execute the same code from within several triggers.

There are limitations on what can appear in statements that a trigger executes when activated:

* The trigger cannot use the [**CALL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#call) statement to invoke stored procedures that return data to the client or that use dynamic SQL. (Stored procedures are permitted to return data to the trigger through **OUT** or **INOUT** parameters.)
* The trigger cannot use statements that explicitly or implicitly begin or end a transaction, such as [**START TRANSACTION**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#commit), [**COMMIT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#commit), or [**ROLLBACK**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#commit). ([**ROLLBACK to SAVEPOINT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#commit) is permitted because it does not end a transaction.).

See also [Section 25.8, “Restrictions on Stored Programs”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-program-restrictions).

MySQL handles errors during trigger execution as follows:

* If a **BEFORE** trigger fails, the operation on the corresponding row is not performed.
* A **BEFORE** trigger is activated by the attempt to insert or modify the row, regardless of whether the attempt subsequently succeeds.
* An **AFTER** trigger is executed only if any **BEFORE** triggers and the row operation execute successfully.
* An error during either a **BEFORE** or **AFTER** trigger results in failure of the entire statement that caused trigger invocation.
* For transactional tables, failure of a statement should cause rollback of all changes performed by the statement. Failure of a trigger causes the statement to fail, so trigger failure also causes rollback. For nontransactional tables, such rollback cannot be done, so although the statement fails, any changes performed prior to the point of the error remain in effect.

Triggers can contain direct references to tables by name, such as the trigger named **testref** shown in this example:

CREATE TABLE test1(a1 INT);

CREATE TABLE test2(a2 INT);

CREATE TABLE test3(a3 INT NOT NULL AUTO\_INCREMENT PRIMARY KEY);

CREATE TABLE test4(

a4 INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,

b4 INT DEFAULT 0

);

delimiter |

CREATE TRIGGER testref BEFORE INSERT ON test1

FOR EACH ROW

BEGIN

INSERT INTO test2 SET a2 = NEW.a1;

DELETE FROM test3 WHERE a3 = NEW.a1;

UPDATE test4 SET b4 = b4 + 1 WHERE a4 = NEW.a1;

END;

|

delimiter ;

INSERT INTO test3 (a3) VALUES

(NULL), (NULL), (NULL), (NULL), (NULL),

(NULL), (NULL), (NULL), (NULL), (NULL);

INSERT INTO test4 (a4) VALUES

(0), (0), (0), (0), (0), (0), (0), (0), (0), (0);

Suppose that you insert the following values into table **test1** as shown here:

mysql> **INSERT INTO test1 VALUES**

**(1), (3), (1), (7), (1), (8), (4), (4);**

Query OK, 8 rows affected (0.01 sec)

Records: 8 Duplicates: 0 Warnings: 0

As a result, the four tables contain the following data:

mysql> **SELECT \* FROM test1;**

+------+

| a1 |

+------+

| 1 |

| 3 |

| 1 |

| 7 |

| 1 |

| 8 |

| 4 |

| 4 |

+------+

8 rows in set (0.00 sec)

mysql> **SELECT \* FROM test2;**

+------+

| a2 |

+------+

| 1 |

| 3 |

| 1 |

| 7 |

| 1 |

| 8 |

| 4 |

| 4 |

+------+

8 rows in set (0.00 sec)

mysql> **SELECT \* FROM test3;**

+----+

| a3 |

+----+

| 2 |

| 5 |

| 6 |

| 9 |

| 10 |

+----+

5 rows in set (0.00 sec)

mysql> **SELECT \* FROM test4;**

+----+------+

| a4 | b4 |

+----+------+

| 1 | 3 |

| 2 | 0 |

| 3 | 1 |

| 4 | 2 |

| 5 | 0 |

| 6 | 0 |

| 7 | 1 |

| 8 | 1 |

| 9 | 0 |

| 10 | 0 |

+----+------+

10 rows in set (0.00 sec)

### 25.3.2 Trigger Metadata

To obtain metadata about triggers:

* Query the [**TRIGGERS**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\information-schema.html#information-schema-triggers-table) table of the **INFORMATION\_SCHEMA** database. See [Section 26.3.45, “The INFORMATION\_SCHEMA TRIGGERS Table”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\information-schema.html#information-schema-triggers-table).
* Use the [**SHOW CREATE TRIGGER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-create-trigger) statement. See [Section 13.7.7.11, “SHOW CREATE TRIGGER Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-create-trigger).
* Use the [**SHOW TRIGGERS**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-triggers) statement. See [Section 13.7.7.40, “SHOW TRIGGERS Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-triggers).

## 25.4 Using the Event Scheduler

[25.4.1 Event Scheduler Overview](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#events-overview)

[25.4.2 Event Scheduler Configuration](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#events-configuration)

[25.4.3 Event Syntax](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#events-syntax)

[25.4.4 Event Metadata](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#events-metadata)

[25.4.5 Event Scheduler Status](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#events-status-info)

[25.4.6 The Event Scheduler and MySQL Privileges](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#events-privileges)

The MySQL Event Scheduler manages the scheduling and execution of events, that is, tasks that run according to a schedule. The following discussion covers the Event Scheduler and is divided into the following sections:

* [Section 25.4.1, “Event Scheduler Overview”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#events-overview), provides an introduction to and conceptual overview of MySQL Events.
* [Section 25.4.3, “Event Syntax”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#events-syntax), discusses the SQL statements for creating, altering, and dropping MySQL Events.
* [Section 25.4.4, “Event Metadata”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#events-metadata), shows how to obtain information about events and how this information is stored by the MySQL Server.
* [Section 25.4.6, “The Event Scheduler and MySQL Privileges”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#events-privileges), discusses the privileges required to work with events and the ramifications that events have with regard to privileges when executing.

Stored routines require the **events** data dictionary table in the **mysql** system database. This table is created during the MySQL 8.0 installation procedure. If you are upgrading to MySQL 8.0 from an earlier version, be sure to perform the upgrade procedure to make sure that your system database is up to date. See [Section 2.11, “Upgrading MySQL”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\installing.html#upgrading).

### Additional Resources

* You may find the [MySQL Event Scheduler User Forum](https://forums.mysql.com/list.php?119) of use when working with scheduled events.
* There are some restrictions on the use of events; see [Section 25.8, “Restrictions on Stored Programs”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-program-restrictions).
* Binary logging for events takes place as described in [Section 25.7, “Stored Program Binary Logging”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-programs-logging).

### 25.4.1 Event Scheduler Overview

MySQL Events are tasks that run according to a schedule. Therefore, we sometimes refer to them as scheduled events. When you create an event, you are creating a named database object containing one or more SQL statements to be executed at one or more regular intervals, beginning and ending at a specific date and time. Conceptually, this is similar to the idea of the Unix **crontab** (also known as a “cron job”) or the Windows Task Scheduler.

Scheduled tasks of this type are also sometimes known as “temporal triggers”, implying that these are objects that are triggered by the passage of time. While this is essentially correct, we prefer to use the term events to avoid confusion with triggers of the type discussed in [Section 25.3, “Using Triggers”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#triggers). Events should more specifically not be confused with “temporary triggers”. Whereas a trigger is a database object whose statements are executed in response to a specific type of event that occurs on a given table, a (scheduled) event is an object whose statements are executed in response to the passage of a specified time interval.

While there is no provision in the SQL Standard for event scheduling, there are precedents in other database systems, and you may notice some similarities between these implementations and that found in the MySQL Server.

MySQL Events have the following major features and properties:

* In MySQL, an event is uniquely identified by its name and the schema to which it is assigned.
* An event performs a specific action according to a schedule. This action consists of an SQL statement, which can be a compound statement in a [**BEGIN ... END**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#begin-end) block if desired (see [Section 13.6, “Compound Statement Syntax”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#sql-compound-statements)). An event's timing can be either one-time or recurrent. A one-time event executes one time only. A recurrent event repeats its action at a regular interval, and the schedule for a recurring event can be assigned a specific start day and time, end day and time, both, or neither. (By default, a recurring event's schedule begins as soon as it is created, and continues indefinitely, until it is disabled or dropped.)

If a repeating event does not terminate within its scheduling interval, the result may be multiple instances of the event executing simultaneously. If this is undesirable, you should institute a mechanism to prevent simultaneous instances. For example, you could use the [**GET\_LOCK()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_get-lock) function, or row or table locking.

* Users can create, modify, and drop scheduled events using SQL statements intended for these purposes. Syntactically invalid event creation and modification statements fail with an appropriate error message. A user may include statements in an event's action which require privileges that the user does not actually have. The event creation or modification statement succeeds but the event's action fails. See [Section 25.4.6, “The Event Scheduler and MySQL Privileges”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#events-privileges) for details.
* Many of the properties of an event can be set or modified using SQL statements. These properties include the event's name, timing, persistence (that is, whether it is preserved following the expiration of its schedule), status (enabled or disabled), action to be performed, and the schema to which it is assigned. See [Section 13.1.3, “ALTER EVENT Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-event).

The default definer of an event is the user who created the event, unless the event has been altered, in which case the definer is the user who issued the last [**ALTER EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-event) statement affecting that event. An event can be modified by any user having the [**EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_event) privilege on the database for which the event is defined. See [Section 25.4.6, “The Event Scheduler and MySQL Privileges”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#events-privileges).

* An event's action statement may include most SQL statements permitted within stored routines. For restrictions, see [Section 25.8, “Restrictions on Stored Programs”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-program-restrictions).

### 25.4.2 Event Scheduler Configuration

Events are executed by a special event scheduler thread; when we refer to the Event Scheduler, we actually refer to this thread. When running, the event scheduler thread and its current state can be seen by users having the [**PROCESS**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_process) privilege in the output of [**SHOW PROCESSLIST**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-processlist), as shown in the discussion that follows.

The global **[event\_scheduler](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_event_scheduler)** system variable determines whether the Event Scheduler is enabled and running on the server. It has one of these 3 values, which affect event scheduling as described here. The default is **ON**.

* **ON**: The Event Scheduler is started; the event scheduler thread runs and executes all scheduled events.

When the Event Scheduler is **ON**, the event scheduler thread is listed in the output of [**SHOW PROCESSLIST**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-processlist) as a daemon process, and its state is represented as shown here:

mysql> **SHOW PROCESSLIST\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Id: 1

User: root

Host: localhost

db: NULL

Command: Query

Time: 0

State: NULL

Info: show processlist

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Id: 2

User: event\_scheduler

Host: localhost

db: NULL

Command: Daemon

Time: 3

State: Waiting for next activation

Info: NULL

2 rows in set (0.00 sec)

Event scheduling can be stopped by setting the value of **[event\_scheduler](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_event_scheduler)** to **OFF**.

* **OFF**: The Event Scheduler is stopped. The event scheduler thread does not run, is not shown in the output of [**SHOW PROCESSLIST**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-processlist), and no scheduled events are executed.

When the Event Scheduler is stopped (**[event\_scheduler](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_event_scheduler)** is **OFF**), it can be started by setting the value of **[event\_scheduler](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_event_scheduler)** to **ON**. (See next item.)

