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

Query OK, 0 rows affected (0.00 sec)

mysql> **SHOW** **FUNCTION** **CODE** **test** **.hello;**

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

| Pos | Instruction |

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

| 0 | freturn 254 concat('Hello, ',s@0,'!') |

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

1 row in set (0.00 sec)

...

Privileges belonging to a specific user are displayed by the SHOW GRANTS statement. See Section 13.7.7.21, “SHOW GRANTS Statement” , for more information.

**13.7.7.27** **SHOW** **PROCEDURE** **CODE** **Statement**

SHOW PROCEDURE CODE *proc\_name*

This statement is a MySQL extension that is available only for servers that have been built with debugging support. It displays a representation of the internal implementation of the named stored procedure. A similar statement, SHOW FUNCTION CODE, displays information about stored functions (see Section 13.7.7.19, “SHOW FUNCTION CODE Statement” ).

To use either statement, you must be the user named as the routine DEFINER, have the SHOW\_ROUTINE privilege, or have the SELECT privilege at the global level.

If the named routine is available, each statement produces a result set. Each row in the result set corresponds to one “instruction” in the routine. The first column is Pos, which is an ordinal number beginning with 0. The second column is Instruction, which contains an SQL statement (usually changed from the original source), or a directive which has meaning only to the stored-routine handler.

mysql> **DELIMITER** **//**

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

**BEGIN**

**DECLARE** **fanta** **INT** **DEFAULT** **55;**

**DROP** **TABLE** **t2;**

**LOOP**

**INSERT** **INTO** **t3** **VALUES** **(fanta);**

**END** **LOOP;**

**END//**

Query OK, 0 rows affected (0.01 sec)

mysql> **SHOW** **PROCEDURE** **CODE** **p1//**

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

| Pos | Instruction |

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

| 0 | set fanta@0 55 |

| 1 | stmt 9 "DROP TABLE t2" |

| 2 | stmt 5 "INSERT INTO t3 VALUES (fanta)" |

| 3 | jump 2 |

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

4 rows in set (0.00 sec)

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

**RETURNS** **CHAR(50)** **DETERMINISTIC**

In this example, the nonexecutable BEGIN and END statements have disappeared, and for the DECLARE *variable\_name* statement, only the executable part appears (the part where the default is assigned). For each statement that is taken from source, there is a code word stmt followed by a type (9 means DROP, 5 means INSERT, and so on). The final row contains an instruction jump 2, meaning GOTO instruction #2.

**13.7.7.28** **SHOW** **PROCEDURE** **STATUS** **Statement**

SHOW PROCEDURE STATUS

[LIKE '*pattern*' | WHERE *expr*]



This statement is a MySQL extension. It returns characteristics of a stored procedure, such as the database, name, type, creator, creation and modification dates, and character set information. A similar statement, SHOW FUNCTION STATUS, displays information about stored functions (see Section 13.7.7.20, “SHOW FUNCTION STATUS Statement” ).

To use either statement, you must be the user named as the routine DEFINER, have the SHOW\_ROUTINE privilege, have the SELECT privilege at the global level, or have the CREATE ROUTINE, ALTER ROUTINE, or EXECUTE privilege granted at a scope that includes the routine.

The LIKE clause, if present, indicates which procedure or function names to match. The WHERE clause can be given to select rows using more general conditions, as discussed in Section 26.8, “Extensions to SHOW Statements” .

mysql> **SHOW** **PROCEDURE** **STATUS** **LIKE** **'sp1'\G**

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

Db: test

Name: sp1

Type: PROCEDURE

Definer: testuser@localhost

Modified: 2018-08-08 13:54:11

Created: 2018-08-08 13:54:11

Security\_type: DEFINER

Comment:

character\_set\_client: utf8mb4

collation\_connection: utf8mb4\_0900\_ai\_ci

Database Collation: utf8mb4\_0900\_ai\_ci

mysql> **SHOW** **FUNCTION** **STATUS** **LIKE** **'hello'\G**

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

Db: test

Name: hello

Type: FUNCTION

Definer: testuser@localhost

Modified: 2020-03-10 11:10:03

Created: 2020-03-10 11:10:03

Security\_type: DEFINER

Comment:

character\_set\_client: utf8mb4

collation\_connection: utf8mb4\_0900\_ai\_ci

Database Collation: utf8mb4\_0900\_ai\_ci

character\_set\_client is the session value of the character\_set\_client system

variable when the routine was created. collation\_connection is the session value of the collation\_connection system variable when the routine was created. Database Collation is the collation of the database with which the routine is associated.

Stored routine information is also available from the INFORMATION\_SCHEMA PARAMETERS and ROUTINES tables. See Section 26.3.20, “The INFORMATION\_SCHEMA PARAMETERS Table” , and Section 26.3.30, “The INFORMATION\_SCHEMA ROUTINES Table” .

**13.7.7.29** **SHOW** **PROCESSLIST** **Statement**

SHOW [FULL] PROCESSLIST

The MySQL process list indicates the operations currently being performed by the set of threads executing within the server. The [SHOW PROCESSLIST](#_bookmark1) statement is one source of process information. For a comparison of this statement with other sources, see Sources of Process Information.

**Note**

As of MySQL 8.0.22, an alternative implementation for [SHOW PROCESSLIST](#_bookmark1) is available based on the Performance Schema processlist table, which, unlike the default [SHOW PROCESSLIST](#_bookmark1) implementation, does not require a mutex and has better performance characteristics. For details, see Section 27.12.21.6, “The processlist Table” .

If you have the PROCESS privilege, you can see all threads, even those belonging to other users. Otherwise (without the PROCESS privilege), nonanonymous users have access to information about their own threads but not threads for other users, and anonymous users have no access to thread information.

Without the FULL keyword, [SHOW PROCESSLIST](#_bookmark1) displays only the first 100 characters of each statement in the Info field.

The [SHOW PROCESSLIST](#_bookmark1) statement is very useful if you get the “too many connections” error message and want to find out what is going on. MySQL reserves one extra connection to be used by accounts that have the CONNECTION\_ADMIN privilege (or the deprecated SUPER privilege), to ensure that administrators should always be able to connect and check the system (assuming that you are not giving this privilege to all your users).

Threads can be killed with the [KILL](#_bookmark2) statement. See [Section 13.7.8.4, “KILL Statement”](#_bookmark2) . Example of [SHOW PROCESSLIST](#_bookmark1) output:

mysql> SHOW FULL PROCESSLIST\G

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

Id: 1

User: system user

Host:

db: NULL

Command: Connect

Time: 1030455

State: Waiting for master to send event

Info: NULL

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

Id: 2

User: system user

Host:

db: NULL

Command: Connect

Time: 1004

State: Has read all relay log; waiting for the slave

I/O thread to update it

Info: NULL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 3. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Id: 3112

User: replikator

Host: artemis:2204

db: NULL

Command: Binlog Dump

Time: 2144

State: Has sent all binlog to slave; waiting for binlog to be updated

Info: NULL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 4. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Id: 3113

User: replikator

Host: iconnect2:45781

db: NULL

Command: Binlog Dump

Time: 2086

State: Has sent all binlog to slave; waiting for binlog to be updated

Info: NULL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 5. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Id: 3123

User: stefan

Host: localhost

db: apollon

Command: Query

Time: 0

State: NULL

Info: SHOW FULL PROCESSLIST

[SHOW PROCESSLIST](#_bookmark1) output has these columns:

• Id



The connection identifier. This is the same value displayed in the ID column of the INFORMATION\_SCHEMA PROCESSLIST table, displayed in the PROCESSLIST\_ID column of the Performance Schema threads table, and returned by the CONNECTION\_ID() function within the thread.

• User

The MySQL user who issued the statement. A value of system user refers to a nonclient thread spawned by the server to handle tasks internally, for example, a delayed-row handler thread or an I/O (receiver) or SQL (applier) thread used on replica hosts. For system user, there is no

host specified in the Host column. unauthenticated user refers to a thread that has become associated with a client connection but for which authentication of the client user has not yet occurred. event\_scheduler refers to the thread that monitors scheduled events (see Section 25.4, “Using the Event Scheduler” ).

**Note**

A User value of system user is distinct from the SYSTEM\_USER privilege. The former designates internal threads. The latter distinguishes the system user and regular user account categories (see Section 6.2.11, “Account Categories”).

• Host

The host name of the client issuing the statement (except for system user, for which there is no host). The host name for TCP/IP connections is reported in *host\_name*:*client\_port* format to make it easier to determine which client is doing what.

• db

The default database for the thread, or NULL if none has been selected.

• Command

The type of command the thread is executing on behalf of the client, or Sleep if the session is idle. For descriptions of thread commands, see Section 8.14, “Examining Server Thread (Process) Information” . The value of this column corresponds to the COM\_*xxx* commands of the client/server protocol and Com\_*xxx* status variables. See Section 5.1.10, “Server Status Variables” .

• Time

The time in seconds that the thread has been in its current state. For a replica SQL thread, the value is the number of seconds between the timestamp of the last replicated event and the real time of the replica host. See Section 17.2.3, “Replication Threads” .

• State

An action, event, or state that indicates what the thread is doing. For descriptions of State values, see Section 8.14, “Examining Server Thread (Process) Information” .

Most states correspond to very quick operations. If a thread stays in a given state for many seconds, there might be a problem that needs to be investigated.

• Info

The statement the thread is executing, or NULL if it is executing no statement. The statement might be the one sent to the server, or an innermost statement if the statement executes other statements. For example, if a CALL statement executes a stored procedure that is executing a SELECT statement, the Info value shows the SELECT statement.

**13.7.7.30** **SHOW** **PROFILE** **Statement**



SHOW PROFILE [*type* [, *type*] ... ]

[FOR QUERY *n*]

[LIMIT *row\_count* [OFFSET *offset*]]

*type*: {

ALL

| BLOCK IO

| CONTEXT SWITCHES

| CPU

| IPC

| MEMORY

| PAGE FAULTS

| SOURCE

| SWAPS

}

The [SHOW PROFILE](#_bookmark3) and [SHOW PROFILES](#_bookmark4) statements display profiling information that indicates resource usage for statements executed during the course of the current session.

**Note**

The [SHOW PROFILE](#_bookmark3) and [SHOW PROFILES](#_bookmark4) statements are deprecated; expect them to be removed in a future MySQL release. Use the Performance Schema instead; see Section 27.19.1, “Query Profiling Using Performance Schema” .

To control profiling, use the profiling session variable, which has a default value of 0 (OFF). Enable profiling by setting profiling to 1 or ON:

mysql> **SET** **profiling** **=** **1;**

[SHOW PROFILES](#_bookmark4) displays a list of the most recent statements sent to the server. The size of the list is controlled by the profiling\_history\_size session variable, which has a default value of 15. The maximum value is 100. Setting the value to 0 has the practical effect of disabling profiling.

All statements are profiled except [SHOW PROFILE](#_bookmark3) and [SHOW PROFILES](#_bookmark4), so neither of those statements appears in the profile list. Malformed statements are profiled. For example, SHOW PROFILING is an illegal statement, and a syntax error occurs if you try to execute it, but it shows up in the profiling list.

[SHOW PROFILE](#_bookmark3) displays detailed information about a single statement. Without the FOR QUERY *n* clause, the output pertains to the most recently executed statement. If FOR QUERY *n* is included, [SHOW](#_bookmark3) [PROFILE](#_bookmark3) displays information for statement *n*. The values of *n* correspond to the Query\_ID values displayed by [SHOW PROFILES](#_bookmark4).

The LIMIT *row\_count* clause may be given to limit the output to *row\_count* rows. If LIMIT is given, OFFSET *offset* may be added to begin the output *offset* rows into the full set of rows.