* **DISABLED**: This value renders the Event Scheduler nonoperational. When the Event Scheduler is **DISABLED**, the event scheduler thread does not run (and so does not appear in the output of [**SHOW PROCESSLIST**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-processlist)). In addition, the Event Scheduler state cannot be changed at runtime.

If the Event Scheduler status has not been set to **DISABLED**, **[event\_scheduler](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_event_scheduler)** can be toggled between **ON** and **OFF** (using [**SET**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#set-variable)). It is also possible to use **0** for **OFF**, and **1** for **ON** when setting this variable. Thus, any of the following 4 statements can be used in the **[mysql](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\programs.html" \l "mysql" \o "4.5.1 mysql — The MySQL Command-Line Client)** client to turn on the Event Scheduler:

SET GLOBAL event\_scheduler = ON;

SET @@GLOBAL.event\_scheduler = ON;

SET GLOBAL event\_scheduler = 1;

SET @@GLOBAL.event\_scheduler = 1;

Similarly, any of these 4 statements can be used to turn off the Event Scheduler:

SET GLOBAL event\_scheduler = OFF;

SET @@GLOBAL.event\_scheduler = OFF;

SET GLOBAL event\_scheduler = 0;

SET @@GLOBAL.event\_scheduler = 0;

Although **ON** and **OFF** have numeric equivalents, the value displayed for **[event\_scheduler](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_event_scheduler)** by [**SELECT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#select) or [**SHOW VARIABLES**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-variables) is always one of **OFF**, **ON**, or **DISABLED**. ***DISABLED*** has no numeric equivalent. For this reason, **ON** and **OFF** are usually preferred over **1** and **0** when setting this variable.

Note that attempting to set **[event\_scheduler](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_event_scheduler)** without specifying it as a global variable causes an error:

mysql< **SET @@event\_scheduler = OFF;**

ERROR 1229 (HY000): Variable 'event\_scheduler' is a GLOBAL

variable and should be set with SET GLOBAL

**Important**

It is possible to set the Event Scheduler to **DISABLED** only at server startup. If **[event\_scheduler](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_event_scheduler)** is **ON** or **OFF**, you cannot set it to **DISABLED** at runtime. Also, if the Event Scheduler is set to **DISABLED** at startup, you cannot change the value of **[event\_scheduler](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_event_scheduler)** at runtime.

To disable the event scheduler, use one of the following two methods:

* As a command-line option when starting the server:
* --event-scheduler=DISABLED
* In the server configuration file (my.cnf, or my.ini on Windows systems), include the line where it can be read by the server (for example, in a **[mysqld]** section):
* event\_scheduler=DISABLED

To enable the Event Scheduler, restart the server without the [--event-scheduler=DISABLED](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sysvar_event_scheduler) command-line option, or after removing or commenting out the line containing [event-scheduler=DISABLED](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#sysvar_event_scheduler) in the server configuration file, as appropriate. Alternatively, you can use **ON** (or **1**) or **OFF** (or **0**) in place of the **DISABLED** value when starting the server.

**Note**

You can issue event-manipulation statements when **[event\_scheduler](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_event_scheduler)** is set to **DISABLED**. No warnings or errors are generated in such cases (provided that the statements are themselves valid). However, scheduled events cannot execute until this variable is set to **ON** (or **1**). Once this has been done, the event scheduler thread executes all events whose scheduling conditions are satisfied.

Starting the MySQL server with the [--skip-grant-tables](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#option_mysqld_skip-grant-tables) option causes **[event\_scheduler](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_event_scheduler)** to be set to **DISABLED**, overriding any other value set either on the command line or in the my.cnf or my.ini file (Bug #26807).

For SQL statements used to create, alter, and drop events, see [Section 25.4.3, “Event Syntax”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#events-syntax).

MySQL provides an [**EVENTS**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\information-schema.html#information-schema-events-table) table in the **INFORMATION\_SCHEMA** database. This table can be queried to obtain information about scheduled events which have been defined on the server. See [Section 25.4.4, “Event Metadata”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#events-metadata), and [Section 26.3.14, “The INFORMATION\_SCHEMA EVENTS Table”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\information-schema.html#information-schema-events-table), for more information.

For information regarding event scheduling and the MySQL privilege system, see [Section 25.4.6, “The Event Scheduler and MySQL Privileges”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#events-privileges).

### 25.4.3 Event Syntax

MySQL provides several SQL statements for working with scheduled events:

* New events are defined using the [**CREATE EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-event) statement. See [Section 13.1.13, “CREATE EVENT Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-event).
* The definition of an existing event can be changed by means of the [**ALTER EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-event) statement. See [Section 13.1.3, “ALTER EVENT Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-event).
* When a scheduled event is no longer wanted or needed, it can be deleted from the server by its definer using the [**DROP EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#drop-event) statement. See [Section 13.1.25, “DROP EVENT Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#drop-event). Whether an event persists past the end of its schedule also depends on its **ON COMPLETION** clause, if it has one. See [Section 13.1.13, “CREATE EVENT Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-event).

An event can be dropped by any user having the [**EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_event) privilege for the database on which the event is defined. See [Section 25.4.6, “The Event Scheduler and MySQL Privileges”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#events-privileges).

### 25.4.4 Event Metadata

To obtain metadata about events:

* Query the [**EVENTS**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\information-schema.html#information-schema-events-table) table of the **INFORMATION\_SCHEMA** database. See [Section 26.3.14, “The INFORMATION\_SCHEMA EVENTS Table”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\information-schema.html#information-schema-events-table).
* Use the [**SHOW CREATE EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-create-event) statement. See [Section 13.7.7.7, “SHOW CREATE EVENT Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-create-event).
* Use the [**SHOW EVENTS**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-events) statement. See [Section 13.7.7.18, “SHOW EVENTS Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-events).

***Event Scheduler Time Representation***

Each session in MySQL has a session time zone (STZ). This is the session **[time\_zone](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_time_zone)** value that is initialized from the server's global **[time\_zone](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_time_zone)** value when the session begins but may be changed during the session.

The session time zone that is current when a [**CREATE EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-event) or [**ALTER EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-event) statement executes is used to interpret times specified in the event definition. This becomes the event time zone (ETZ); that is, the time zone that is used for event scheduling and is in effect within the event as it executes.

For representation of event information in the data dictionary, the **execute\_at**, **starts**, and **ends** times are converted to UTC and stored along with the event time zone. This enables event execution to proceed as defined regardless of any subsequent changes to the server time zone or daylight saving time effects. The **last\_executed** time is also stored in UTC.

Event times can be obtained by selecting from the [**INFORMATION\_SCHEMA.EVENTS**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\information-schema.html#information-schema-events-table) table or from [**SHOW EVENTS**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-events), but they are reported as ETZ or STZ values. The following table summarizes representation of event times.

| **Value** | [**INFORMATION\_SCHEMA.EVENTS**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\information-schema.html#information-schema-events-table) | [**SHOW EVENTS**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-events) |
| --- | --- | --- |
| **Execute at** | ETZ | ETZ |
| **Starts** | ETZ | ETZ |
| **Ends** | ETZ | ETZ |
| **Last executed** | ETZ | n/a |
| **Created** | STZ | n/a |
| **Last altered** | STZ | n/a |

### 25.4.5 Event Scheduler Status

The Event Scheduler writes information about event execution that terminates with an error or warning to the MySQL Server's error log. See [Section 25.4.6, “The Event Scheduler and MySQL Privileges”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#events-privileges) for an example.

To obtain information about the state of the Event Scheduler for debugging and troubleshooting purposes, run **[mysqladmin debug](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\programs.html" \l "mysqladmin" \o "4.5.2 mysqladmin — A MySQL Server Administration Program)** (see [Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\programs.html#mysqladmin)); after running this command, the server's error log contains output relating to the Event Scheduler, similar to what is shown here:

Events status:

LLA = Last Locked At LUA = Last Unlocked At

WOC = Waiting On Condition DL = Data Locked

Event scheduler status:

State : INITIALIZED

Thread id : 0

LLA : n/a:0

LUA : n/a:0

WOC : NO

Workers : 0

Executed : 0

Data locked: NO

Event queue status:

Element count : 0

Data locked : NO

Attempting lock : NO

LLA : init\_queue:95

LUA : init\_queue:103

WOC : NO

Next activation : never

In statements that occur as part of events executed by the Event Scheduler, diagnostics messages (not only errors, but also warnings) are written to the error log, and, on Windows, to the application event log. For frequently executed events, it is possible for this to result in many logged messages. For example, for **SELECT ... INTO *var\_list*** statements, if the query returns no rows, a warning with error code 1329 occurs (**No data**), and the variable values remain unchanged. If the query returns multiple rows, error 1172 occurs (**Result consisted of more than one row**). For either condition, you can avoid having the warnings be logged by declaring a condition handler; see [Section 13.6.7.2, “DECLARE ... HANDLER Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#declare-handler). For statements that may retrieve multiple rows, another strategy is to use **LIMIT 1** to limit the result set to a single row.

### 25.4.6 The Event Scheduler and MySQL Privileges

To enable or disable the execution of scheduled events, it is necessary to set the value of the global **[event\_scheduler](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_event_scheduler)** system variable. This requires privileges sufficient to set global system variables. See [Section 5.1.9.1, “System Variable Privileges”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#system-variable-privileges).

The [**EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_event) privilege governs the creation, modification, and deletion of events. This privilege can be bestowed using [**GRANT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#grant). For example, this [**GRANT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#grant) statement confers the [**EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_event) privilege for the schema named **myschema** on the user **jon@ghidora**:

GRANT EVENT ON myschema.\* TO jon@ghidora;

(We assume that this user account already exists, and that we wish for it to remain unchanged otherwise.)

To grant this same user the [**EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_event) privilege on all schemas, use the following statement:

GRANT EVENT ON \*.\* TO jon@ghidora;

The [**EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_event) privilege has global or schema-level scope. Therefore, trying to grant it on a single table results in an error as shown:

mysql> **GRANT EVENT ON myschema.mytable TO jon@ghidora;**

ERROR 1144 (42000): Illegal GRANT/REVOKE command; please

consult the manual to see which privileges can be used

It is important to understand that an event is executed with the privileges of its definer, and that it cannot perform any actions for which its definer does not have the requisite privileges. For example, suppose that **jon@ghidora** has the [**EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_event) privilege for **myschema**. Suppose also that this user has the [**SELECT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_select) privilege for **myschema**, but no other privileges for this schema. It is possible for **jon@ghidora** to create a new event such as this one:

CREATE EVENT e\_store\_ts

ON SCHEDULE

EVERY 10 SECOND

DO

INSERT INTO myschema.mytable VALUES (UNIX\_TIMESTAMP());

The user waits for a minute or so, and then performs a **SELECT \* FROM mytable;** query, expecting to see several new rows in the table. Instead, the table is empty. Since the user does not have the [**INSERT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_insert) privilege for the table in question, the event has no effect.