By default, [SHOW PROFILE](#_bookmark3) displays Status and Duration columns. The Status values are like the State values displayed by [SHOW PROCESSLIST](#_bookmark1), although there might be some minor differences in interpretation for the two statements for some status values (see Section 8.14, “Examining Server Thread (Process) Information” ).

Optional *type* values may be specified to display specific additional types of information:

• ALL displays all information

• BLOCK IO displays counts for block input and output operations

• CONTEXT SWITCHES displays counts for voluntary and involuntary context switches

• CPU displays user and system CPU usage times

• IPC displays counts for messages sent and received

• MEMORY is not currently implemented

• PAGE FAULTS displays counts for major and minor page faults

• SOURCE displays the names of functions from the source code, together with the name and line number of the file in which the function occurs

• SWAPS displays swap counts

Profiling is enabled per session. When a session ends, its profiling information is lost.

mysql> **SELECT** **@@profiling;**

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

| @@profiling |

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

| 0 |

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

1 row in set (0.00 sec)

mysql> **SET** **profiling** **=** **1;**

Query OK, 0 rows affected (0.00 sec)

mysql> **DROP** **TABLE** **IF** **EXISTS** **t1;**

Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> **CREATE** **TABLE** **T1** **(id** **INT);**

Query OK, 0 rows affected (0.01 sec)

mysql> **SHOW** **PROFILES;**

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

| Query\_ID | Duration | Query |

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  | | 0  1  2 | |  |  | | 0.000088  0.000136  0.011947 | | SET PROFILING = 1 |  | DROP TABLE IF EXISTS t1 |  | CREATE TABLE t1 (id INT) | |

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

3 rows in set (0.00 sec)

mysql> **SHOW** **PROFILE;**

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

| Status | Duration |

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

| checking permissions | 0.000040 |

|  |  |  |  |
| --- | --- | --- | --- |
| | creating table  | After create  | query end  | freeing items  | logging slow query  | cleaning up | |  |  |  |  |  | | 0.000056  0.011363  0.000375  0.000089  0.000019  0.000005 | |  |  |  |  |  | |

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

7 rows in set (0.00 sec)

mysql> **SHOW** **PROFILE** **FOR** **QUERY** **1;**

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

| Status | Duration |

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

| query end | 0.000107 |

| freeing items | 0.000008 |

| logging slow query | 0.000015 |

| cleaning up | 0.000006 |

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

4 rows in set (0.00 sec)

mysql> **SHOW** **PROFILE** **CPU** **FOR** **QUERY** **2;**

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

| Status | Duration | CPU\_user | CPU\_system |

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

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | | checking permissions | | | 0.000040 | | | 0.000038 | | | 0.000002 | | |
| | | creating table | | | 0.000056 | | | 0.000028 | | | 0.000028 | | |
| | | After create | | | 0.011363 | | | 0.000217 | | | 0.001571 | | |
| | | query end | | | 0.000375 | | | 0.000013 | | | 0.000028 | | |
| | | freeing items | | | 0.000089 | | | 0.000010 | | | 0.000014 | | |
| | | logging slow query | | | 0.000019 | | | 0.000009 | | | 0.000010 | | |
| | | cleaning up | | | 0.000005 | | | 0.000003 | | | 0.000002 | | |



time- and resource-consuming process because the server returns to the client the complete contents of the relay log (including all statements modifying data that have been received by the replica).

The optional FOR CHANNEL *channel* clause enables you to name which replication channel the statement applies to. Providing a FOR CHANNEL *channel* clause applies the statement to a specific replication channel. If no channel is named and no extra channels exist, the statement applies to the default channel.

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

7 rows in set (0.00 sec)

**Note**

Profiling is only partially functional on some architectures. For values that depend on the getrusage() system call, NULL is returned on systems such as Windows that do not support the call. In addition, profiling is per process and not per thread. This means that activity on threads within the server other than your own may affect the timing information that you see.

Profiling information is also available from the INFORMATION\_SCHEMA PROFILING table. See Section 26.3.24, “The INFORMATION\_SCHEMA PROFILING Table” . For example, the following queries are equivalent:

SHOW PROFILE FOR QUERY 2;

SELECT STATE, FORMAT(DURATION, 6) AS DURATION

FROM INFORMATION\_SCHEMA.PROFILING

WHERE QUERY\_ID = 2 ORDER BY SEQ;

**13.7.7.31** **SHOW** **PROFILES** **Statement**

SHOW PROFILES

The [SHOW PROFILES](#_bookmark4) statement, together with [SHOW PROFILE](#_bookmark3), displays profiling information that indicates resource usage for statements executed during the course of the current session. For more information, see [Section 13.7.7.30, “SHOW PROFILE Statement”](#_bookmark3) .

**Note**

The [SHOW PROFILE](#_bookmark3) and [SHOW PROFILES](#_bookmark4) statements are deprecated; expect it to be removed in a future MySQL release. Use the Performance Schema instead; see Section 27.19.1, “Query Profiling Using Performance Schema” .

**13.7.7.32** **SHOW** **RELAYLOG** **EVENTS** **Statement**

SHOW RELAYLOG EVENTS

[IN '*log\_name* ']

[FROM *pos*]

[LIMIT [*offset*,] *row\_count*]

[*channel\_option*]

*channel\_option*:

FOR CHANNEL *channel*

Shows the events in the relay log of a replica. If you do not specify '*log\_name*', the first relay log is displayed. This statement has no effect on the source. SHOW RELAYLOG EVENTS requires the REPLICATION SLAVE privilege.

The LIMIT clause has the same syntax as for the SELECT statement. See Section 13.2.13, “SELECT

Statement” .

 **Note**

 Issuing a [SHOW RELAYLOG EVENTS](#_bookmark5) with no LIMIT clause could start a very



When using multiple replication channels, if a [SHOW RELAYLOG EVENTS](#_bookmark5) statement does not have a channel defined using a FOR CHANNEL *channel* clause an error is generated. See Section 17.2.2, “Replication Channels” for more information.

[SHOW RELAYLOG EVENTS](#_bookmark5) displays the following fields for each event in the relay log:

• Log\_name

The name of the file that is being listed.

• Pos

The position at which the event occurs.

• Event\_type

An identifier that describes the event type.

• Server\_id

The server ID of the server on which the event originated.

• End\_log\_pos

The value of End\_log\_pos for this event in the source's binary log.

• Info

More detailed information about the event type. The format of this information depends on the event type.

For compressed transaction payloads, the Transaction\_payload\_event is first printed as a single unit, then it is unpacked and each event inside it is printed.

Some events relating to the setting of user and system variables are not included in the output from [SHOW RELAYLOG EVENTS](#_bookmark5). To get complete coverage of events within a relay log, use mysqlbinlog.

**13.7.7.33** **SHOW** **REPLICAS** **Statement**

{SHOW REPLICAS}

Displays a list of replicas currently registered with the source. From MySQL 8.0.22, use [SHOW](#_bookmark6) [REPLICAS](#_bookmark6) in place of [SHOW SLAVE HOSTS](#_bookmark7), which is deprecated from that release. In releases before MySQL 8.0.22, use [SHOW SLAVE HOSTS](#_bookmark7). SHOW REPLICAS requires the REPLICATION SLAVE privilege.

SHOW REPLICAS should be executed on a server that acts as a replication source. The statement displays information about servers that are or have been connected as replicas, with each row of the result corresponding to one replica server, as shown here:

mysql> **SHOW** **REPLICAS**;

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

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| | Server\_id | Host | | | Port | | | Source\_id | | | Replica\_UUID | | |

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

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | | 10 | | | iconnect2 | | | 3306 | | | 3 | | | 14cb6624-7f93-11e0-b2c0-c80aa9429562 | | |
| | | 21 | | | athena | | | 3306 | | | 3 | | | 07af4990-f41f-11df-a566-7ac56fdaf645 | | |

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

• Server\_id: The unique server ID of the replica server, as configured in the replica server's option file, or on the command line with --server-id=*value*.

• Host: The host name of the replica server, as specified on the replica with the --report-host option. This can differ from the machine name as configured in the operating system.

• User: The replica server user name, as specified on the replica with the --report-user option. Statement output includes this column only if the source server is started with the --show- replica-auth-info or --show-slave-auth-info option.

• Password: The replica server password, as specified on the replica with the --report-password option. Statement output includes this column only if the source server is started with the --show- replica-auth-info or --show-slave-auth-info option.

• Port: The port on the source to which the replica server is listening, as specified on the replica with the --report-port option.

A zero in this column means that the replica port (--report-port) was not set.

• Source\_id: The unique server ID of the source server that the replica server is replicating from. This is the server ID of the server on which SHOW REPLICAS is executed, so this same value is listed for each row in the result.

• Replica\_UUID: The globally unique ID of this replica, as generated on the replica and found in the replica's auto.cnf file.

**13.7.7.34** **SHOW** **SLAVE** **HOSTS** **|** **SHOW** **REPLICAS** **Statement**

{SHOW SLAVE HOSTS | SHOW REPLICAS}

Displays a list of replicas currently registered with the source. From MySQL 8.0.22, [SHOW SLAVE](#_bookmark7) [HOSTS](#_bookmark7) is deprecated and the alias [SHOW REPLICAS](#_bookmark6) should be used instead. The statement works in the same way as before, only the terminology used for the statement and its output has changed. Both versions of the statement update the same status variables when used. Please see the documentation for [SHOW REPLICAS](#_bookmark6) for a description of the statement.

**13.7.7.35** **SHOW** **REPLICA** **STATUS** **Statement**

SHOW {REPLICA | SLAVE} STATUS [FOR CHANNEL *channel*]

This statement provides status information on essential parameters of the replica threads. From MySQL 8.0.22, use [SHOW REPLICA STATUS](#_bookmark8) in place of [SHOW SLAVE STATUS](#_bookmark9), which is deprecated from that release. In releases before MySQL 8.0.22, use [SHOW SLAVE STATUS](#_bookmark9). The statement requires the REPLICATION CLIENT privilege (or the deprecated SUPER privilege).

SHOW REPLICA STATUS is nonblocking. When run concurrently with STOP REPLICA, SHOW REPLICA STATUS returns without waiting for STOP REPLICA to finish shutting down the replication SQL (applier) thread or replication I/O (receiver) thread (or both). This permits use in monitoring and other applications where getting an immediate response from SHOW REPLICA STATUS is more important than ensuring that it returned the latest data. The SLAVE keyword was replaced with REPLICA in MySQL 8.0.22.

If you issue this statement using the mysql client, you can use a \G statement terminator rather than a semicolon to obtain a more readable vertical layout:

mysql> **SHOW** **REPLICA** **STATUS\G**

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

Replica\_IO\_State: Waiting for source to send event

Source\_Host: localhost

Source\_User: repl

Source\_Port: 13000

Connect\_Retry: 60

Source\_Log\_File: source-bin.000002

Read\_Source\_Log\_Pos: 1307

Relay\_Log\_File: replica-relay-bin.000003

Relay\_Log\_Pos: 1508

Relay\_Source\_Log\_File: source-bin.000002

Replica\_IO\_Running: Yes

Replica\_SQL\_Running: Yes

Replicate\_Do\_DB:

Replicate\_Ignore\_DB:

Replicate\_Do\_Table:

Replicate\_Ignore\_Table:

Replicate\_Wild\_Do\_Table:

Replicate\_Wild\_Ignore\_Table:

Last\_Errno: 0

Last\_Error:

Skip\_Counter: 0

Exec\_Source\_Log\_Pos: 1307

Relay\_Log\_Space: 1858

Until\_Condition: None

Until\_Log\_File:

Until\_Log\_Pos: 0

Source\_SSL\_Allowed: No

Source\_SSL\_CA\_File:

Source\_SSL\_CA\_Path:

Source\_SSL\_Cert:

Source\_SSL\_Cipher:

Source\_SSL\_Key:

Seconds\_Behind\_Source: 0

Source\_SSL\_Verify\_Server\_Cert: No

Last\_IO\_Errno: 0

Last\_IO\_Error:

Last\_SQL\_Errno: 0

Last\_SQL\_Error:

Replicate\_Ignore\_Server\_Ids:

Source\_Server\_Id: 1

Source\_UUID: 3e11fa47-71ca-11e1-9e33-c80aa9429562

Source\_Info\_File:

SQL\_Delay: 0

SQL\_Remaining\_Delay: NULL

Replica\_SQL\_Running\_State: Reading event from the relay log

Source\_Retry\_Count: 10

Source\_Bind:

Last\_IO\_Error\_Timestamp:

Last\_SQL\_Error\_Timestamp:

Source\_SSL\_Crl:

Source\_SSL\_Crlpath:

Retrieved\_Gtid\_Set: 3e11fa47-71ca-11e1-9e33-c80aa9429562:1-5

Executed\_Gtid\_Set: 3e11fa47-71ca-11e1-9e33-c80aa9429562:1-5

Auto\_Position: 1

Replicate\_Rewrite\_DB:

Channel\_name:

Source\_TLS\_Version: TLSv1.2

Source\_public\_key\_path: public\_key.pem

Get\_source\_public\_key: 0

Network\_Namespace:

The Performance Schema provides tables that expose replication information. This is similar to the information available from the [SHOW REPLICA STATUS](#_bookmark8) statement, but represented in table form. For details, see Section 27.12.11, “Performance Schema Replication Tables” .

From MySQL 8.0.27, you can set the GTID\_ONLY option on the CHANGE REPLICATION SOURCE TO statement to stop a replication channel from persisting file names and file positions in the replication metadata repositories. With this setting, file positions for the source binary log file and the relay log file are tracked in memory. The [SHOW REPLICA STATUS](#_bookmark8) statement still displays file positions in normal use. However, because the file positions are not being regularly updated in the connection metadata repository and the applier metadata repository except in a few situations, they are likely to be out of date if the server is restarted.

For a replication channel with the GTID\_ONLY setting after a server start, the read and applied file positions for the source binary log file (Read\_Source\_Log\_Pos and Exec\_Source\_Log\_Pos) are set to zero, and the file names (Source\_Log\_File and Relay\_Source\_Log\_File) are set to INVALID. The relay log file name (Relay\_Log\_File) is set according to the relay\_log\_recovery setting, either a new file that was created at server start or the first relay log file present. The file position (Relay\_Log\_Pos) is set to position 4, and GTID auto-skip is used to skip any transactions in the file that were already applied.

When the receiver thread contacts the source and gets valid position information, the read position (Read\_Source\_Log\_Pos) and file name (Source\_Log\_File) are updated with the correct

data and become valid. When the applier thread applies a transaction from the source, or skips an already executed transaction, the executed position (Exec\_Source\_Log\_Pos) and file name (Relay\_Source\_Log\_File) are updated with the correct data and become valid. The relay log file position (Relay\_Log\_Pos) is also updated at that time.

The following list describes the fields returned by [SHOW REPLICA STATUS](#_bookmark8). For additional information about interpreting their meanings, see Section 17.1.7.1, “Checking Replication Status” .

• Replica\_IO\_State

A copy of the State field of the [SHOW PROCESSLIST](#_bookmark1) output for the replica I/O (receiver) thread. This tells you what the thread is doing: trying to connect to the source, waiting for events from the source, reconnecting to the source, and so on. For a listing of possible states, see Section 8.14.5, “Replication I/O (Receiver) Thread States” .

• Source\_Host

The source host that the replica is connected to.

• Source\_User

The user name of the account used to connect to the source.

• Source\_Port

The port used to connect to the source.

• Connect\_Retry

The number of seconds between connect retries (default 60). This can be set with a CHANGE REPLICATION SOURCE TO statement (from MySQL 8.0.23) or CHANGE MASTER TO statement (before MySQL 8.0.23).

• Source\_Log\_File

The name of the source binary log file from which the I/O (receiver) thread is currently reading. This is set to INVALID for a replication channel with the GTID\_ONLY setting after a server start. It will be updated when the replica contacts the source.

• Read\_Source\_Log\_Pos

The position in the current source binary log file up to which the I/O (receiver) thread has read. This is set to zero for a replication channel with the GTID\_ONLY setting after a server start. It will be updated when the replica contacts the source.

• Relay\_Log\_File

The name of the relay log file from which the SQL (applier) thread is currently reading and executing.

• Relay\_Log\_Pos

The position in the current relay log file up to which the SQL (applier) thread has read and executed.

• Relay\_Source\_Log\_File

The name of the source binary log file containing the most recent event executed by the SQL (applier) thread. This is set to INVALID for a replication channel with the GTID\_ONLY setting after a server start. It will be updated when a transaction is executed or skipped.

• Replica\_IO\_Running



Whether the replication I/O (receiver) thread is started and has connected successfully to the source. Internally, the state of this thread is represented by one of the following three values:

• **MYSQL\_REPLICA\_NOT\_RUN.** The replication I/O (receiver) thread is not running. For this state, Replica\_IO\_Running is No.

• **MYSQL\_REPLICA\_RUN\_NOT\_CONNECT.** The replication I/O (receiver) thread is running, but is not connected to a replication source. For this state, Replica\_IO\_Running is Connecting.

• **MYSQL\_REPLICA\_RUN\_CONNECT.** The replication I/O (receiver) thread is running, and is connected to a replication source. For this state, Replica\_IO\_Running is Yes.

• Replica\_SQL\_Running

Whether the replication SQL (applier) thread is started.

• Replicate\_Do\_DB, Replicate\_Ignore\_DB

The names of any databases that were specified with the --replicate-do-db and -- replicate-ignore-db options, or the CHANGE REPLICATION FILTER statement. If the FOR CHANNEL clause was used, the channel specific replication filters are shown. Otherwise, the replication filters for every replication channel are shown.

• Replicate\_Do\_Table, Replicate\_Ignore\_Table, Replicate\_Wild\_Do\_Table, Replicate\_Wild\_Ignore\_Table

The names of any tables that were specified with the --replicate-do-table, --replicate- ignore-table, --replicate-wild-do-table, and --replicate-wild-ignore-table options, or the CHANGE REPLICATION FILTER statement. If the FOR CHANNEL clause was used, the channel specific replication filters are shown. Otherwise, the replication filters for every replication channel are shown.

• Last\_Errno, Last\_Error

These columns are aliases for Last\_SQL\_Errno and Last\_SQL\_Error.

Issuing RESET MASTER or RESET REPLICA resets the values shown in these columns.

 **Note**

 When the replication SQL thread receives an error, it reports the error first,

then stops the SQL thread. This means that there is a small window of time during which SHOW REPLICA STATUS shows a nonzero value for Last\_SQL\_Errno even though Replica\_SQL\_Running still displays Yes.

• Skip\_Counter

The current value of the sql\_slave\_skip\_counter system variable. See [SET GLOBAL](https://dev.mysql.com/doc/refman/5.7/en/set-global-sql-slave-skip-counter.html) [sql\_slave\_skip\_counter Syntax](https://dev.mysql.com/doc/refman/5.7/en/set-global-sql-slave-skip-counter.html).

• Exec\_Source\_Log\_Pos

The position in the current source binary log file to which the replication SQL thread has read and executed, marking the start of the next transaction or event to be processed. This is set to zero for a replication channel with the GTID\_ONLY setting after a server start. It will be updated when a transaction is executed or skipped.

You can use this value with the CHANGE REPLICATION SOURCE TO statement's SOURCE\_LOG\_POS option (from MySQL 8.0.23) or the CHANGE MASTER TO statement's MASTER\_LOG\_POS option (before MySQL 8.0.23) when starting a new replica from an existing replica, so that the new replica reads from this point. The coordinates given by

(Relay\_Source\_Log\_File, Exec\_Source\_Log\_Pos) in the source's binary log correspond to the coordinates given by (Relay\_Log\_File, Relay\_Log\_Pos) in the relay log.

Inconsistencies in the sequence of transactions from the relay log which have been executed can cause this value to be a “low-water mark” . In other words, transactions appearing before the position are guaranteed to have committed, but transactions after the position may have committed or not. If these gaps need to be corrected, use START REPLICA UNTIL SQL\_AFTER\_MTS\_GAPS. See Section 17.5.1.34, “Replication and Transaction Inconsistencies” for more information.

• Relay\_Log\_Space

The total combined size of all existing relay log files.

• Until\_Condition, Until\_Log\_File, Until\_Log\_Pos

The values specified in the UNTIL clause of the START REPLICA statement.

Until\_Condition has these values:

• None if no UNTIL clause was specified.

• Source if the replica is reading until a given position in the source's binary log.

• Relay if the replica is reading until a given position in its relay log.

• SQL\_BEFORE\_GTIDS if the replication SQL thread is processing transactions until it has reached the first transaction whose GTID is listed in the gtid\_set.

• SQL\_AFTER\_GTIDS if the replication threads are processing all transactions until the last transaction in the gtid\_set has been processed by both threads.

• SQL\_AFTER\_MTS\_GAPS if a multithreaded replica's SQL threads are running until no more gaps are found in the relay log.

Until\_Log\_File and Until\_Log\_Pos indicate the log file name and position that define the coordinates at which the replication SQL thread stops executing.

For more information on UNTIL clauses, see Section 13.4.2.9, “START SLAVE Statement” .

• Source\_SSL\_Allowed, Source\_SSL\_CA\_File, Source\_SSL\_CA\_Path, Source\_SSL\_Cert, Source\_SSL\_Cipher, Source\_SSL\_CRL\_File, Source\_SSL\_CRL\_Path, Source\_SSL\_Key, Source\_SSL\_Verify\_Server\_Cert

These fields show the SSL parameters used by the replica to connect to the source, if any. Source\_SSL\_Allowed has these values:

• Yes if an SSL connection to the source is permitted.

• No if an SSL connection to the source is not permitted.

• Ignored if an SSL connection is permitted but the replica server does not have SSL support enabled.

The values of the other SSL-related fields correspond to the values of the SOURCE\_SSL\_\* options of the CHANGE REPLICATION SOURCE TO statement (from MySQL 8.0.23), or the MASTER\_SSL\_\* options of the CHANGE MASTER TO statement (before MySQL 8.0.23). See Section 13.4.2.1,

“CHANGE MASTER TO Statement” .

• Seconds\_Behind\_Source

This field is an indication of how “late” the replica is:

• When the replica is actively processing updates, this field shows the difference between the current timestamp on the replica and the original timestamp logged on the source for the event currently being processed on the replica.

• When no event is currently being processed on the replica, this value is 0.

In essence, this field measures the time difference in seconds between the replication SQL (applier) thread and the replication I/O (receiver) thread. If the network connection between source and replica is fast, the replication receiver thread is very close to the source, so this field is a good approximation of how late the replication applier thread is compared to the source. If the network is slow, this is *not* a good approximation; the replication applier thread may quite often be caught up with the slow- reading replication receiver thread, so Seconds\_Behind\_Source often shows a value of 0, even if the replication receiver thread is late compared to the source. In other words, *this* *column* *is* *useful* *only* *for* *fast* *networks*.