If you inspect the MySQL error log (***hostname***.err), you can see that the event is executing, but the action it is attempting to perform fails:

2013-09-24T12:41:31.261992Z 25 [ERROR] Event Scheduler:

[jon@ghidora][cookbook.e\_store\_ts] INSERT command denied to user

'jon'@'ghidora' for table 'mytable'

2013-09-24T12:41:31.262022Z 25 [Note] Event Scheduler:

[jon@ghidora].[myschema.e\_store\_ts] event execution failed.

2013-09-24T12:41:41.271796Z 26 [ERROR] Event Scheduler:

[jon@ghidora][cookbook.e\_store\_ts] INSERT command denied to user

'jon'@'ghidora' for table 'mytable'

2013-09-24T12:41:41.272761Z 26 [Note] Event Scheduler:

[jon@ghidora].[myschema.e\_store\_ts] event execution failed.

Since this user very likely does not have access to the error log, it is possible to verify whether the event's action statement is valid by executing it directly:

mysql> **INSERT INTO myschema.mytable VALUES (UNIX\_TIMESTAMP());**

ERROR 1142 (42000): INSERT command denied to user

'jon'@'ghidora' for table 'mytable'

Inspection of the [**INFORMATION\_SCHEMA.EVENTS**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\information-schema.html#information-schema-events-table) table shows that **e\_store\_ts** exists and is enabled, but its **LAST\_EXECUTED** column is **NULL**:

mysql> **SELECT \* FROM INFORMATION\_SCHEMA.EVENTS**

> **WHERE EVENT\_NAME='e\_store\_ts'**

> **AND EVENT\_SCHEMA='myschema'\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

EVENT\_CATALOG: NULL

EVENT\_SCHEMA: myschema

EVENT\_NAME: e\_store\_ts

DEFINER: jon@ghidora

EVENT\_BODY: SQL

EVENT\_DEFINITION: INSERT INTO myschema.mytable VALUES (UNIX\_TIMESTAMP())

EVENT\_TYPE: RECURRING

EXECUTE\_AT: NULL

INTERVAL\_VALUE: 5

INTERVAL\_FIELD: SECOND

SQL\_MODE: NULL

STARTS: 0000-00-00 00:00:00

ENDS: 0000-00-00 00:00:00

STATUS: ENABLED

ON\_COMPLETION: NOT PRESERVE

CREATED: 2006-02-09 22:36:06

LAST\_ALTERED: 2006-02-09 22:36:06

LAST\_EXECUTED: NULL

EVENT\_COMMENT:

1 row in set (0.00 sec)

To rescind the [**EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_event) privilege, use the [**REVOKE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#revoke) statement. In this example, the [**EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_event) privilege on the schema **myschema** is removed from the **jon@ghidora** user account:

REVOKE EVENT ON myschema.\* FROM jon@ghidora;

**Important**

Revoking the [**EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_event) privilege from a user does not delete or disable any events that may have been created by that user.

An event is not migrated or dropped as a result of renaming or dropping the user who created it.

Suppose that the user **jon@ghidora** has been granted the [**EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_event) and [**INSERT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_insert) privileges on the **myschema** schema. This user then creates the following event:

CREATE EVENT e\_insert

ON SCHEDULE

EVERY 7 SECOND

DO

INSERT INTO myschema.mytable;

After this event has been created, **root** revokes the [**EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_event) privilege for **jon@ghidora**. However, **e\_insert** continues to execute, inserting a new row into **mytable** each seven seconds. The same would be true if **root** had issued either of these statements:

* **DROP USER jon@ghidora;**
* **RENAME USER jon@ghidora TO someotherguy@ghidora;**

You can verify that this is true by examining the [**INFORMATION\_SCHEMA.EVENTS**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\information-schema.html#information-schema-events-table) table (see [Section 26.3.14, “The INFORMATION\_SCHEMA EVENTS Table”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\information-schema.html#information-schema-events-table)) before and after issuing a [**DROP USER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#drop-user) or [**RENAME USER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#rename-user) statement.

Event definitions are stored in the data dictionary. To drop an event created by another user account, you must be the MySQL **root** user or another user with the necessary privileges.

Users' [**EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_event) privileges are stored in the **Event\_priv** columns of the **mysql.user** and **mysql.db** tables. In both cases, this column holds one of the values '**Y**' or '**N**'. '**N**' is the default. **mysql.user.Event\_priv** is set to '**Y**' for a given user only if that user has the global [**EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_event) privilege (that is, if the privilege was bestowed using **GRANT EVENT ON \*.\***). For a schema-level [**EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_event) privilege, [**GRANT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#grant) creates a row in **mysql.db** and sets that row's **Db** column to the name of the schema, the **User** column to the name of the user, and the **Event\_priv** column to '**Y**'. There should never be any need to manipulate these tables directly, since the [**GRANT EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#grant) and **REVOKE EVENT** statements perform the required operations on them.

Five status variables provide counts of event-related operations (but not of statements executed by events; see [Section 25.8, “Restrictions on Stored Programs”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-program-restrictions)). These are:

* **Com\_create\_event**: The number of [**CREATE EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-event) statements executed since the last server restart.
* **Com\_alter\_event**: The number of [**ALTER EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-event) statements executed since the last server restart.
* **Com\_drop\_event**: The number of [**DROP EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#drop-event) statements executed since the last server restart.
* **Com\_show\_create\_event**: The number of [**SHOW CREATE EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-create-event) statements executed since the last server restart.
* **Com\_show\_events**: The number of [**SHOW EVENTS**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-events) statements executed since the last server restart.

You can view current values for all of these at one time by running the statement **SHOW STATUS LIKE '%event%';**.

## 25.5 Using Views

[25.5.1 View Syntax](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#view-syntax)

[25.5.2 View Processing Algorithms](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#view-algorithms)

[25.5.3 Updatable and Insertable Views](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#view-updatability)

[25.5.4 The View WITH CHECK OPTION Clause](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#view-check-option)

[25.5.5 View Metadata](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#view-metadata)

MySQL supports views, including updatable views. Views are stored queries that when invoked produce a result set. A view acts as a virtual table.

The following discussion describes the syntax for creating and dropping views, and shows some examples of how to use them.

### Additional Resources

* You may find the [Views User Forum](https://forums.mysql.com/list.php?100) of use when working with views.
* For answers to some commonly asked questions regarding views in MySQL, see [Section A.6, “MySQL 8.0 FAQ: Views”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\faqs.html#faqs-views).
* There are some restrictions on the use of views; see [Section 25.9, “Restrictions on Views”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#view-restrictions).

### 25.5.1 View Syntax

The [**CREATE VIEW**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-view) statement creates a new view (see [Section 13.1.23, “CREATE VIEW Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-view)). To alter the definition of a view or drop a view, use [**ALTER VIEW**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-view) (see [Section 13.1.11, “ALTER VIEW Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-view)), or [**DROP VIEW**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#drop-view) (see [Section 13.1.35, “DROP VIEW Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#drop-view)).

A view can be created from many kinds of [**SELECT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#select) statements. It can refer to base tables or other views. It can use joins, [**UNION**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#union), and subqueries. The [**SELECT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#select) need not even refer to any tables. The following example defines a view that selects two columns from another table, as well as an expression calculated from those columns:

mysql> **CREATE TABLE t (qty INT, price INT);**

mysql> **INSERT INTO t VALUES(3, 50), (5, 60);**

mysql> **CREATE VIEW v AS SELECT qty, price, qty\*price AS value FROM t;**

mysql> **SELECT \* FROM v;**

+------+-------+-------+

| qty | price | value |

+------+-------+-------+

| 3 | 50 | 150 |

| 5 | 60 | 300 |

+------+-------+-------+

mysql> **SELECT \* FROM v WHERE qty = 5;**

+------+-------+-------+

| qty | price | value |

+------+-------+-------+

| 5 | 60 | 300 |

+------+-------+-------+

### 25.5.2 View Processing Algorithms

The optional **ALGORITHM** clause for [**CREATE VIEW**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-view) or [**ALTER VIEW**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-view) is a MySQL extension to standard SQL. It affects how MySQL processes the view. **ALGORITHM** takes three values: **MERGE**, **TEMPTABLE**, or **UNDEFINED**.

* For **MERGE**, the text of a statement that refers to the view and the view definition are merged such that parts of the view definition replace corresponding parts of the statement.
* For **TEMPTABLE**, the results from the view are retrieved into a temporary table, which then is used to execute the statement.
* For **UNDEFINED**, MySQL chooses which algorithm to use. It prefers **MERGE** over **TEMPTABLE** if possible, because **MERGE** is usually more efficient and because a view cannot be updatable if a temporary table is used.
* If no **ALGORITHM** clause is present, the default algorithm is determined by the value of the **[derived\_merge](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\optimization.html" \l "optflag_derived-merge)** flag of the **[optimizer\_switch](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_optimizer_switch)** system variable. For additional discussion, see [Section 8.2.2.4, “Optimizing Derived Tables, View References, and Common Table Expressions with Merging or Materialization”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\optimization.html#derived-table-optimization).

A reason to specify **TEMPTABLE** explicitly is that locks can be released on underlying tables after the temporary table has been created and before it is used to finish processing the statement. This might result in quicker lock release than the **MERGE** algorithm so that other clients that use the view are not blocked as long.

A view algorithm can be **UNDEFINED** for three reasons:

* No **ALGORITHM** clause is present in the [**CREATE VIEW**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-view) statement.
* The [**CREATE VIEW**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-view) statement has an explicit **ALGORITHM = UNDEFINED** clause.
* **ALGORITHM = MERGE** is specified for a view that can be processed only with a temporary table. In this case, MySQL generates a warning and sets the algorithm to **UNDEFINED**.

As mentioned earlier, **MERGE** is handled by merging corresponding parts of a view definition into the statement that refers to the view. The following examples briefly illustrate how the **MERGE** algorithm works. The examples assume that there is a view **v\_merge** that has this definition:

CREATE ALGORITHM = MERGE VIEW v\_merge (vc1, vc2) AS

SELECT c1, c2 FROM t WHERE c3 > 100;

Example 1: Suppose that we issue this statement:

SELECT \* FROM v\_merge;

MySQL handles the statement as follows:

* **v\_merge** becomes **t**
* **\*** becomes **vc1, vc2**, which corresponds to **c1, c2**
* The view **WHERE** clause is added

The resulting statement to be executed becomes:

SELECT c1, c2 FROM t WHERE c3 > 100;

Example 2: Suppose that we issue this statement:

SELECT \* FROM v\_merge WHERE vc1 < 100;

This statement is handled similarly to the previous one, except that **vc1 < 100** becomes **c1 < 100** and the view **WHERE** clause is added to the statement **WHERE** clause using an [**AND**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#operator_and) connective (and parentheses are added to make sure the parts of the clause are executed with correct precedence). The resulting statement to be executed becomes:

SELECT c1, c2 FROM t WHERE (c3 > 100) AND (c1 < 100);

Effectively, the statement to be executed has a **WHERE** clause of this form:

WHERE (select WHERE) AND (view WHERE)

If the **MERGE** algorithm cannot be used, a temporary table must be used instead. Constructs that prevent merging are the same as those that prevent merging in derived tables and common table expressions. Examples are **SELECT DISTINCT** or **LIMIT** in the subquery. For details, see [Section 8.2.2.4, “Optimizing Derived Tables, View References, and Common Table Expressions with Merging or Materialization”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\optimization.html#derived-table-optimization).