This time difference computation works even if the source and replica do not have identical clock times, provided that the difference, computed when the replica receiver thread starts, remains constant from then on. Any changes, including NTP updates, can lead to clock skews that can make calculation of Seconds\_Behind\_Source less reliable.

In MySQL 8.0, this field is NULL (undefined or unknown) if the replication applier thread is not running, or if the applier thread has consumed all of the relay log and the replication receiver thread is not running. (In older versions of MySQL, this field was NULL if the replication applier thread or the replication receiver thread was not running or was not connected to the source.) If the replication receiver thread is running but the relay log is exhausted, Seconds\_Behind\_Source is set to 0.

The value of Seconds\_Behind\_Source is based on the timestamps stored in events, which are preserved through replication. This means that if a source M1 is itself a replica of M0, any event from M1's binary log that originates from M0's binary log has M0's timestamp for that event. This enables MySQL to replicate TIMESTAMP successfully. However, the problem for Seconds\_Behind\_Source is that if M1 also receives direct updates from clients, the Seconds\_Behind\_Source value randomly fluctuates because sometimes the last event from M1 originates from M0 and sometimes is the result of a direct update on M1.

When using a multithreaded replica, you should keep in mind that this value is based on Exec\_Source\_Log\_Pos, and so may not reflect the position of the most recently committed transaction.

• Last\_IO\_Errno, Last\_IO\_Error

The error number and error message of the most recent error that caused the replication I/O (receiver) thread to stop. An error number of 0 and message of the empty string mean “no error.” If the Last\_IO\_Error value is not empty, the error values also appear in the replica's error log.

I/O error information includes a timestamp showing when the most recent I/O (receiver)thread error occurred. This timestamp uses the format *YYMMDD* *hh:mm:ss*, and appears in the Last\_IO\_Error\_Timestamp column.

Issuing RESET MASTER or RESET REPLICA resets the values shown in these columns.

• Last\_SQL\_Errno, Last\_SQL\_Error

The error number and error message of the most recent error that caused the replication SQL (applier) thread to stop. An error number of 0 and message of the empty string mean “no error.” If the Last\_SQL\_Error value is not empty, the error values also appear in the replica's error log.

If the replica is multithreaded, the replication SQL thread is the coordinator for worker threads. In this case, the Last\_SQL\_Error field shows exactly what the Last\_Error\_Message column in the Performance Schema replication\_applier\_status\_by\_coordinator table shows.



The field value is modified to suggest that there may be more failures in the other worker threads which can be seen in the replication\_applier\_status\_by\_worker table that shows each worker thread's status. If that table is not available, the replica error log can be used. The log or the replication\_applier\_status\_by\_worker table should also be used to learn more about the failure shown by [SHOW REPLICA STATUS](#_bookmark8) or the coordinator table.

SQL error information includes a timestamp showing when the most recent SQL (applier) thread error occurred. This timestamp uses the format *YYMMDD* *hh:mm:ss*, and appears in the Last\_SQL\_Error\_Timestamp column.

Issuing RESET MASTER or RESET REPLICA resets the values shown in these columns.

In MySQL 8.0, all error codes and messages displayed in the Last\_SQL\_Errno and Last\_SQL\_Error columns correspond to error values listed in [Server Error Message Reference](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html). This was not always true in previous versions. (Bug #11760365, Bug #52768)

• Replicate\_Ignore\_Server\_Ids

Any server IDs that have been specified using the IGNORE\_SERVER\_IDS option of the CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO statement, so that the replica ignores events from these servers. This option is used in a circular or other multi-source replication setup when one of the servers is removed. If any server IDs have been set in this way, a comma-delimited list of one or more numbers is shown. If no server IDs have been set, the field is blank.

**Note**

The Ignored\_server\_ids value in the slave\_master\_info table also shows the server IDs to be ignored, but as a space-delimited list, preceded by the total number of server IDs to be ignored. For example, if a CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO statement containing the IGNORE\_SERVER\_IDS = (2,6,9) option has been issued to tell a replica to ignore sources having the server ID 2, 6, or 9, that information appears as shown here:

Replicate\_Ignore\_Server\_Ids: 2, 6, 9

Ignored\_server\_ids: 3, 2, 6, 9

Replicate\_Ignore\_Server\_Ids filtering is performed by the I/O (receiver) thread, rather than by the SQL (applier) thread, which means that events which are filtered out are not written to the relay log. This differs from the filtering actions taken by server options such --replicate-do-table, which apply to the applier thread.

**Note**

From MySQL 8.0, a deprecation warning is issued if SET GTID\_MODE=ON is issued when any channel has existing server IDs set with IGNORE\_SERVER\_IDS. Before starting GTID-based replication, use [SHOW REPLICA STATUS](#_bookmark8) to check for and clear all ignored server ID

lists on the servers involved. You can clear a list by issuing a CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO statement containing the IGNORE\_SERVER\_IDS option with an empty list.

• Source\_Server\_Id

The server\_id value from the source.

• Source\_UUID

The server\_uuid value from the source.

• Source\_Info\_File

The location of the master.info file, the use of which is now deprecated. By default from MySQL 8.0, a table is used instead for the replica's connection metadata repository.

• SQL\_Delay

The number of seconds that the replica must lag the source.

• SQL\_Remaining\_Delay

When Replica\_SQL\_Running\_State is Waiting until MASTER\_DELAY seconds after source executed event, this field contains the number of delay seconds remaining. At other times, this field is NULL.

• Replica\_SQL\_Running\_State

The state of the SQL thread (analogous to Replica\_IO\_State). The value is identical to the State value of the SQL thread as displayed by [SHOW PROCESSLIST](#_bookmark1). Section 8.14.6, “Replication SQL Thread States” , provides a listing of possible states.

• Source\_Retry\_Count

The number of times the replica can attempt to reconnect to the source in the event of a lost connection. This value can be set using the SOURCE\_RETRY\_COUNT | MASTER\_RETRY\_COUNT option of the CHANGE REPLICATION SOURCE TO statement (from MySQL 8.0.23) or CHANGE MASTER TO statement (before MySQL 8.0.23), or the older --master-retry-count server option (still supported for backward compatibility).

• Source\_Bind

The network interface that the replica is bound to, if any. This is set using the SOURCE\_BIND | MASTER\_BIND option for the CHANGE REPLICATION SOURCE TO statement (from MySQL 8.0.23) or CHANGE MASTER TO statement (before MySQL 8.0.23).

• Last\_IO\_Error\_Timestamp

A timestamp in *YYMMDD* *hh:mm:ss* format that shows when the most recent I/O error took place.

• Last\_SQL\_Error\_Timestamp

A timestamp in *YYMMDD* *hh:mm:ss* format that shows when the most recent SQL error occurred.

• Retrieved\_Gtid\_Set

The set of global transaction IDs corresponding to all transactions received by this replica. Empty if GTIDs are not in use. See GTID Sets for more information.

This is the set of all GTIDs that exist or have existed in the relay logs. Each GTID is added as soon as the Gtid\_log\_event is received. This can cause partially transmitted transactions to have their GTIDs included in the set.

When all relay logs are lost due to executing RESET REPLICA or CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO, or due to the effects of the --relay-log-recovery option, the set is cleared. When relay\_log\_purge = 1, the newest relay log is always kept, and the set is not cleared.

• Executed\_Gtid\_Set

The set of global transaction IDs written in the binary log. This is the same as the value for the global gtid\_executed system variable on this server, as well as the value for Executed\_Gtid\_Set in the output of SHOW MASTER STATUS on this server. Empty if GTIDs are not in use. See GTID Sets for more information.

• Auto\_Position

1 if GTID auto-positioning is in use for the channel, otherwise 0.

• Replicate\_Rewrite\_DB

The Replicate\_Rewrite\_DB value displays any replication filtering rules that were specified. For example, if the following replication filter rule was set:

|  |  |
| --- | --- |
| CHANGE REPLICATION FILTER REPLICATE\_REWRITE\_DB=((db1,db2), | (db3,db4)); |
| the Replicate\_Rewrite\_DB value displays: |  |
| Replicate Rewrite DB: (db1,db2),(db3,db4) |  |

For more information, see Section 13.4.2.2, “CHANGE REPLICATION FILTER Statement” .

• Channel\_name

The replication channel which is being displayed. There is always a default replication channel, and more replication channels can be added. See Section 17.2.2, “Replication Channels” for more information.

• Master\_TLS\_Version

The TLS version used on the source. For TLS version information, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers” .

• Source\_public\_key\_path

The path name to a file containing a replica-side copy of the public key required by the source for RSA key pair-based password exchange. The file must be in PEM format. This column applies to replicas that authenticate with the sha256\_password or caching\_sha2\_password authentication plugin.

If Source\_public\_key\_path is given and specifies a valid public key file, it takes precedence over Get\_source\_public\_key.

• Get\_source\_public\_key

Whether to request from the source the public key required for RSA key pair-based password exchange. This column applies to replicas that authenticate with the caching\_sha2\_password authentication plugin. For that plugin, the source does not send the public key unless requested.

If Source\_public\_key\_path is given and specifies a valid public key file, it takes precedence over Get\_source\_public\_key.

• Network\_Namespace

The network namespace name; empty if the connection uses the default (global) namespace. For information about network namespaces, see Section 5.1.14, “Network Namespace Support” . This column was added in MySQL 8.0.22.

**13.7.7.36** **SHOW** **SLAVE** **|** **REPLICA** **STATUS** **Statement**

SHOW {SLAVE | REPLICA} STATUS [FOR CHANNEL *channel*]

This statement provides status information on essential parameters of the replica threads. From MySQL 8.0.22, [SHOW SLAVE STATUS](#_bookmark9) is deprecated and the alias [SHOW REPLICA STATUS](#_bookmark8) should be used instead. The statement works in the same way as before, only the terminology used for the statement and its output has changed. Both versions of the statement update the same status variables when used. Please see the documentation for [SHOW REPLICA STATUS](#_bookmark8) for a description of the statement.

**13.7.7.37** **SHOW** **STATUS** **Statement**

SHOW [GLOBAL | SESSION] STATUS

[LIKE '*pattern* ' | WHERE *expr*]

[SHOW STATUS](#_bookmark10) provides server status information (see Section 5.1.10, “Server Status Variables” ). This statement does not require any privilege. It requires only the ability to connect to the server.

Status variable information is also available from these sources:

• Performance Schema tables. See Section 27.12.15, “Performance Schema Status Variable Tables” .

• The mysqladmin extended-status command. See Section 4.5.2, “mysqladmin — A MySQL Server Administration Program” .

For [SHOW STATUS](#_bookmark10), a LIKE clause, if present, indicates which variable names to match. A WHERE clause can be given to select rows using more general conditions, as discussed in Section 26.8, “Extensions to SHOW Statements” .

[SHOW STATUS](#_bookmark10) accepts an optional GLOBAL or SESSION variable scope modifier:

• With a GLOBAL modifier, the statement displays the global status values. A global status variable may represent status for some aspect of the server itself (for example, Aborted\_connects), or the aggregated status over all connections to MySQL (for example, Bytes\_received and Bytes\_sent). If a variable has no global value, the session value is displayed.

• With a SESSION modifier, the statement displays the status variable values for the current connection. If a variable has no session value, the global value is displayed. LOCAL is a synonym for SESSION.

• If no modifier is present, the default is SESSION.

The scope for each status variable is listed at Section 5.1.10, “Server Status Variables” .

Each invocation of the [SHOW STATUS](#_bookmark10) statement uses an internal temporary table and increments the global Created\_tmp\_tables value.

Partial output is shown here. The list of names and values may differ for your server. The meaning of each variable is given in Section 5.1.10, “Server Status Variables” .

mysql> **SHOW** **STATUS;**

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

|  |  |  |  |
| --- | --- | --- | --- |
| | Variable\_name | | | Value | | |

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

|  |  |  |  |
| --- | --- | --- | --- |
| | Aborted\_clients  | Aborted\_connects  | Bytes\_received  | Bytes\_sent  | Connections  | Created\_tmp\_disk\_tables  | Created\_tmp\_tables  | Created\_tmp\_files  ...  | Open\_tables  | Open\_files  | Open\_streams  | Opened\_tables  | Questions  ...  | Table\_locks\_immediate  | Table\_locks\_waited  | Threads\_cached  | Threads\_created  | Threads\_connected  | Threads\_running  | Uptime | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | 0  0  155372598  1176560426  30023  0  8340  60  1  2  0  44600  2026873  1920382  0  0  30022  1  1  80380 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |

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

With a LIKE clause, the statement displays only rows for those variables with names that match the pattern:

mysql> **SHOW** **STATUS** **LIKE** **'Key%';**

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

| Variable\_name | Value |

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

|

| Key\_blocks\_used

| Key\_read\_requests

| Key\_reads

| Key\_write\_requests

| Key\_writes

|

14955

96854827

162040

7589728

3813196

|

|

|

|

|

|

|

|

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

**13.7.7.38** **SHOW** **TABLE** **STATUS** **Statement**

SHOW TABLE STATUS

[{FROM | IN} *db\_name*]

[LIKE '*pattern* ' | WHERE *expr*]

[SHOW TABLE STATUS](#_bookmark11) works likes [SHOW TABLES](#_bookmark12), but provides a lot of information about each non-TEMPORARY table. You can also get this list using the mysqlshow --status *db\_name* command. The LIKE clause, if present, indicates which table names to match. The WHERE clause can be given to select rows using more general conditions, as discussed in Section 26.8, “Extensions to

SHOW Statements” .

This statement also displays information about views.

[SHOW TABLE STATUS](#_bookmark11) output has these columns:

• Name

The name of the table.

• Engine

The storage engine for the table. See Chapter 15, *The* *InnoDB* *Storage* *Engine*, and Chapter 16, *Alternative* *Storage* *Engines*.

For partitioned tables, Engine shows the name of the storage engine used by all partitions.

• Version

This column is unused. With the removal of .frm files in MySQL 8.0, this column now reports a hardcoded value of 10, which is the last .frm file version used in MySQL 5.7.

• Row\_format

The row-storage format (Fixed, Dynamic, Compressed, Redundant, Compact). For MyISAM tables, Dynamic corresponds to what myisamchk -dvv reports as Packed.

• Rows

The number of rows. Some storage engines, such as MyISAM, store the exact count. For other storage engines, such as InnoDB, this value is an approximation, and may vary from the actual value by as much as 40% to 50%. In such cases, use SELECT COUNT(\*) to obtain an accurate count.

The Rows value is NULL for INFORMATION\_SCHEMA tables.

For InnoDB tables, the row count is only a rough estimate used in SQL optimization. (This is also true if the InnoDB table is partitioned.)

• Avg\_row\_length

The average row length.

• Data\_length

For MyISAM, Data\_length is the length of the data file, in bytes.

For InnoDB, Data\_length is the approximate amount of space allocated for the clustered index, in

bytes. Specifically, it is the clustered index size, in pages, multiplied by the InnoDB page size. Refer to the notes at the end of this section for information regarding other storage engines.

• Max\_data\_length

For MyISAM, Max\_data\_length is maximum length of the data file. This is the total number of bytes of data that can be stored in the table, given the data pointer size used.

Unused for InnoDB.

Refer to the notes at the end of this section for information regarding other storage engines.

• Index\_length

For MyISAM, Index\_length is the length of the index file, in bytes.

For InnoDB, Index\_length is the approximate amount of space allocated for non-clustered indexes, in bytes. Specifically, it is the sum of non-clustered index sizes, in pages, multiplied by the InnoDB page size.

Refer to the notes at the end of this section for information regarding other storage engines.

• Data\_free

The number of allocated but unused bytes.

InnoDB tables report the free space of the tablespace to which the table belongs. For a table located in the shared tablespace, this is the free space of the shared tablespace. If you are using multiple tablespaces and the table has its own tablespace, the free space is for only that table. Free space means the number of bytes in completely free extents minus a safety margin. Even if free space displays as 0, it may be possible to insert rows as long as new extents need not be allocated.

For NDB Cluster, Data\_free shows the space allocated on disk for, but not used by, a Disk Data table or fragment on disk. (In-memory data resource usage is reported by the Data\_length column.)

For partitioned tables, this value is only an estimate and may not be absolutely correct. A more accurate method of obtaining this information in such cases is to query the INFORMATION\_SCHEMA PARTITIONS table, as shown in this example:

SELECT SUM(DATA\_FREE)

FROM INFORMATION\_SCHEMA.PARTITIONS

WHERE TABLE\_SCHEMA = 'mydb'

AND TABLE\_NAME = 'mytable';

For more information, see Section 26.3.21, “The INFORMATION\_SCHEMA PARTITIONS Table” .

• Auto\_increment

The next AUTO\_INCREMENT value.

• Create\_time

When the table was created.

• Update\_time

When the data file was last updated. For some storage engines, this value is NULL. For example, InnoDB stores multiple tables in its system tablespace and the data file timestamp does not apply. Even with file-per-table mode with each InnoDB table in a separate .ibd file, change buffering can delay the write to the data file, so the file modification time is different from the time of the last insert, update, or delete. For MyISAM, the data file timestamp is used; however, on Windows the timestamp is not updated by updates, so the value is inaccurate.

Update\_time displays a timestamp value for the last UPDATE, INSERT, or DELETE performed on InnoDB tables that are not partitioned. For MVCC, the timestamp value reflects the COMMIT time, which is considered the last update time. Timestamps are not persisted when the server is restarted or when the table is evicted from the InnoDB data dictionary cache.

• Check\_time

When the table was last checked. Not all storage engines update this time, in which case, the value is always NULL.

For partitioned InnoDB tables, Check\_time is always NULL.

• Collation

The table default collation. The output does not explicitly list the table default character set, but the collation name begins with the character set name.

• Checksum

The live checksum value, if any.

• Create\_options

Extra options used with CREATE TABLE.

Create\_options shows partitioned for a partitioned table.

Prior to MySQL 8.0.16, Create\_options shows the ENCRYPTION clause specified for tables created in file-per-table tablespaces. As of MySQL 8.0.16, it shows the encryption clause for file- per-table tablespaces if the table is encrypted or if the specified encryption differs from the schema encryption. The encryption clause is not shown for tables created in general tablespaces. To identify encrypted file-per-table and general tablespaces, query the INNODB\_TABLESPACES ENCRYPTION column.

When creating a table with strict mode disabled, the storage engine's default row format is used if the specified row format is not supported. The actual row format of the table is reported in the Row\_format column. Create\_options shows the row format that was specified in the CREATE TABLE statement.

When altering the storage engine of a table, table options that are not applicable to the new storage engine are retained in the table definition to enable reverting the table with its previously defined options to the original storage engine, if necessary. Create\_options may show retained options.

• Comment

The comment used when creating the table (or information as to why MySQL could not access the table information).

**Notes**

• For NDB tables, the output of this statement shows appropriate values for the Avg\_row\_length and Data\_length columns, with the exception that BLOB columns are not taken into account.

• For NDB tables, Data\_length includes data stored in main memory only; the Max\_data\_length and Data\_free columns apply to Disk Data.

• For NDB Cluster Disk Data tables, Max\_data\_length shows the space allocated for the disk part of a Disk Data table or fragment. (In-memory data resource usage is reported by the Data\_length column.)

• For MEMORY tables, the Data\_length, Max\_data\_length, and Index\_length values approximate the actual amount of allocated memory. The allocation algorithm reserves memory in large amounts to reduce the number of allocation operations.

• For views, most columns displayed by [SHOW TABLE STATUS](#_bookmark11) are 0 or NULL except that Name indicates the view name, Create\_time indicates the creation time, and Comment says VIEW.

Table information is also available from the INFORMATION\_SCHEMA TABLES table. See Section 26.3.38, “The INFORMATION\_SCHEMA TABLES Table” .

**13.7.7.39** **SHOW** **TABLES** **Statement**

SHOW [EXTENDED] [FULL] TABLES

[{FROM | IN} *db\_name*]

[LIKE '*pattern* ' | WHERE *expr*]

[SHOW TABLES](#_bookmark12) lists the non-TEMPORARY tables in a given database. You can also get this list using the mysqlshow *db\_name* command. The LIKE clause, if present, indicates which table names to match. The WHERE clause can be given to select rows using more general conditions, as discussed in Section 26.8, “Extensions to SHOW Statements” .

Matching performed by the LIKE clause is dependent on the setting of the

lower\_case\_table\_names system variable.

The optional EXTENDED modifier causes [SHOW TABLES](#_bookmark12) to list hidden tables created by failed ALTER TABLE statements. These temporary tables have names beginning with #sql and can be dropped using DROP TABLE.

This statement also lists any views in the database. The optional FULL modifier causes [SHOW TABLES](#_bookmark12) to display a second output column with values of BASE TABLE for a table, VIEW for a view, or SYSTEM VIEW for an INFORMATION\_SCHEMA table.

If you have no privileges for a base table or view, it does not show up in the output from [SHOW TABLES](#_bookmark12) or mysqlshow db\_name.

Table information is also available from the INFORMATION\_SCHEMA TABLES table. See Section 26.3.38, “The INFORMATION\_SCHEMA TABLES Table” .

**13.7.7.40** **SHOW** **TRIGGERS** **Statement**

SHOW TRIGGERS

[{FROM | IN} *db\_name*]

[LIKE '*pattern* ' | WHERE *expr*]

[SHOW TRIGGERS](#_bookmark13) lists the triggers currently defined for tables in a database (the default database unless a FROM clause is given). This statement returns results only for databases and tables for which you have the TRIGGER privilege. The LIKE clause, if present, indicates which table names (not trigger names) to match and causes the statement to display triggers for those tables. The WHERE clause can be given to select rows using more general conditions, as discussed in Section 26.8, “Extensions to

SHOW Statements” .

For the ins\_sum trigger defined in Section 25.3, “Using Triggers” , the output of [SHOW TRIGGERS](#_bookmark13) is as shown here:

mysql> **SHOW** **TRIGGERS** **LIKE** **'acc%'\G**

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

Trigger: ins\_sum

Event: INSERT

Table: account

Statement: SET @sum = @sum + NEW.amount

Timing: BEFORE

Created: 2018-08-08 10:10:12.61

sql\_mode: ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,

NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,

ERROR\_FOR\_DIVISION\_BY\_ZERO,

NO\_ENGINE\_SUBSTITUTION

Definer: me@localhost

character\_set\_client: utf8mb4

collation\_connection: utf8mb4\_0900\_ai\_ci

Database Collation: utf8mb4\_0900\_ai\_ci

[SHOW TRIGGERS](#_bookmark13) output has these columns:

• Trigger

The name of the trigger.

• Event

The trigger event. This is the type of operation on the associated table for which the trigger activates. The value is INSERT (a row was inserted), DELETE (a row was deleted), or UPDATE (a row was modified).

• Table

The table for which the trigger is defined.

• Statement

The trigger body; that is, the statement executed when the trigger activates.

• Timing

Whether the trigger activates before or after the triggering event. The value is BEFORE or AFTER.

• Created

The date and time when the trigger was created. This is a TIMESTAMP(2) value (with a fractional part in hundredths of seconds) for triggers.