### 25.5.3 Updatable and Insertable Views

Some views are updatable and references to them can be used to specify tables to be updated in data change statements. That is, you can use them in statements such as [**UPDATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#update), [**DELETE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#delete), or [**INSERT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#insert) to update the contents of the underlying table. Derived tables and common table expressions can also be specified in multiple-table [**UPDATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#update) and [**DELETE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#delete) statements, but can only be used for reading data to specify rows to be updated or deleted. Generally, the view references must be updatable, meaning that they may be merged and not materialized. Composite views have more complex rules.

For a view to be updatable, there must be a one-to-one relationship between the rows in the view and the rows in the underlying table. There are also certain other constructs that make a view nonupdatable. To be more specific, a view is not updatable if it contains any of the following:

* Aggregate functions or window functions ([**SUM()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_sum), [**MIN()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_min), [**MAX()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_max), [**COUNT()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_count), and so forth)
* **DISTINCT**
* **GROUP BY**
* **HAVING**
* [**UNION**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#union) or [**UNION ALL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#union)
* Subquery in the select list

Nondependent subqueries in the select list fail for [**INSERT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#insert), but are okay for [**UPDATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#update), [**DELETE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#delete). For dependent subqueries in the select list, no data change statements are permitted.

* Certain joins (see additional join discussion later in this section)
* Reference to nonupdatable view in the **FROM** clause
* Subquery in the **WHERE** clause that refers to a table in the **FROM** clause
* Refers only to literal values (in this case, there is no underlying table to update)
* **ALGORITHM = TEMPTABLE** (use of a temporary table always makes a view nonupdatable)
* Multiple references to any column of a base table (fails for [**INSERT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#insert), okay for [**UPDATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#update), [**DELETE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#delete))

A generated column in a view is considered updatable because it is possible to assign to it. However, if such a column is updated explicitly, the only permitted value is **DEFAULT**. For information about generated columns, see [Section 13.1.20.8, “CREATE TABLE and Generated Columns”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-table-generated-columns).

It is sometimes possible for a multiple-table view to be updatable, assuming that it can be processed with the **MERGE** algorithm. For this to work, the view must use an inner join (not an outer join or a [**UNION**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#union)). Also, only a single table in the view definition can be updated, so the **SET** clause must name only columns from one of the tables in the view. Views that use [**UNION ALL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#union) are not permitted even though they might be theoretically updatable.

With respect to insertability (being updatable with [**INSERT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#insert) statements), an updatable view is insertable if it also satisfies these additional requirements for the view columns:

* There must be no duplicate view column names.
* The view must contain all columns in the base table that do not have a default value.
* The view columns must be simple column references. They must not be expressions, such as these:
* 3.14159
* col1 + 3
* UPPER(col2)
* col3 / col4
* (***subquery***)

MySQL sets a flag, called the view updatability flag, at [**CREATE VIEW**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-view) time. The flag is set to **YES** (true) if [**UPDATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#update) and [**DELETE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#delete) (and similar operations) are legal for the view. Otherwise, the flag is set to **NO** (false). The **IS\_UPDATABLE** column in the [**INFORMATION\_SCHEMA.VIEWS**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\information-schema.html#information-schema-views-table) table displays the status of this flag. It means that the server always knows whether a view is updatable.

If a view is not updatable, statements such [**UPDATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#update), [**DELETE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#delete), and [**INSERT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#insert) are illegal and are rejected. (Even if a view is updatable, it might not be possible to insert into it, as described elsewhere in this section.)

The updatability of views may be affected by the value of the **[updatable\_views\_with\_limit](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\server-administration.html" \l "sysvar_updatable_views_with_limit)** system variable. See [Section 5.1.8, “Server System Variables”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#server-system-variables).

For the following discussion, suppose that these tables and views exist:

CREATE TABLE t1 (x INTEGER);

CREATE TABLE t2 (c INTEGER);

CREATE VIEW vmat AS SELECT SUM(x) AS s FROM t1;

CREATE VIEW vup AS SELECT \* FROM t2;

CREATE VIEW vjoin AS SELECT \* FROM vmat JOIN vup ON vmat.s=vup.c;

[**INSERT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#insert), [**UPDATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#update), and [**DELETE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#delete) statements are permitted as follows:

* [**INSERT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#insert): The insert table of an [**INSERT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#insert) statement may be a view reference that is merged. If the view is a join view, all components of the view must be updatable (not materialized). For a multiple-table updatable view, [**INSERT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#insert) can work if it inserts into a single table.

This statement is invalid because one component of the join view is nonupdatable:

INSERT INTO vjoin (c) VALUES (1);

This statement is valid; the view contains no materialized components:

INSERT INTO vup (c) VALUES (1);

* [**UPDATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#update): The table or tables to be updated in an [**UPDATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#update) statement may be view references that are merged. If a view is a join view, at least one component of the view must be updatable (this differs from [**INSERT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#insert)).

In a multiple-table [**UPDATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#update) statement, the updated table references of the statement must be base tables or updatable view references. Nonupdated table references may be materialized views or derived tables.

This statement is valid; column **c** is from the updatable part of the join view:

UPDATE vjoin SET c=c+1;

This statement is invalid; column **x** is from the nonupdatable part:

UPDATE vjoin SET x=x+1;

This statement is valid; the updated table reference of the multiple-table [**UPDATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#update) is an updatable view (**vup**):

UPDATE vup JOIN (SELECT SUM(x) AS s FROM t1) AS dt ON ...

SET c=c+1;

This statement is invalid; it tries to update a materialized derived table:

UPDATE vup JOIN (SELECT SUM(x) AS s FROM t1) AS dt ON ...

SET s=s+1;

* [**DELETE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#delete): The table or tables to be deleted from in a [**DELETE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#delete) statement must be merged views. Join views are not allowed (this differs from [**INSERT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#insert) and [**UPDATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#update)).

This statement is invalid because the view is a join view:

DELETE vjoin WHERE ...;

This statement is valid because the view is a merged (updatable) view:

DELETE vup WHERE ...;

This statement is valid because it deletes from a merged (updatable) view:

DELETE vup FROM vup JOIN (SELECT SUM(x) AS s FROM t1) AS dt ON ...;

Additional discussion and examples follow.

Earlier discussion in this section pointed out that a view is not insertable if not all columns are simple column references (for example, if it contains columns that are expressions or composite expressions). Although such a view is not insertable, it can be updatable if you update only columns that are not expressions. Consider this view:

CREATE VIEW v AS SELECT col1, 1 AS col2 FROM t;

This view is not insertable because **col2** is an expression. But it is updatable if the update does not try to update **col2**. This update is permissible:

UPDATE v SET col1 = 0;

This update is not permissible because it attempts to update an expression column:

UPDATE v SET col2 = 0;

If a table contains an **AUTO\_INCREMENT** column, inserting into an insertable view on the table that does not include the **AUTO\_INCREMENT** column does not change the value of [**LAST\_INSERT\_ID()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_last-insert-id), because the side effects of inserting default values into columns not part of the view should not be visible.

### 25.5.4 The View WITH CHECK OPTION Clause

The **WITH CHECK OPTION** clause can be given for an updatable view to prevent inserts to rows for which the **WHERE** clause in the ***select\_statement*** is not true. It also prevents updates to rows for which the **WHERE** clause is true but the update would cause it to be not true (in other words, it prevents visible rows from being updated to nonvisible rows).

In a **WITH CHECK OPTION** clause for an updatable view, the **LOCAL** and **CASCADED** keywords determine the scope of check testing when the view is defined in terms of another view. When neither keyword is given, the default is **CASCADED**.

**WITH CHECK OPTION** testing is standard-compliant:

* With **LOCAL**, the view **WHERE** clause is checked, then checking recurses to underlying views and applies the same rules.
* With **CASCADED**, the view **WHERE** clause is checked, then checking recurses to underlying views, adds **WITH CASCADED CHECK OPTION** to them (for purposes of the check; their definitions remain unchanged), and applies the same rules.
* With no check option, the view **WHERE** clause is not checked, then checking recurses to underlying views, and applies the same rules.

Consider the definitions for the following table and set of views:

CREATE TABLE t1 (a INT);

CREATE VIEW v1 AS SELECT \* FROM t1 WHERE a < 2

WITH CHECK OPTION;

CREATE VIEW v2 AS SELECT \* FROM v1 WHERE a > 0

WITH LOCAL CHECK OPTION;

CREATE VIEW v3 AS SELECT \* FROM v1 WHERE a > 0

WITH CASCADED CHECK OPTION;

Here the **v2** and **v3** views are defined in terms of another view, **v1**.

Inserts for **v2** are checked against its **LOCAL** check option, then the check recurses to **v1** and the rules are applied again. The rules for **v1** cause a check failure. The check for **v3** also fails:

mysql> **INSERT INTO v2 VALUES (2);**

ERROR 1369 (HY000): CHECK OPTION failed 'test.v2'

mysql> **INSERT INTO v3 VALUES (2);**

ERROR 1369 (HY000): CHECK OPTION failed 'test.v3'

### 25.5.5 View Metadata

To obtain metadata about views:

* Query the [**VIEWS**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\information-schema.html#information-schema-views-table) table of the **INFORMATION\_SCHEMA** database. See [Section 26.3.48, “The INFORMATION\_SCHEMA VIEWS Table”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\information-schema.html#information-schema-views-table).
* Use the [**SHOW CREATE VIEW**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-create-view) statement. See [Section 13.7.7.13, “SHOW CREATE VIEW Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-create-view).

## 25.6 Stored Object Access Control

Stored programs (procedures, functions, triggers, and events) and views are defined prior to use and, when referenced, execute within a security context that determines their privileges. The privileges applicable to execution of a stored object are controlled by its **DEFINER** attribute and **SQL SECURITY** characteristic.

* [The DEFINER Attribute](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-objects-security-definer)
* [The SQL SECURITY Characteristic](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-objects-security-sql-security)
* [Examples](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-objects-security-examples)
* [Orphan Stored Objects](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-objects-security-orphan-objects)
* [Risk-Minimization Guidelines](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-objects-security-guidelines)

### The DEFINER Attribute

A stored object definition can include a **DEFINER** attribute that names a MySQL account. If a definition omits the **DEFINER** attribute, the default object definer is the user who creates it.