• sql\_mode

The SQL mode in effect when the trigger was created, and under which the trigger executes. For the permitted values, see Section 5.1.11, “Server SQL Modes” .

• Definer

The account of the user who created the trigger, in '*user\_name*'@'*host\_name*' format.

• character\_set\_client

The session value of the character\_set\_client system variable when the trigger was created.

• collation\_connection

The session value of the collation\_connection system variable when the trigger was created.

• Database Collation

The collation of the database with which the trigger is associated.

Trigger information is also available from the INFORMATION\_SCHEMA TRIGGERS table. See Section 26.3.45, “The INFORMATION\_SCHEMA TRIGGERS Table” .

**13.7.7.41** **SHOW** **VARIABLES** **Statement**

SHOW [GLOBAL | SESSION] VARIABLES

[LIKE '*pattern* ' | WHERE *expr*]

[SHOW VARIABLES](#_bookmark14) shows the values of MySQL system variables (see Section 5.1.8, “Server System Variables” ). This statement does not require any privilege. It requires only the ability to connect to the server.

System variable information is also available from these sources:

• Performance Schema tables. See Section 27.12.14, “Performance Schema System Variable Tables” .

• The mysqladmin variables command. See Section 4.5.2, “mysqladmin — A MySQL Server Administration Program” .

For [SHOW VARIABLES](#_bookmark14), a LIKE clause, if present, indicates which variable names to match. A WHERE clause can be given to select rows using more general conditions, as discussed in Section 26.8, “Extensions to SHOW Statements” .

[SHOW VARIABLES](#_bookmark14) accepts an optional GLOBAL or SESSION variable scope modifier:

• With a GLOBAL modifier, the statement displays global system variable values. These are the values used to initialize the corresponding session variables for new connections to MySQL. If a variable has no global value, no value is displayed.

• With a SESSION modifier, the statement displays the system variable values that are in effect for the current connection. If a variable has no session value, the global value is displayed. LOCAL is a synonym for SESSION.

• If no modifier is present, the default is SESSION.

The scope for each system variable is listed at Section 5.1.8, “Server System Variables” .

[SHOW VARIABLES](#_bookmark14) is subject to a version-dependent display-width limit. For variables with very long values that are not completely displayed, use SELECT as a workaround. For example:

SELECT @@GLOBAL.innodb\_data\_file\_path;

Most system variables can be set at server startup (read-only variables such as version\_comment are exceptions). Many can be changed at runtime with the SET statement. See Section 5.1.9, “Using System Variables” , and Section 13.7.6.1, “SET Syntax for Variable Assignment” .

Partial output is shown here. The list of names and values may differ for your server. Section 5.1.8, “Server System Variables” , describes the meaning of each variable, and Section 5.1.1, “Configuring the Server” , provides information about tuning them.

mysql> **SHOW** **VARIABLES;**

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

|  |  |  |  |
| --- | --- | --- | --- |
| | Variable\_name | | | Value | | |

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| | | activate\_all\_roles\_on\_login | | | OFF | | |
| | | auto\_generate\_certs | | | ON | | |
| | | auto\_increment\_increment | | | 1 | | |
| | | auto\_increment\_offset | | | 1 | | |
| | | autocommit | | | ON | | |
| | | automatic\_sp\_privileges | | | ON | | |
| | | avoid\_temporal\_upgrade | | | OFF | | |
| | | back\_log | | | 151 | | |

|  |  |  |
| --- | --- | --- |
| | basedir |  big\_tables  bind\_address  | binlog\_cache\_size |  | binlog\_checksum |  | binlog\_direct\_non\_transactional\_updates |  binlog\_error\_action  binlog\_expire\_logs\_seconds  | binlog\_format |  | binlog\_group\_commit\_sync\_delay |  | binlog\_group\_commit\_sync\_no\_delay\_count |  binlog\_gtid\_simple\_recovery  binlog\_max\_flush\_queue\_time  | binlog\_order\_commits |  | binlog\_row\_image |  | binlog\_row\_metadata |  binlog\_row\_value\_options  binlog\_rows\_query\_log\_events  | binlog\_stmt\_cache\_size |  | binlog\_transaction\_dependency\_history\_size |  | binlog\_transaction\_dependency\_tracking |  block\_encryption\_mode  bulk insert buffer size | /usr/  OFF  \*  32768  CRC32  OFF  ABORT\_SERVER  2592000  ROW  0  0  ON  0  ON  FULL  MINIMAL  OFF  32768  25000  COMMIT\_ORDER  aes-128-ecb  8388608 | |  |  |  |  |  |  |  |  |  |  |  |  | |

...

|

|

|

|

|

|

|

|

|

|

|

|

|

max\_allowed\_packet

max\_binlog\_cache\_size

max\_binlog\_size

max\_binlog\_stmt\_cache\_size

max\_connect\_errors

max\_connections

max\_delayed\_threads

max\_digest\_length

max\_error\_count

max\_execution\_time

max\_heap\_table\_size

max\_insert\_delayed\_threads

max\_join\_size

|

|

|

|

|

|

|

|

|

|

|

|

|

67108864

18446744073709547520

1073741824

18446744073709547520

100

151

20

1024

1024

0

16777216

20

18446744073709551615

|

|

|

|

|

|

|

|

|

|

|

|

|

...

|  |  |  |  |
| --- | --- | --- | --- |
| | thread\_handling  thread\_stack  time\_zone  | timestamp  | tls\_version  | tmp\_table\_size  tmpdir  transaction\_alloc\_block\_size  | transaction\_allow\_batching  | transaction\_isolation  | transaction\_prealloc\_size  transaction\_read\_only  transaction\_write\_set\_extraction  | unique\_checks  | updatable\_views\_with\_limit  | version  version\_comment  version\_compile\_machine  | version\_compile\_os  | version\_compile\_zlib  | wait\_timeout  warning\_count  windowing use high precision |  | one-thread-per-connection  286720  SYSTEM  1530906638.765316  TLSv1.2,TLSv1.3  16777216  /tmp  8192  OFF  REPEATABLE-READ  4096  OFF  XXHASH64  ON  YES  8.0.12  MySQL Community Server - GPL  x86\_64  Linux  1.2.11  28800  0  ON | |  |  |  |  |  |  |  |  |  |  |  |  | |

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

With a LIKE clause, the statement displays only rows for those variables with names that match the pattern. To obtain the row for a specific variable, use a LIKE clause as shown:

SHOW VARIABLES LIKE 'max\_join\_size';

SHOW SESSION VARIABLES LIKE 'max\_join\_size';

To get a list of variables whose name match a pattern, use the % wildcard character in a LIKE clause:

SHOW VARIABLES LIKE '%size%';

SHOW GLOBAL VARIABLES LIKE '%size%';

Wildcard characters can be used in any position within the pattern to be matched. Strictly speaking, because \_ is a wildcard that matches any single character, you should escape it as \\_ to match it literally. In practice, this is rarely necessary.

**13.7.7.42** **SHOW** **WARNINGS** **Statement**

SHOW WARNINGS [LIMIT [*offset*,] *row\_count*]

SHOW COUNT(\*) WARNINGS

[SHOW WARNINGS](#_bookmark15) is a diagnostic statement that displays information about the conditions (errors, warnings, and notes) resulting from executing a statement in the current session. Warnings are generated for DML statements such as INSERT, UPDATE, and LOAD DATA as well as DDL statements such as CREATE TABLE and ALTER TABLE.

The LIMIT clause has the same syntax as for the SELECT statement. See Section 13.2.13, “SELECT

Statement” .

[SHOW WARNINGS](#_bookmark15) is also used following [EXPLAIN](#_bookmark16), to display the extended information generated by [EXPLAIN](#_bookmark16). See Section 8.8.3, “Extended EXPLAIN Output Format” .

[SHOW WARNINGS](#_bookmark15) displays information about the conditions resulting from execution of the most recent nondiagnostic statement in the current session. If the most recent statement resulted in an error during parsing, [SHOW WARNINGS](#_bookmark15) shows the resulting conditions, regardless of statement type (diagnostic or nondiagnostic).

The [SHOW COUNT(\*) WARNINGS](#_bookmark15) diagnostic statement displays the total number of errors, warnings, and notes. You can also retrieve this number from the warning\_count system variable:

SHOW COUNT(\*) WARNINGS;

SELECT @@warning\_count;

A difference in these statements is that the first is a diagnostic statement that does not clear the message list. The second, because it is a SELECT statement is considered nondiagnostic and does clear the message list.

A related diagnostic statement, SHOW ERRORS, shows only error conditions (it excludes warnings and notes), and [SHOW COUNT(\*) ERRORS](#_bookmark15) statement displays the total number of errors. See Section 13.7.7.17, “SHOW ERRORS Statement” . GET DIAGNOSTICS can be used to examine information for individual conditions. See Section 13.6.7.3, “GET DIAGNOSTICS Statement” .

Here is a simple example that shows data-conversion warnings for INSERT. The example assumes that strict SQL mode is disabled. With strict mode enabled, the warnings would become errors and terminate the INSERT.

mysql> **CREATE** **TABLE** **t1** **(a** **TINYINT** **NOT** **NULL,** **b** **CHAR(4));**

Query OK, 0 rows affected (0.05 sec)

mysql> **INSERT** **INTO** **t1** **VALUES(10,'mysql'),** **(NULL,'test'),** **(300,'xyz');**

Query OK, 3 rows affected, 3 warnings (0.00 sec)

Records: 3 Duplicates: 0 Warnings: [3](#_bookmark17)

mysql> **SHOW** **WARNINGS\G**

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

Level: Warning

Code: 1265

Message: Data truncated for column 'b' at row 1

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

Level: Warning

Code: 1048

Message: Column 'a' cannot be null

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 3. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Level: Warning

Code: 1264

Message: Out of range value for column 'a' at row 3

3 rows in set (0.00 sec)

The max\_error\_count system variable controls the maximum number of error, warning, and note messages for which the server stores information, and thus the number of messages that [SHOW](#_bookmark15) [WARNINGS](#_bookmark15) displays. To change the number of messages the server can store, change the value of max\_error\_count.

max\_error\_count controls only how many messages are stored, not how many are counted. The value of warning\_count is not limited by max\_error\_count, even if the number of messages generated exceeds max\_error\_count. The following example demonstrates this. The ALTER TABLE statement produces three warning messages (strict SQL mode is disabled for the example to prevent an error from occurring after a single conversion issue). Only one message is stored and displayed because max\_error\_count has been set to 1, but all three are counted (as shown by the value of warning\_count):

mysql> **SHOW** **VARIABLES** **LIKE** **'max\_error\_count';**

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

| Variable\_name | Value |

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

| max\_error\_count | 1024 |

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

1 row in set (0.00 sec)

mysql> **SET** **max\_error\_count=1,** **sql\_mode** **=** **'';**

Query OK, 0 rows affected (0.00 sec)

mysql> **ALTER** **TABLE** **t1** **MODIFY** **b** **CHAR;**

Query OK, 3 rows affected, 3 warnings (0.00 sec)

Records: 3 Duplicates: 0 Warnings: [3](#_bookmark18)

mysql> **SHOW** **WARNINGS;**

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

| Level | Code | Message |

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

| Warning | 1263 | Data truncated for column 'b' at row 1 |

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

1 row in set (0.00 sec)

mysql> **SELECT** **@@warning\_count;**

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

| @@warning\_count |

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

| 3 |

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

1 row in set (0.01 sec)

To disable message storage, set max\_error\_count to 0. In this case, warning\_count still indicates how many warnings occurred, but messages are not stored and cannot be displayed.

The sql\_notes system variable controls whether note messages increment warning\_count and whether the server stores them. By default, sql\_notes is 1, but if set to 0, notes do not increment warning\_count and the server does not store them:

mysql> **SET** **sql\_notes** **=** **1;**

mysql> **DROP** **TABLE** **IF** **EXISTS** **test** **.no\_such\_table;**

Query OK, 0 rows affected, 1 warning (0 .00 sec)

mysql> **SHOW** **WARNINGS;**

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

| Level | Code | Message |

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

| Note | 1051 | Unknown table 'test .no\_such\_table' |

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

1 row in set (0.00 sec)

*partition\_name* [, *partition\_name*] ...

| ALL

}

mysql> **SET** **sql\_notes** **=** **0;**

mysql> **DROP** **TABLE** **IF** **EXISTS** **test** **.no\_such\_table;**

Query OK, 0 rows affected (0 .00 sec)

mysql> **SHOW** **WARNINGS;**

Empty set (0.00 sec)

The MySQL server sends to each client a count indicating the total number of errors, warnings, and notes resulting from the most recent statement executed by that client. From the C API, this value can be obtained by calling [mysql\_warning\_count()](https://dev.mysql.com/doc/c-api/8.0/en/mysql-warning-count.html). See [mysql\_warning\_count()](https://dev.mysql.com/doc/c-api/8.0/en/mysql-warning-count.html).

In the mysql client, you can enable and disable automatic warnings display using the warnings and nowarning commands, respectively, or their shortcuts, \W and \w (see Section 4.5.1.2, “mysql Client Commands” ). For example:

mysql> **\W**

enabled .

Show warnings

mysql> **SELECT**

**1/0;**

+------+

| 1/0 |

+------+

| NULL |

+------+

1 row in set, 1 warning (0.03 sec)

Warning (Code 1365): Division by 0

mysql> **\w**

Show warnings disabled.

**13.7.8** **Other** **Administrative** **Statements**

**13.7.8.1** **BINLOG** **Statement**

BINLOG '*str*'

[BINLOG](#_bookmark19) is an internal-use statement. It is generated by the mysqlbinlog program as the printable representation of certain events in binary log files. (See Section 4.6.9, “mysqlbinlog — Utility for Processing Binary Log Files” .) The '*str*' value is a base 64-encoded string the that server decodes to determine the data change indicated by the corresponding event.

To execute [BINLOG](#_bookmark19) statements when applying mysqlbinlog output, a user account requires the BINLOG\_ADMIN privilege (or the deprecated SUPER privilege), or the REPLICATION\_APPLIER privilege plus the appropriate privileges to execute each log event.

This statement can execute only format description events and row events.

**13.7.8.2** **CACHE** **INDEX** **Statement**

CACHE INDEX {

*tbl\_index\_list* [, *tbl\_index\_list*] ...

| *tbl\_name* PARTITION (*partition\_list*)

}

IN *key\_cache\_name*

*tbl\_index\_list*:

*tbl\_name* [{INDEX |KEY} (*index\_name* [, *index\_name*] ...)]

*partition\_list*: {

The [CACHE INDEX](#_bookmark20) statement assigns table indexes to a specific key cache. It applies only to MyISAM tables, including partitioned MyISAM tables. After the indexes have been assigned, they can be preloaded into the cache if desired with [LOAD INDEX INTO CACHE](#_bookmark21).

The following statement assigns indexes from the tables t1, t2, and t3 to the key cache named hot\_cache:

mysql> **CACHE** **INDEX** **t1,** **t2,** **t3** **IN** **hot\_cache;**

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

|  |  |  |
| --- | --- | --- |
| | Table | | Op | | Msg\_type | Msg\_text | |

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

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | | test .t1 | | | assign\_to\_keycache | | | status | | | OK | | |
| | | test .t2 | | | assign\_to\_keycache | | | status | | | OK | | |
| | | test .t3 | | | assign\_to\_keycache | | | status | | | OK | | |

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

The syntax of [CACHE INDEX](#_bookmark20) enables you to specify that only particular indexes from a table should be assigned to the cache. However, the implementation assigns all the table's indexes to the cache, so there is no reason to specify anything other than the table name.

The key cache referred to in a [CACHE INDEX](#_bookmark20) statement can be created by setting its size with a parameter setting statement or in the server parameter settings. For example:

SET GLOBAL keycache1.key\_buffer\_size=128\*1024;

Key cache parameters are accessed as members of a structured system variable. See Section 5.1.9.5, “Structured System Variables” .

A key cache must exist before you assign indexes to it, or an error occurs:

mysql> **CACHE** **INDEX** **t1** **IN** **non\_existent\_cache;**

ERROR 1284 (HY000): Unknown key cache 'non\_existent\_cache'

By default, table indexes are assigned to the main (default) key cache created at the server startup. When a key cache is destroyed, all indexes assigned to it are reassigned to the default key cache.

Index assignment affects the server globally: If one client assigns an index to a given cache, this cache is used for all queries involving the index, no matter which client issues the queries.

[CACHE INDEX](#_bookmark20) is supported for partitioned MyISAM tables. You can assign one or more indexes for one, several, or all partitions to a given key cache. For example, you can do the following:

CREATE TABLE pt (c1 INT, c2 VARCHAR(50), INDEX i(c1))

ENGINE=MyISAM

PARTITION BY HASH(c1)

PARTITIONS 4;

SET GLOBAL kc\_fast.key\_buffer\_size = 128 \* 1024;

SET GLOBAL kc\_slow.key\_buffer\_size = 128 \* 1024;

CACHE INDEX pt PARTITION (p0) IN kc\_fast;

CACHE INDEX pt PARTITION (p1, p3) IN kc\_slow;

The previous set of statements performs the following actions:

• Creates a partitioned table with 4 partitions; these partitions are automatically named p0, ..., p3; this table has an index named i on column c1.

• Creates 2 key caches named kc\_fast and kc\_slow

• Assigns the index for partition p0 to the kc\_fast key cache and the index for partitions p1 and p3 to the kc\_slow key cache; the index for the remaining partition (p2) uses the server's default key

cache.

If you wish instead to assign the indexes for all partitions in table pt to a single key cache named kc\_all, you can use either of the following two statements:

CACHE INDEX pt PARTITION (ALL) IN kc\_all;



CACHE INDEX pt IN kc\_all;

The two statements just shown are equivalent, and issuing either one has exactly the same effect. In other words, if you wish to assign indexes for all partitions of a partitioned table to the same key cache, the PARTITION (ALL) clause is optional.

When assigning indexes for multiple partitions to a key cache, the partitions need not be contiguous, and you need not list their names in any particular order. Indexes for any partitions not explicitly assigned to a key cache automatically use the server default key cache.

Index preloading is also supported for partitioned MyISAM tables. For more information, see [Section 13.7.8.5, “LOAD INDEX INTO CACHE Statement”](#_bookmark21) .

**13.7.8.3** **FLUSH** **Statement**

FLUSH [NO\_WRITE\_TO\_BINLOG | LOCAL] {

*flush\_option* [, *flush\_option*] ...

| *tables\_option*

}

*flush\_option*: {

BINARY LOGS

|

ENGINE LOGS

ERROR LOGS

GENERAL LOGS

HOSTS

LOGS

PRIVILEGES

OPTIMIZER\_COSTS

|

|

|

|

|

|

| RELAY LOGS [FOR CHANNEL *channel*]

| SLOW LOGS

| STATUS

| USER\_RESOURCES

}

*tables\_option*: {

TABLES

| TABLES *tbl\_name* [, *tbl\_name*] ...

| TABLES WITH READ LOCK

| TABLES *tbl\_name* [, *tbl\_name*] ... WITH READ LOCK

| TABLES *tbl\_name* [, *tbl\_name*] ... FOR EXPORT

}

The [FLUSH](#_bookmark22) statement has several variant forms that clear or reload various internal caches, flush tables, or acquire locks. Each [FLUSH](#_bookmark22) operation requires the privileges indicated in its description.

**Note**

It is not possible to issue [FLUSH](#_bookmark22) statements within stored functions or triggers. However, you may use [FLUSH](#_bookmark22) in stored procedures, so long as these are not called from stored functions or triggers. See Section 25.8, “Restrictions on Stored Programs” .

By default, the server writes [FLUSH](#_bookmark22) statements to the binary log so that they replicate to replicas. To suppress logging, specify the optional NO\_WRITE\_TO\_BINLOG keyword or its alias LOCAL.

**Note**

[FLUSH LOGS](#_bookmark23), [FLUSH BINARY LOGS](#_bookmark24), [FLUSH TABLES WITH READ LOCK](#_bookmark25) (with or without a table list), and [FLUSH TABLES *tbl\_name* ... FOR](#_bookmark26) [EXPORT](#_bookmark26) are not written to the binary log in any case because they would cause problems if replicated to a replica.

The [FLUSH](#_bookmark22) statement causes an implicit commit. See Section 13.3.3, “Statements That Cause an Implicit Commit” .



The mysqladmin utility provides a command-line interface to some flush operations, using commands such as flush-hosts, flush-logs, flush-privileges, flush-status, and flush-tables. See Section 4.5.2, “mysqladmin — A MySQL Server Administration Program” .

Sending a SIGHUP or SIGUSR1 signal to the server causes several flush operations to occur that are similar to various forms of the [FLUSH](#_bookmark22) statement. Signals can be sent by the root system account or the system account that owns the server process. This enables the flush operations to be performed without having to connect to the server, which requires a MySQL account that has privileges sufficient for those operations. See Section 4.10, “Unix Signal Handling in MySQL” .

The [RESET](#_bookmark27) statement is similar to [FLUSH](#_bookmark22). See [Section 13.7.8.6, “RESET Statement”](#_bookmark27) , for information about using [RESET](#_bookmark27) with replication.

The following list describes the permitted [FLUSH](#_bookmark22) statement *flush\_option* values. For descriptions of the permitted *tables\_option* values, see [FLUSH TABLES Syntax](#_bookmark28).

• [FLUSH BINARY LOGS](#_bookmark24)

Closes and reopens any binary log file to which the server is writing. If binary logging is enabled, the sequence number of the binary log file is incremented by one relative to the previous file.

This operation requires the RELOAD privilege.

• [FLUSH ENGINE LOGS](#_bookmark29)

Closes and reopens any flushable logs for installed storage engines. This causes InnoDB to flush its logs to disk.

This operation requires the RELOAD privilege.

• [FLUSH ERROR LOGS](#_bookmark30)

Closes and reopens any error log file to which the server is writing.

This operation requires the RELOAD privilege.

• [FLUSH GENERAL LOGS](#_bookmark31)

Closes and reopens any general query log file to which the server is writing.

This operation requires the RELOAD privilege.

This operation has no effect on tables used for the general query log (see Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations” ).

• [FLUSH HOSTS](#_bookmark32)

Empties the host cache and the Performance Schema host\_cache table that exposes the cache

contents, and unblocks any blocked hosts.

This operation requires the RELOAD privilege.

For information about why host cache flushing might be advisable or desirable, see Section 5.1.12.3, “DNS Lookups and the Host Cache” .

**Note**

[FLUSH HOSTS](#_bookmark32) is deprecated as of MySQL 8.0.23; expect it to be removed in a future MySQL release. Instead, truncate the Performance Schema host\_cache table:

TRUNCATE TABLE performance\_schema.host\_cache;



The TRUNCATE TABLE operation requires the DROP privilege for the table rather than the RELOAD privilege. You should be aware that the TRUNCATE TABLE statement is not written to the binary log. To obtain the same behavior from FLUSH HOSTS, specify NO\_WRITE\_TO\_BINLOG or LOCAL as part of the statement.