The following rules determine which accounts you can specify as the **DEFINER** attribute for a stored object:

* If you have the [**SET\_USER\_ID**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_set-user-id) privilege (or the deprecated [**SUPER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_super) privilege), you can specify any account as the **DEFINER** attribute. If the account does not exist, a warning is generated. Additionally, to set a stored object **DEFINER** attribute to an account that has the [**SYSTEM\_USER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_system-user) privilege, you must have the [**SYSTEM\_USER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_system-user) privilege.
* Otherwise, the only permitted account is your own, specified either literally or as [**CURRENT\_USER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_current-user) or [**CURRENT\_USER()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_current-user). You cannot set the definer to any other account.

Creating a stored object with a nonexistent **DEFINER** account creates an orphan object, which may have negative consequences; see [Orphan Stored Objects](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-objects-security-orphan-objects).

### The SQL SECURITY Characteristic

For stored routines (procedures and functions) and views, the object definition can include an **SQL SECURITY** characteristic with a value of **DEFINER** or **INVOKER** to specify whether the object executes in definer or invoker context. If the definition omits the **SQL SECURITY** characteristic, the default is definer context.

Triggers and events have no **SQL SECURITY** characteristic and always execute in definer context. The server invokes these objects automatically as necessary, so there is no invoking user.

Definer and invoker security contexts differ as follows:

* A stored object that executes in definer security context executes with the privileges of the account named by its **DEFINER** attribute. These privileges may be entirely different from those of the invoking user. The invoker must have appropriate privileges to reference the object (for example, [**EXECUTE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_execute) to call a stored procedure or [**SELECT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_select) to select from a view), but during object execution, the invoker's privileges are ignored and only the **DEFINER** account privileges matter. If the **DEFINER** account has few privileges, the object is correspondingly limited in the operations it can perform. If the **DEFINER** account is highly privileged (such as an administrative account), the object can perform powerful operations no matter who invokes it.
* A stored routine or view that executes in invoker security context can perform only operations for which the invoker has privileges. The **DEFINER** attribute has no effect on object execution.

### Examples

Consider the following stored procedure, which is declared with **SQL SECURITY DEFINER** to execute in definer security context:

CREATE DEFINER = 'admin'@'localhost' PROCEDURE p1()

SQL SECURITY DEFINER

BEGIN

UPDATE t1 SET counter = counter + 1;

END;

Any user who has the [**EXECUTE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_execute) privilege for **p1** can invoke it with a [**CALL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#call) statement. However, when **p1** executes, it does so in definer security context and thus executes with the privileges of **'admin'@'localhost'**, the account named as its **DEFINER** attribute. This account must have the [**EXECUTE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_execute) privilege for **p1** as well as the [**UPDATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_update) privilege for the table **t1** referenced within the object body. Otherwise, the procedure fails.

Now consider this stored procedure, which is identical to **p1** except that its **SQL SECURITY** characteristic is **INVOKER**:

CREATE DEFINER = 'admin'@'localhost' PROCEDURE p2()

SQL SECURITY INVOKER

BEGIN

UPDATE t1 SET counter = counter + 1;

END;

Unlike **p1**, **p2** executes in invoker security context and thus with the privileges of the invoking user regardless of the **DEFINER** attribute value. **p2** fails if the invoker lacks the [**EXECUTE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_execute) privilege for **p2** or the [**UPDATE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_update) privilege for the table **t1**.

### Orphan Stored Objects

An orphan stored object is one for which its **DEFINER** attribute names a nonexistent account:

* An orphan stored object can be created by specifying a nonexistent **DEFINER** account at object-creation time.
* An existing stored object can become orphaned through execution of a [**DROP USER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#drop-user) statement that drops the object **DEFINER** account, or a [**RENAME USER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#rename-user) statement that renames the object **DEFINER** account.

An orphan stored object may be problematic in these ways:

* Because the **DEFINER** account does not exist, the object may not work as expected if it executes in definer security context:
  + For a stored routine, an error occurs at routine execution time if the **SQL SECURITY** value is **DEFINER** but the definer account does not exist.
  + For a trigger, it is not a good idea for trigger activation to occur until the account actually does exist. Otherwise, the behavior with respect to privilege checking is undefined.
  + For an event, an error occurs at event execution time if the account does not exist.
  + For a view, an error occurs when the view is referenced if the **SQL SECURITY** value is **DEFINER** but the definer account does not exist.
* The object may present a security risk if the nonexistent **DEFINER** account is subsequently re-created for a purpose unrelated to the object. In this case, the account “adopts” the object and, with the appropriate privileges, is able to execute it even if that is not intended.

As of MySQL 8.0.22, the server imposes additional account-management security checks designed to prevent operations that (perhaps inadvertently) cause stored objects to become orphaned or that cause adoption of stored objects that are currently orphaned:

* [**DROP USER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#drop-user) fails with an error if any account to be dropped is named as the **DEFINER** attribute for any stored object. (That is, the statement fails if dropping an account would cause a stored object to become orphaned.)
* [**RENAME USER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#rename-user) fails with an error if any account to be renamed is named as the **DEFINER** attribute for any stored object. (That is, the statement fails if renaming an account would cause a stored object to become orphaned.)
* [**CREATE USER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-user) fails with an error if any account to be created is named as the **DEFINER** attribute for any stored object. (That is, the statement fails if creating an account would cause the account to adopt a currently orphaned stored object.)

In certain situations, it may be necessary to deliberately execute those account-management statements even when they would otherwise fail. To make this possible, if a user has the [**SET\_USER\_ID**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_set-user-id) privilege, that privilege overrides the orphan object security checks and the statements succeed with a warning rather than failing with an error.

To obtain information about the accounts used as stored object definers in a MySQL installation, query the **INFORMATION\_SCHEMA**.

This query identifies which **INFORMATION\_SCHEMA** tables describe objects that have a **DEFINER** attribute:

mysql> **SELECT TABLE\_SCHEMA, TABLE\_NAME FROM INFORMATION\_SCHEMA.COLUMNS**

**WHERE COLUMN\_NAME = 'DEFINER';**

+--------------------+------------+

| TABLE\_SCHEMA | TABLE\_NAME |

+--------------------+------------+

| information\_schema | EVENTS |

| information\_schema | ROUTINES |

| information\_schema | TRIGGERS |

| information\_schema | VIEWS |

+--------------------+------------+

The result tells you which tables to query to discover which stored object **DEFINER** values exist and which objects have a particular **DEFINER** value:

* To identify which **DEFINER** values exist in each table, use these queries:
* SELECT DISTINCT DEFINER FROM INFORMATION\_SCHEMA.EVENTS;
* SELECT DISTINCT DEFINER FROM INFORMATION\_SCHEMA.ROUTINES;
* SELECT DISTINCT DEFINER FROM INFORMATION\_SCHEMA.TRIGGERS;
* SELECT DISTINCT DEFINER FROM INFORMATION\_SCHEMA.VIEWS;

The query results are significant for any account displayed as follows:

* + If the account exists, dropping or renaming it causes stored objects to become orphaned. If you plan to drop or rename the account, consider first dropping its associated stored objects or redefining them to have a different definer.
  + If the account does not exist, creating it causes it to adopt currently orphaned stored objects. If you plan to create the account, consider whether the orphaned objects should be associated with it. If not, redefine them to have a different definer.

To redefine an object with a different definer, you can use [**ALTER EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-event) or [**ALTER VIEW**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-view) to directly modify the **DEFINER** account of events and views. For stored procedures and functions and for triggers, you must drop the object and re-create it to assign a different **DEFINER** account

* To identify which objects have a given **DEFINER** account, use these queries, substituting the account of interest for ***user\_name*@*host\_name***:
* SELECT EVENT\_SCHEMA, EVENT\_NAME FROM INFORMATION\_SCHEMA.EVENTS
* WHERE DEFINER = '***user\_name***@***host\_name***';
* SELECT ROUTINE\_SCHEMA, ROUTINE\_NAME, ROUTINE\_TYPE
* FROM INFORMATION\_SCHEMA.ROUTINES
* WHERE DEFINER = '***user\_name***@***host\_name***';
* SELECT TRIGGER\_SCHEMA, TRIGGER\_NAME FROM INFORMATION\_SCHEMA.TRIGGERS
* WHERE DEFINER = '***user\_name***@***host\_name***';
* SELECT TABLE\_SCHEMA, TABLE\_NAME FROM INFORMATION\_SCHEMA.VIEWS
* WHERE DEFINER = '***user\_name***@***host\_name***';

For the [**ROUTINES**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\information-schema.html#information-schema-routines-table) table, the query includes the **ROUTINE\_TYPE** column so that output rows distinguish whether the **DEFINER** is for a stored procedure or stored function.

If the account you are searching for does not exist, any objects displayed by those queries are orphan objects.

### Risk-Minimization Guidelines

To minimize the risk potential for stored object creation and use, follow these guidelines:

* Do not create orphan stored objects; that is, objects for which the **DEFINER** attribute names a nonexistent account. Do not cause stored objects to become orphaned by dropping or renaming an account named by the **DEFINER** attribute of any existing object.
* For a stored routine or view, use **SQL SECURITY INVOKER** in the object definition when possible so that it can be used only by users with permissions appropriate for the operations performed by the object.
* If you create definer-context stored objects while using an account that has the [**SET\_USER\_ID**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_set-user-id) privilege (or the deprecated [**SUPER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_super) privilege), specify an explicit **DEFINER** attribute that names an account possessing only the privileges required for the operations performed by the object. Specify a highly privileged **DEFINER** account only when absolutely necessary.
* Administrators can prevent users from creating stored objects that specify highly privileged **DEFINER** accounts by not granting them the [**SET\_USER\_ID**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_set-user-id) privilege (or the deprecated [**SUPER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_super) privilege).
* Definer-context objects should be written keeping in mind that they may be able to access data for which the invoking user has no privileges. In some cases, you can prevent references to these objects by not granting unauthorized users particular privileges:
  + A stored routine cannot be referenced by a user who does not have the [**EXECUTE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_execute) privilege for it.
  + A view cannot be referenced by a user who does not have the appropriate privilege for it ([**SELECT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_select) to select from it, [**INSERT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_insert) to insert into it, and so forth).

However, no such control exists for triggers and events because they always execute in definer context. The server invokes these objects automatically as necessary, and users do not reference them directly:

* + A trigger is activated by access to the table with which it is associated, even ordinary table accesses by users with no special privileges.
  + An event is executed by the server on a scheduled basis.

In both cases, if the **DEFINER** account is highly privileged, the object may be able to perform sensitive or dangerous operations. This remains true if the privileges needed to create the object are revoked from the account of the user who created it. Administrators should be especially careful about granting users object-creation privileges.

## 25.7 Stored Program Binary Logging

The binary log contains information about SQL statements that modify database contents. This information is stored in the form of “events” that describe the modifications. (Binary log events differ from scheduled event stored objects.) The binary log has two important purposes:

* For replication, the binary log is used on source replication servers as a record of the statements to be sent to replica servers. The source sends the events contained in its binary log to its replicas, which execute those events to make the same data changes that were made on the source. See [Section 17.2, “Replication Implementation”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\replication.html#replication-implementation).
* Certain data recovery operations require use of the binary log. After a backup file has been restored, the events in the binary log that were recorded after the backup was made are re-executed. These events bring databases up to date from the point of the backup. See [Section 7.3.2, “Using Backups for Recovery”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\backup-and-recovery.html#recovery-from-backups).

However, if logging occurs at the statement level, there are certain binary logging issues with respect to stored programs (stored procedures and functions, triggers, and events):

* In some cases, a statement might affect different sets of rows on source and replica.
* Replicated statements executed on a replica are processed by the replica's applier thread. Unless you implement replication privilege checks, which are available from MySQL 8.0.18 (see [Section 17.3.3, “Replication Privilege Checks”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\replication.html#replication-privilege-checks)), the applier thread has full privileges. In this situation, it is possible for a procedure to follow different execution paths on source and replica servers, so a user could write a routine containing a dangerous statement that executes only on the replica.
* If a stored program that modifies data is nondeterministic, it is not repeatable. This can result in different data on source and replica, or cause restored data to differ from the original data.

This section describes how MySQL handles binary logging for stored programs. It states the current conditions that the implementation places on the use of stored programs, and what you can do to avoid logging problems. It also provides additional information about the reasons for these conditions.

Unless noted otherwise, the remarks here assume that binary logging is enabled on the server (see [Section 5.4.4, “The Binary Log”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\server-administration.html#binary-log).) If the binary log is not enabled, replication is not possible, nor is the binary log available for data recovery. From MySQL 8.0, binary logging is enabled by default, and is only disabled if you specify the [--skip-log-bin](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\replication.html#option_mysqld_log-bin) or [--disable-log-bin](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\replication.html#option_mysqld_log-bin) option at startup.

In general, the issues described here result when binary logging occurs at the SQL statement level (statement-based binary logging). If you use row-based binary logging, the log contains changes made to individual rows as a result of executing SQL statements. When routines or triggers execute, row changes are logged, not the statements that make the changes. For stored procedures, this means that the [**CALL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#call) statement is not logged. For stored functions, row changes made within the function are logged, not the function invocation. For triggers, row changes made by the trigger are logged. On the replica side, only the row changes are seen, not the stored program invocation.

Mixed format binary logging (**[binlog\_format=MIXED](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\replication.html" \l "sysvar_binlog_format)**) uses statement-based binary logging, except for cases where only row-based binary logging is guaranteed to lead to proper results. With mixed format, when a stored function, stored procedure, trigger, event, or prepared statement contains anything that is not safe for statement-based binary logging, the entire statement is marked as unsafe and logged in row format. The statements used to create and drop procedures, functions, triggers, and events are always safe, and are logged in statement format. For more information about row-based, mixed, and statement-based logging, and how safe and unsafe statements are determined, see [Section 17.2.1, “Replication Formats”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\replication.html#replication-formats).

The conditions on the use of stored functions in MySQL can be summarized as follows. These conditions do not apply to stored procedures or Event Scheduler events and they do not apply unless binary logging is enabled.

* To create or alter a stored function, you must have the [**SET\_USER\_ID**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_set-user-id) privilege (or the deprecated [**SUPER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_super) privilege), in addition to the [**CREATE ROUTINE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_create-routine) or [**ALTER ROUTINE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_alter-routine) privilege that is normally required. (Depending on the **DEFINER** value in the function definition, [**SET\_USER\_ID**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_set-user-id) or [**SUPER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_super) might be required regardless of whether binary logging is enabled. See [Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Statements”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-procedure).)
* When you create a stored function, you must declare either that it is deterministic or that it does not modify data. Otherwise, it may be unsafe for data recovery or replication.

By default, for a [**CREATE FUNCTION**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-function) statement to be accepted, at least one of **DETERMINISTIC**, **NO SQL**, or **READS SQL DATA** must be specified explicitly. Otherwise an error occurs:

ERROR 1418 (HY000): This function has none of DETERMINISTIC, NO SQL,

or READS SQL DATA in its declaration and binary logging is enabled

(you \*might\* want to use the less safe log\_bin\_trust\_function\_creators

variable)

This function is deterministic (and does not modify data), so it is safe:

CREATE FUNCTION f1(i INT)

RETURNS INT

DETERMINISTIC

READS SQL DATA

BEGIN

RETURN i;

END;

This function uses [**UUID()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_uuid), which is not deterministic, so the function also is not deterministic and is not safe:

CREATE FUNCTION f2()

RETURNS CHAR(36) CHARACTER SET utf8

BEGIN

RETURN UUID();

END;

This function modifies data, so it may not be safe:

CREATE FUNCTION f3(p\_id INT)

RETURNS INT

BEGIN

UPDATE t SET modtime = NOW() WHERE id = p\_id;

RETURN ROW\_COUNT();

END;

Assessment of the nature of a function is based on the “honesty” of the creator. MySQL does not check that a function declared **DETERMINISTIC** is free of statements that produce nondeterministic results.

* When you attempt to execute a stored function, if **[binlog\_format=STATEMENT](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\replication.html" \l "sysvar_binlog_format)** is set, the **DETERMINISTIC** keyword must be specified in the function definition. If this is not the case, an error is generated and the function does not run, unless **[log\_bin\_trust\_function\_creators=1](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\replication.html" \l "sysvar_log_bin_trust_function_creators)** is specified to override this check (see below). For recursive function calls, the **DETERMINISTIC** keyword is required on the outermost call only. If row-based or mixed binary logging is in use, the statement is accepted and replicated even if the function was defined without the **DETERMINISTIC** keyword.
* Because MySQL does not check if a function really is deterministic at creation time, the invocation of a stored function with the **DETERMINISTIC** keyword might carry out an action that is unsafe for statement-based logging, or invoke a function or procedure containing unsafe statements. If this occurs when **[binlog\_format=STATEMENT](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\replication.html" \l "sysvar_binlog_format)** is set, a warning message is issued. If row-based or mixed binary logging is in use, no warning is issued, and the statement is replicated in row-based format.
* To relax the preceding conditions on function creation (that you must have the [**SUPER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_super) privilege and that a function must be declared deterministic or to not modify data), set the global **[log\_bin\_trust\_function\_creators](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\replication.html" \l "sysvar_log_bin_trust_function_creators)** system variable to 1. By default, this variable has a value of 0, but you can change it like this:
* mysql> **SET GLOBAL log\_bin\_trust\_function\_creators = 1;**

You can also set this variable at server startup.

If binary logging is not enabled, **[log\_bin\_trust\_function\_creators](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\replication.html" \l "sysvar_log_bin_trust_function_creators)** does not apply. [**SUPER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_super) is not required for function creation unless, as described previously, the **DEFINER** value in the function definition requires it.

* For information about built-in functions that may be unsafe for replication (and thus cause stored functions that use them to be unsafe as well), see [Section 17.5.1, “Replication Features and Issues”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\replication.html#replication-features).

Triggers are similar to stored functions, so the preceding remarks regarding functions also apply to triggers with the following exception: [**CREATE TRIGGER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-trigger) does not have an optional **DETERMINISTIC** characteristic, so triggers are assumed to be always deterministic. However, this assumption might be invalid in some cases. For example, the [**UUID()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_uuid) function is nondeterministic (and does not replicate). Be careful about using such functions in triggers.

Triggers can update tables, so error messages similar to those for stored functions occur with [**CREATE TRIGGER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-trigger) if you do not have the required privileges. On the replica side, the replica uses the trigger **DEFINER** attribute to determine which user is considered to be the creator of the trigger.

The rest of this section provides additional detail about the logging implementation and its implications. You need not read it unless you are interested in the background on the rationale for the current logging-related conditions on stored routine use. This discussion applies only for statement-based logging, and not for row-based logging, with the exception of the first item: **CREATE** and **DROP** statements are logged as statements regardless of the logging mode.

* The server writes [**CREATE EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-event), [**CREATE PROCEDURE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-procedure), [**CREATE FUNCTION**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-function), [**ALTER EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-event), [**ALTER PROCEDURE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-procedure), [**ALTER FUNCTION**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-function), [**DROP EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#drop-event), [**DROP PROCEDURE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#drop-procedure), and [**DROP FUNCTION**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#drop-function) statements to the binary log.
* A stored function invocation is logged as a [**SELECT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#select) statement if the function changes data and occurs within a statement that would not otherwise be logged. This prevents nonreplication of data changes that result from use of stored functions in nonlogged statements. For example, [**SELECT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#select) statements are not written to the binary log, but a [**SELECT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#select) might invoke a stored function that makes changes. To handle this, a **SELECT *func\_name*()** statement is written to the binary log when the given function makes a change. Suppose that the following statements are executed on the source server:
* CREATE FUNCTION f1(a INT) RETURNS INT
* BEGIN
* IF (a < 3) THEN
* INSERT INTO t2 VALUES (a);
* END IF;
* RETURN 0;
* END;
* CREATE TABLE t1 (a INT);
* INSERT INTO t1 VALUES (1),(2),(3);
* SELECT f1(a) FROM t1;

When the [**SELECT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#select) statement executes, the function **f1()** is invoked three times. Two of those invocations insert a row, and MySQL logs a [**SELECT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#select) statement for each of them. That is, MySQL writes the following statements to the binary log:

SELECT f1(1);

SELECT f1(2);

The server also logs a [**SELECT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#select) statement for a stored function invocation when the function invokes a stored procedure that causes an error. In this case, the server writes the [**SELECT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#select) statement to the log along with the expected error code. On the replica, if the same error occurs, that is the expected result and replication continues. Otherwise, replication stops.

* Logging stored function invocations rather than the statements executed by a function has a security implication for replication, which arises from two factors:
  + It is possible for a function to follow different execution paths on source and replica servers.
  + Statements executed on a replica are processed by the replica's applier thread. Unless you implement replication privilege checks, which are available from MySQL 8.0.18 (see [Section 17.3.3, “Replication Privilege Checks”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\replication.html#replication-privilege-checks)), the applier thread has full privileges.

The implication is that although a user must have the [**CREATE ROUTINE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_create-routine) privilege to create a function, the user can write a function containing a dangerous statement that executes only on the replica where it is processed by a thread that has full privileges. For example, if the source and replica servers have server ID values of 1 and 2, respectively, a user on the source server could create and invoke an unsafe function **unsafe\_func()** as follows:

mysql> **delimiter //**

mysql> **CREATE FUNCTION unsafe\_func () RETURNS INT**

-> **BEGIN**

-> **IF @@server\_id=2 THEN *dangerous\_statement*; END IF;**

-> **RETURN 1;**

-> **END;**

-> **//**

mysql> **delimiter ;**

mysql> **INSERT INTO t VALUES(unsafe\_func());**

The [**CREATE FUNCTION**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-function) and [**INSERT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#insert) statements are written to the binary log, so the replica executes them. Because the replica's applier thread has full privileges, it executes the dangerous statement. Thus, the function invocation has different effects on the source and replica and is not replication-safe.

To guard against this danger for servers that have binary logging enabled, stored function creators must have the [**SUPER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_super) privilege, in addition to the usual [**CREATE ROUTINE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_create-routine) privilege that is required. Similarly, to use [**ALTER FUNCTION**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-function), you must have the [**SUPER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_super) privilege in addition to the [**ALTER ROUTINE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_alter-routine) privilege. Without the [**SUPER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_super) privilege, an error occurs:

ERROR 1419 (HY000): You do not have the SUPER privilege and

binary logging is enabled (you \*might\* want to use the less safe

log\_bin\_trust\_function\_creators variable)

If you do not want to require function creators to have the [**SUPER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_super) privilege (for example, if all users with the [**CREATE ROUTINE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_create-routine) privilege on your system are experienced application developers), set the global **[log\_bin\_trust\_function\_creators](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\replication.html" \l "sysvar_log_bin_trust_function_creators)** system variable to 1. You can also set this variable at server startup. If binary logging is not enabled, **[log\_bin\_trust\_function\_creators](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\replication.html" \l "sysvar_log_bin_trust_function_creators)** does not apply. [**SUPER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_super) is not required for function creation unless, as described previously, the **DEFINER** value in the function definition requires it.

* The use of replication privilege checks where available (from MySQL 8.0.18) is recommended whatever choice you make about privileges for function creators. Replication privilege checks can be set up to ensure that only expected and relevant operations are authorized for the replication channel. For instructions to do this, see [Section 17.3.3, “Replication Privilege Checks”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\replication.html#replication-privilege-checks).
* If a function that performs updates is nondeterministic, it is not repeatable. This can have two undesirable effects:
  + It causes a replica to differ from the source.
  + Restored data does not match the original data.

To deal with these problems, MySQL enforces the following requirement: On a source server, creation and alteration of a function is refused unless you declare the function to be deterministic or to not modify data. Two sets of function characteristics apply here:

* + The **DETERMINISTIC** and **NOT DETERMINISTIC** characteristics indicate whether a function always produces the same result for given inputs. The default is **NOT DETERMINISTIC** if neither characteristic is given. To declare that a function is deterministic, you must specify **DETERMINISTIC** explicitly.
  + The **CONTAINS SQL**, **NO SQL**, **READS SQL DATA**, and **MODIFIES SQL DATA** characteristics provide information about whether the function reads or writes data. Either **NO SQL** or **READS SQL DATA** indicates that a function does not change data, but you must specify one of these explicitly because the default is **CONTAINS SQL** if no characteristic is given.

By default, for a [**CREATE FUNCTION**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-function) statement to be accepted, at least one of **DETERMINISTIC**, **NO SQL**, or **READS SQL DATA** must be specified explicitly. Otherwise an error occurs:

ERROR 1418 (HY000): This function has none of DETERMINISTIC, NO SQL,

or READS SQL DATA in its declaration and binary logging is enabled

(you \*might\* want to use the less safe log\_bin\_trust\_function\_creators

variable)

If you set **[log\_bin\_trust\_function\_creators](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\replication.html" \l "sysvar_log_bin_trust_function_creators)** to 1, the requirement that functions be deterministic or not modify data is dropped.

* Stored procedure calls are logged at the statement level rather than at the [**CALL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#call) level. That is, the server does not log the [**CALL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#call) statement, it logs those statements within the procedure that actually execute. As a result, the same changes that occur on the source server also occur on replicas. This prevents problems that could result from a procedure having different execution paths on different machines.

In general, statements executed within a stored procedure are written to the binary log using the same rules that would apply were the statements to be executed in standalone fashion. Some special care is taken when logging procedure statements because statement execution within procedures is not quite the same as in nonprocedure context:

* + A statement to be logged might contain references to local procedure variables. These variables do not exist outside of stored procedure context, so a statement that refers to such a variable cannot be logged literally. Instead, each reference to a local variable is replaced by this construct for logging purposes:
  + NAME\_CONST(***var\_name***, ***var\_value***)

***var\_name*** is the local variable name, and ***var\_value*** is a constant indicating the value that the variable has at the time the statement is logged. [**NAME\_CONST()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_name-const) has a value of ***var\_value***, and a “name” of ***var\_name***. Thus, if you invoke this function directly, you get a result like this:

mysql> **SELECT NAME\_CONST('myname', 14);**

+--------+

| myname |

+--------+

| 14 |

+--------+

[**NAME\_CONST()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_name-const) enables a logged standalone statement to be executed on a replica with the same effect as the original statement that was executed on the source within a stored procedure.

The use of [**NAME\_CONST()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_name-const) can result in a problem for [**CREATE TABLE ... SELECT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-table) statements when the source column expressions refer to local variables. Converting these references to [**NAME\_CONST()**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\functions.html#function_name-const) expressions can result in column names that are different on the source and replica servers, or names that are too long to be legal column identifiers. A workaround is to supply aliases for columns that refer to local variables. Consider this statement when **myvar** has a value of 1:

CREATE TABLE t1 SELECT myvar;

This is rewritten as follows:

CREATE TABLE t1 SELECT NAME\_CONST(myvar, 1);

To ensure that the source and replica tables have the same column names, write the statement like this:

CREATE TABLE t1 SELECT myvar AS myvar;

The rewritten statement becomes:

CREATE TABLE t1 SELECT NAME\_CONST(myvar, 1) AS myvar;

* + A statement to be logged might contain references to user-defined variables. To handle this, MySQL writes a [**SET**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#set-variable) statement to the binary log to make sure that the variable exists on the replica with the same value as on the source. For example, if a statement refers to a variable **@my\_var**, that statement is preceded in the binary log by the following statement, where ***value*** is the value of **@my\_var** on the source:
  + SET @my\_var = ***value***;
  + Procedure calls can occur within a committed or rolled-back transaction. Transactional context is accounted for so that the transactional aspects of procedure execution are replicated correctly. That is, the server logs those statements within the procedure that actually execute and modify data, and also logs [**BEGIN**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#commit), [**COMMIT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#commit), and [**ROLLBACK**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#commit) statements as necessary. For example, if a procedure updates only transactional tables and is executed within a transaction that is rolled back, those updates are not logged. If the procedure occurs within a committed transaction, [**BEGIN**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#commit) and [**COMMIT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#commit) statements are logged with the updates. For a procedure that executes within a rolled-back transaction, its statements are logged using the same rules that would apply if the statements were executed in standalone fashion:
    - Updates to transactional tables are not logged.
    - Updates to nontransactional tables are logged because rollback does not cancel them.
    - Updates to a mix of transactional and nontransactional tables are logged surrounded by [**BEGIN**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#commit) and [**ROLLBACK**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#commit) so that replicas make the same changes and rollbacks as on the source.
* A stored procedure call is not written to the binary log at the statement level if the procedure is invoked from within a stored function. In that case, the only thing logged is the statement that invokes the function (if it occurs within a statement that is logged) or a [**DO**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#do) statement (if it occurs within a statement that is not logged). For this reason, care should be exercised in the use of stored functions that invoke a procedure, even if the procedure is otherwise safe in itself.

## 25.8 Restrictions on Stored Programs

* [SQL Statements Not Permitted in Stored Routines](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\stored-objects.html" \l "stored-routine-sql-restrictions" \o "SQL Statements Not Permitted in Stored Routines)
* [Restrictions for Stored Functions](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-routines-function-restrictions)
* [Restrictions for Triggers](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-routines-trigger-restrictions)
* [Name Conflicts within Stored Routines](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-routine-name-conflicts)
* [Replication Considerations](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-routines-replication-restrictions)
* [Debugging Considerations](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-routines-debugging-restrictions)
* [Unsupported Syntax from the SQL:2003 Standard](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-routines-standard-restrictions)
* [Stored Routine Concurrency Considerations](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-routines-concurrency-restrictions)
* [Event Scheduler Restrictions](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-routines-event-restrictions)
* [Stored routines and triggers in NDB Cluster](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-routines-ndbcluster)

These restrictions apply to the features described in [Chapter 25, *Stored Objects*](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html).

Some of the restrictions noted here apply to all stored routines; that is, both to stored procedures and stored functions. There are also some [restrictions specific to stored functions](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-routines-function-restrictions) but not to stored procedures.

The restrictions for stored functions also apply to triggers. There are also some [restrictions specific to triggers](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-routines-trigger-restrictions).

The restrictions for stored procedures also apply to the [**DO**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#do) clause of Event Scheduler event definitions. There are also some [restrictions specific to events](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-routines-event-restrictions).

### SQL Statements Not Permitted in Stored Routines

Stored routines cannot contain arbitrary SQL statements. The following statements are not permitted:

* The locking statements [**LOCK TABLES**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#lock-tables) and [**UNLOCK TABLES**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#lock-tables).
* [**ALTER VIEW**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-view).
* [**LOAD DATA**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#load-data).
* SQL prepared statements ([**PREPARE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#prepare), [**EXECUTE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#execute), [**DEALLOCATE PREPARE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#deallocate-prepare)) can be used in stored procedures, but not stored functions or triggers. Thus, stored functions and triggers cannot use dynamic SQL (where you construct statements as strings and then execute them).
* Generally, statements not permitted in SQL prepared statements are also not permitted in stored programs. For a list of statements supported as prepared statements, see [Section 13.5, “Prepared Statements”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#sql-prepared-statements). Exceptions are [**SIGNAL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#signal), [**RESIGNAL**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#resignal), and [**GET DIAGNOSTICS**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#get-diagnostics), which are not permissible as prepared statements but are permitted in stored programs.
* Because local variables are in scope only during stored program execution, references to them are not permitted in prepared statements created within a stored program. Prepared statement scope is the current session, not the stored program, so the statement could be executed after the program ends, at which point the variables would no longer be in scope. For example, **SELECT ... INTO *local\_var*** cannot be used as a prepared statement. This restriction also applies to stored procedure and function parameters. See [Section 13.5.1, “PREPARE Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#prepare).
* Within all stored programs (stored procedures and functions, triggers, and events), the parser treats [**BEGIN [WORK]**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#commit) as the beginning of a [**BEGIN ... END**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#begin-end) block. To begin a transaction in this context, use [**START TRANSACTION**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#commit) instead.

### Restrictions for Stored Functions

The following additional statements or operations are not permitted within stored functions. They are permitted within stored procedures, except stored procedures that are invoked from within a stored function or trigger. For example, if you use [**FLUSH**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#flush) in a stored procedure, that stored procedure cannot be called from a stored function or trigger.

* Statements that perform explicit or implicit commit or rollback. Support for these statements is not required by the SQL standard, which states that each DBMS vendor may decide whether to permit them.
* Statements that return a result set. This includes [**SELECT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#select) statements that do not have an **INTO *var\_list*** clause and other statements such as [**SHOW**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show), [**EXPLAIN**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#explain), and [**CHECK TABLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#check-table). A function can process a result set either with [**SELECT ... INTO *var\_list***](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#select-into) or by using a cursor and [**FETCH**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#fetch) statements. See [Section 13.2.10.1, “SELECT ... INTO Statement”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#select-into), and [Section 13.6.6, “Cursors”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#cursors).
* [**FLUSH**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#flush) statements.
* Stored functions cannot be used recursively.
* A stored function or trigger cannot modify a table that is already being used (for reading or writing) by the statement that invoked the function or trigger.
* If you refer to a temporary table multiple times in a stored function under different aliases, a **Can't reopen table: '*tbl\_name*'** error occurs, even if the references occur in different statements within the function.
* [**HANDLER ... READ**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#handler) statements that invoke stored functions can cause replication errors and are disallowed.

### Restrictions for Triggers

For triggers, the following additional restrictions apply:

* Triggers are not activated by foreign key actions.
* When using row-based replication, triggers on the replica are not activated by statements originating on the source. The triggers on the replica are activated when using statement-based replication. For more information, see [Section 17.5.1.36, “Replication and Triggers”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\replication.html#replication-features-triggers).
* The [**RETURN**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#return) statement is not permitted in triggers, which cannot return a value. To exit a trigger immediately, use the [**LEAVE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#leave) statement.
* Triggers are not permitted on tables in the **mysql** database. Nor are they permitted on **INFORMATION\_SCHEMA** or **performance\_schema** tables. Those tables are actually views and triggers are not permitted on views.
* The trigger cache does not detect when metadata of the underlying objects has changed. If a trigger uses a table and the table has changed since the trigger was loaded into the cache, the trigger operates using the outdated metadata.

### Name Conflicts within Stored Routines

The same identifier might be used for a routine parameter, a local variable, and a table column. Also, the same local variable name can be used in nested blocks. For example:

CREATE PROCEDURE p (i INT)

BEGIN

DECLARE i INT DEFAULT 0;

SELECT i FROM t;

BEGIN

DECLARE i INT DEFAULT 1;

SELECT i FROM t;

END;

END;

In such cases, the identifier is ambiguous and the following precedence rules apply:

* A local variable takes precedence over a routine parameter or table column.
* A routine parameter takes precedence over a table column.
* A local variable in an inner block takes precedence over a local variable in an outer block.

The behavior that variables take precedence over table columns is nonstandard.

### Replication Considerations

Use of stored routines can cause replication problems. This issue is discussed further in [Section 25.7, “Stored Program Binary Logging”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-programs-logging).

The [--replicate-wild-do-table=***db\_name.tbl\_name***](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\replication.html#option_mysqld_replicate-wild-do-table) option applies to tables, views, and triggers. It does not apply to stored procedures and functions, or events. To filter statements operating on the latter objects, use one or more of the --replicate-\*-db options.

### Debugging Considerations

There are no stored routine debugging facilities.

### Unsupported Syntax from the SQL:2003 Standard

The MySQL stored routine syntax is based on the SQL:2003 standard. The following items from that standard are not currently supported:

* **UNDO** handlers
* **FOR** loops

### Stored Routine Concurrency Considerations

To prevent problems of interaction between sessions, when a client issues a statement, the server uses a snapshot of routines and triggers available for execution of the statement. That is, the server calculates a list of procedures, functions, and triggers that may be used during execution of the statement, loads them, and then proceeds to execute the statement. While the statement executes, it does not see changes to routines performed by other sessions.

For maximum concurrency, stored functions should minimize their side-effects; in particular, updating a table within a stored function can reduce concurrent operations on that table. A stored function acquires table locks before executing, to avoid inconsistency in the binary log due to mismatch of the order in which statements execute and when they appear in the log. When statement-based binary logging is used, statements that invoke a function are recorded rather than the statements executed within the function. Consequently, stored functions that update the same underlying tables do not execute in parallel. In contrast, stored procedures do not acquire table-level locks. All statements executed within stored procedures are written to the binary log, even for statement-based binary logging. See [Section 25.7, “Stored Program Binary Logging”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#stored-programs-logging).

### Event Scheduler Restrictions

The following limitations are specific to the Event Scheduler:

* Event names are handled in case-insensitive fashion. For example, you cannot have two events in the same database with the names **anEvent** and **AnEvent**.
* An event may not be created, altered, or dropped from within a stored program, if the event name is specified by means of a variable. An event also may not create, alter, or drop stored routines or triggers.
* DDL statements on events are prohibited while a [**LOCK TABLES**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#lock-tables) statement is in effect.
* Event timings using the intervals **YEAR**, **QUARTER**, **MONTH**, and **YEAR\_MONTH** are resolved in months; those using any other interval are resolved in seconds. There is no way to cause events scheduled to occur at the same second to execute in a given order. In addition—due to rounding, the nature of threaded applications, and the fact that a nonzero length of time is required to create events and to signal their execution—events may be delayed by as much as 1 or 2 seconds. However, the time shown in the [**INFORMATION\_SCHEMA.EVENTS**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\information-schema.html#information-schema-events-table) table's **LAST\_EXECUTED** column is always accurate to within one second of the actual event execution time. (See also Bug #16522.)
* Each execution of the statements contained in the body of an event takes place in a new connection; thus, these statements have no effect in a given user session on the server's statement counts such as **Com\_select** and **Com\_insert** that are displayed by [**SHOW STATUS**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-status). However, such counts are updated in the global scope. (Bug #16422)
* Events do not support times later than the end of the Unix Epoch; this is approximately the beginning of the year 2038. Such dates are specifically not permitted by the Event Scheduler. (Bug #16396)
* References to stored functions, user-defined functions, and tables in the **ON SCHEDULE** clauses of [**CREATE EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-event) and [**ALTER EVENT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-event) statements are not supported. These sorts of references are not permitted. (See Bug #22830 for more information.)

### Stored routines and triggers in NDB Cluster

While stored procedures, stored functions, triggers, and scheduled events are all supported by tables using the [**NDB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\mysql-cluster.html) storage engine, you must keep in mind that these do not propagate automatically between MySQL Servers acting as Cluster SQL nodes. This is because stored routine and trigger definitions are stored in tables in the **mysql** system database using **InnoDB** tables, which are not copied between Cluster nodes.

Any stored routine or trigger that interacts with MySQL Cluster tables must be re-created by running the appropriate [**CREATE PROCEDURE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-procedure), [**CREATE FUNCTION**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-function), or [**CREATE TRIGGER**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-trigger) statements on each MySQL Server that participates in the cluster where you wish to use the stored routine or trigger. Similarly, any changes to existing stored routines or triggers must be carried out explicitly on all Cluster SQL nodes, using the appropriate **ALTER** or **DROP** statements on each MySQL Server accessing the cluster.

**Warning**

Do not attempt to work around the issue just described by converting any **mysql** database tables to use the [**NDB**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\mysql-cluster.html) storage engine. Altering the system tables in the ***mysql*** database is not supported and is very likely to produce undesirable results.

## 25.9 Restrictions on Views

The maximum number of tables that can be referenced in the definition of a view is 61.

View processing is not optimized:

* It is not possible to create an index on a view.
* Indexes can be used for views processed using the merge algorithm. However, a view that is processed with the temptable algorithm is unable to take advantage of indexes on its underlying tables (although indexes can be used during generation of the temporary tables).

There is a general principle that you cannot modify a table and select from the same table in a subquery. See [Section 13.2.11.12, “Restrictions on Subqueries”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#subquery-restrictions).

The same principle also applies if you select from a view that selects from the table, if the view selects from the table in a subquery and the view is evaluated using the merge algorithm. Example:

CREATE VIEW v1 AS

SELECT \* FROM t2 WHERE EXISTS (SELECT 1 FROM t1 WHERE t1.a = t2.a);

UPDATE t1, v2 SET t1.a = 1 WHERE t1.b = v2.b;

If the view is evaluated using a temporary table, you can select from the table in the view subquery and still modify that table in the outer query. In this case, the view is stored in a temporary table and thus you are not really selecting from the table in a subquery and modifying it at the same time. (This is another reason you might wish to force MySQL to use the temptable algorithm by specifying **ALGORITHM = TEMPTABLE** in the view definition.)

You can use [**DROP TABLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#drop-table) or [**ALTER TABLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#alter-table) to drop or alter a table that is used in a view definition. No warning results from the **DROP** or **ALTER** operation, even though this invalidates the view. Instead, an error occurs later, when the view is used. [**CHECK TABLE**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#check-table) can be used to check for views that have been invalidated by **DROP** or **ALTER** operations.

With regard to view updatability, the overall goal for views is that if any view is theoretically updatable, it should be updatable in practice. MySQL as quickly as possible. Many theoretically updatable views can be updated now, but limitations still exist. For details, see [Section 25.5.3, “Updatable and Insertable Views”](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\stored-objects.html#view-updatability).

There exists a shortcoming with the current implementation of views. If a user is granted the basic privileges necessary to create a view (the [**CREATE VIEW**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_create-view) and [**SELECT**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_select) privileges), that user cannot call [**SHOW CREATE VIEW**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-create-view) on that object unless the user is also granted the [**SHOW VIEW**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_show-view) privilege.

That shortcoming can lead to problems backing up a database with **[mysqldump](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\programs.html" \l "mysqldump" \o "4.5.4 mysqldump — A Database Backup Program)**, which may fail due to insufficient privileges. This problem is described in Bug #22062.

The workaround to the problem is for the administrator to manually grant the [**SHOW VIEW**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_show-view) privilege to users who are granted [**CREATE VIEW**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\security.html#priv_create-view), since MySQL doesn't grant it implicitly when views are created.

Views do not have indexes, so index hints do not apply. Use of index hints when selecting from a view is not permitted.

[**SHOW CREATE VIEW**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-create-view) displays view definitions using an **AS *alias\_name*** clause for each column. If a column is created from an expression, the default alias is the expression text, which can be quite long. Aliases for column names in [**CREATE VIEW**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-view) statements are checked against the maximum column length of 64 characters (not the maximum alias length of 256 characters). As a result, views created from the output of [**SHOW CREATE VIEW**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#show-create-view) fail if any column alias exceeds 64 characters. This can cause problems in the following circumstances for views with too-long aliases:

* View definitions fail to replicate to newer replicas that enforce the column-length restriction.
* Dump files created with **[mysqldump](file:///E:\\backup\\%E4%B8%8B%E8%BD%BD\\refman-8.0-en.html-chapter\\refman-8.0-en.html-chapter\\programs.html" \l "mysqldump" \o "4.5.4 mysqldump — A Database Backup Program)** cannot be loaded into servers that enforce the column-length restriction.

A workaround for either problem is to modify each problematic view definition to use aliases that provide shorter column names. Then the view replicates properly, and can be dumped and reloaded without causing an error. To modify the definition, drop and create the view again with [**DROP VIEW**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#drop-view) and [**CREATE VIEW**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-view), or replace the definition with [**CREATE OR REPLACE VIEW**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-view).

For problems that occur when reloading view definitions in dump files, another workaround is to edit the dump file to modify its [**CREATE VIEW**](file:///E:\backup\%E4%B8%8B%E8%BD%BD\refman-8.0-en.html-chapter\refman-8.0-en.html-chapter\sql-statements.html#create-view) statements. However, this does not change the original view definitions, which may cause problems for subsequent dump operations.