• [FLUSH LOGS](#_bookmark23)

Closes and reopens any log file to which the server is writing.

This operation requires the RELOAD privilege.

The effect of this operation is equivalent to the combined effects of these operations:

|  |  |
| --- | --- |
| FLUSH  FLUSH  FLUSH  FLUSH  FLUSH  FLUSH | BINARY LOGS  ENGINE LOGS  ERROR LOGS  GENERAL LOGS  RELAY LOGS  SLOW LOGS |

• [FLUSH OPTIMIZER\_COSTS](#_bookmark33)

Re-reads the cost model tables so that the optimizer starts using the current cost estimates stored in them.

This operation requires the FLUSH\_OPTIMIZER\_COSTS or RELOAD privilege.

The server writes a warning to the error log for any unrecognized cost model table entries. For information about these tables, see Section 8.9.5, “The Optimizer Cost Model” . This operation affects only sessions that begin subsequent to the flush. Existing sessions continue to use the cost estimates that were current when they began.

• [FLUSH PRIVILEGES](#_bookmark34)

Re-reads the privileges from the grant tables in the mysql system schema. As part of this operation, the server reads the global\_grants table containing dynamic privilege assignments and registers any unregistered privileges found there.

Reloading the grant tables is necessary to enable updates to MySQL privileges and users only if you make such changes directly to the grant tables; it is not needed for account management statements such as GRANT or REVOKE, which take effect immediately. See Section 6.2.13, “When Privilege Changes Take Effect” , for more information.

This operation requires the RELOAD privilege.

If the --skip-grant-tables option was specified at server startup to disable the MySQL privilege system, [FLUSH PRIVILEGES](#_bookmark34) provides a way to enable the privilege system at runtime.

Resets failed-login tracking (or enables it if the server was started with --skip-grant-tables) and unlocks any temporarily locked accounts. See Section 6.2.15, “Password Management” .

Frees memory cached by the server as a result of GRANT, CREATE USER, CREATE SERVER, and INSTALL PLUGIN statements. This memory is not released by the corresponding REVOKE, DROP USER, DROP SERVER, and UNINSTALL PLUGIN statements, so for a server that executes many instances of the statements that cause caching, there is an increase in cached memory use unless it is freed with [FLUSH PRIVILEGES](#_bookmark34).

Clears the in-memory cache used by the caching\_sha2\_password authentication plugin. See Cache Operation for SHA-2 Pluggable Authentication.

• [FLUSH RELAY LOGS [FOR CHANNEL *channel*]](#_bookmark35)



Closes and reopens any relay log file to which the server is writing. If relay logging is enabled, the

sequence number of the relay log file is incremented by one relative to the previous file. This operation requires the RELOAD privilege.

The FOR CHANNEL *channel* clause enables you to name which replication channel the operation applies to. Execute [FLUSH RELAY LOGS FOR CHANNEL *channel*](#_bookmark35) to flush the relay log for a specific replication channel. If no channel is named and no extra replication channels exist, the operation applies to the default channel. If no channel is named and multiple replication channels exist, the operation applies to all replication channels. For more information, see Section 17.2.2, “Replication Channels” .

• [FLUSH SLOW LOGS](#_bookmark36)

Closes and reopens any slow query log file to which the server is writing.

This operation requires the RELOAD privilege.

This operation has no effect on tables used for the slow query log (see Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations” ).

• [FLUSH STATUS](#_bookmark37)

Flushes status indicators.

This operation adds the current thread's session status variable values to the global values and resets the session values to zero. Some global variables may be reset to zero as well. It also resets the counters for key caches (default and named) to zero and sets Max\_used\_connections to the current number of open connections. This information may be of use when debugging a query. See Section 1.5, “How to Report Bugs or Problems” .

FLUSH STATUS is unaffected by read\_only or super\_read\_only, and is always written to the binary log.

This operation requires the FLUSH\_STATUS or RELOAD privilege.

• [FLUSH USER\_RESOURCES](#_bookmark38)

Resets all per-hour user resource indicators to zero.

This operation requires the FLUSH\_USER\_RESOURCES or RELOAD privilege.

Resetting resource indicators enables clients that have reached their hourly connection, query, or update limits to resume activity immediately. [FLUSH USER\_RESOURCES](#_bookmark38) does not apply to the limit on maximum simultaneous connections that is controlled by the max\_user\_connections system variable. See Section 6.2.21, “Setting Account Resource Limits” .

**FLUSH** **TABLES** **Syntax**

[FLUSH TABLES](#_bookmark39) flushes tables, and, depending on the variant used, acquires locks. Any TABLES variant used in a [FLUSH](#_bookmark22) statement must be the only option used. [FLUSH TABLE](#_bookmark39) is a synonym for [FLUSH TABLES](#_bookmark39).

**Note**

The descriptions here that indicate tables are flushed by closing them apply differently for InnoDB, which flushes table contents to disk but leaves them open. This still permits table files to be copied while the tables are open, as long as other activity does not modify them.

• [FLUSH TABLES](#_bookmark39)

Closes all open tables, forces all tables in use to be closed, and flushes the prepared statement cache.

This operation requires the FLUSH\_TABLES or RELOAD privilege.

For information about prepared statement caching, see Section 8.10.3, “Caching of Prepared Statements and Stored Programs” .

[FLUSH TABLES](#_bookmark39) is not permitted when there is an active LOCK TABLES ... READ. To flush and lock tables, use [FLUSH TABLES *tbl\_name* ... WITH READ LOCK](#_bookmark40) instead.

• [FLUSH TABLES *tbl\_name* [, *tbl\_name*] ...](#_bookmark41)

With a list of one or more comma-separated table names, this operation is like [FLUSH TABLES](#_bookmark39) with no names except that the server flushes only the named tables. If a named table does not exist, no error occurs.

This operation requires the FLUSH\_TABLES or RELOAD privilege.

• [FLUSH TABLES WITH READ LOCK](#_bookmark25)

Closes all open tables and locks all tables for all databases with a global read lock. This operation requires the FLUSH\_TABLES or RELOAD privilege.

This operation is a very convenient way to get backups if you have a file system such as Veritas or ZFS that can take snapshots in time. Use UNLOCK TABLES to release the lock.

[FLUSH TABLES WITH READ LOCK](#_bookmark25) acquires a global read lock rather than table locks, so it is not subject to the same behavior as LOCK TABLES and UNLOCK TABLES with respect to table locking and implicit commits:

• UNLOCK TABLES implicitly commits any active transaction only if any tables currently have been locked with LOCK TABLES. The commit does not occur for UNLOCK TABLES following [FLUSH](#_bookmark25) [TABLES WITH READ LOCK](#_bookmark25) because the latter statement does not acquire table locks.

• Beginning a transaction causes table locks acquired with LOCK TABLES to be released, as though you had executed UNLOCK TABLES. Beginning a transaction does not release a global read lock acquired with [FLUSH TABLES WITH READ LOCK](#_bookmark25).

[FLUSH TABLES WITH READ LOCK](#_bookmark25) does not prevent the server from inserting rows into the log tables (see Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations” ).

• [FLUSH TABLES *tbl\_name* [, *tbl\_name*] ... WITH READ LOCK](#_bookmark40) Flushes and acquires read locks for the named tables.

This operation requires the FLUSH\_TABLES or RELOAD privilege. Because it acquires table locks, it also requires the LOCK TABLES privilege for each table.

The operation first acquires exclusive metadata locks for the tables, so it waits for transactions that have those tables open to complete. Then the operation flushes the tables from the table cache, reopens the tables, acquires table locks (like LOCK TABLES ... READ), and downgrades the

metadata locks from exclusive to shared. After the operation acquires locks and downgrades the metadata locks, other sessions can read but not modify the tables.

This operation applies only to existing base (non-TEMPORARY) tables. If a name refers to a base table, that table is used. If it refers to a TEMPORARY table, it is ignored. If a name applies to a view, an [ER\_WRONG\_OBJECT](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_wrong_object) error occurs. Otherwise, an [ER\_NO\_SUCH\_TABLE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_no_such_table) error occurs.

Use UNLOCK TABLES to release the locks, LOCK TABLES to release the locks and acquire other locks, or START TRANSACTION to release the locks and begin a new transaction.

This [FLUSH TABLES](#_bookmark39) variant enables tables to be flushed and locked in a single operation. It provides a workaround for the restriction that [FLUSH TABLES](#_bookmark39) is not permitted when there is an active LOCK TABLES ... READ.

This operation does not perform an implicit UNLOCK TABLES, so an error results if you perform the operation while there is any active LOCK TABLES or use it a second time without first releasing the locks acquired.

If a flushed table was opened with HANDLER, the handler is implicitly flushed and loses its position.

• [FLUSH TABLES *tbl\_name* [, *tbl\_name*] ... FOR EXPORT](#_bookmark26)

This [FLUSH TABLES](#_bookmark39) variant applies to InnoDB tables. It ensures that changes to the named tables have been flushed to disk so that binary table copies can be made while the server is running.

This operation requires the FLUSH\_TABLES or RELOAD privilege. Because it acquires locks on tables in preparation for exporting them, it also requires the LOCK TABLES and SELECT privileges for each table.

The operation works like this:

1. It acquires shared metadata locks for the named tables. The operation blocks as long as other sessions have active transactions that have modified those tables or hold table locks for them. When the locks have been acquired, the operation blocks transactions that attempt to update the tables, while permitting read-only operations to continue.

2. It checks whether all storage engines for the tables support FOR EXPORT. If any do not, an [ER\_ILLEGAL\_HA](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_illegal_ha) error occurs and the operation fails.

3. The operation notifies the storage engine for each table to make the table ready for export. The storage engine must ensure that any pending changes are written to disk.

4. The operation puts the session in lock-tables mode so that the metadata locks acquired earlier are not released when the FOR EXPORT operation completes.

This operation applies only to existing base (non-TEMPORARY) tables. If a name refers to a base table, that table is used. If it refers to a TEMPORARY table, it is ignored. If a name applies to a view, an [ER\_WRONG\_OBJECT](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_wrong_object) error occurs. Otherwise, an [ER\_NO\_SUCH\_TABLE](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_no_such_table) error occurs.

InnoDB supports FOR EXPORT for tables that have their own .ibd file file (that is, tables created with the innodb\_file\_per\_table setting enabled). InnoDB ensures when notified by the FOR EXPORT operation that any changes have been flushed to disk. This permits a binary copy of

table contents to be made while the FOR EXPORT operation is in effect because the .ibd file is transaction consistent and can be copied while the server is running. FOR EXPORT does not apply to

InnoDB system tablespace files, or to InnoDB tables that have FULLTEXT indexes. [FLUSH TABLES ...FOR EXPORT](#_bookmark26) is supported for partitioned InnoDB tables.

When notified by FOR EXPORT, InnoDB writes to disk certain kinds of data that is normally held in memory or in separate disk buffers outside the tablespace files. For each table, InnoDB also

produces a file named *table\_name*.cfg in the same database directory as the table. The .cfg file contains metadata needed to reimport the tablespace files later, into the same or different server.

When the FOR EXPORT operation completes, InnoDB has flushed all dirty pages to the table data files. Any change buffer entries are merged prior to flushing. At this point, the tables are locked and quiescent: The tables are in a transactionally consistent state on disk and you can copy the .ibd tablespace files along with the corresponding .cfg files to get a consistent snapshot of those tables.

For the procedure to reimport the copied table data into a MySQL instance, see Section 15.6.1.3, “Importing InnoDB Tables” .

After you are done with the tables, use UNLOCK TABLES to release the locks, LOCK TABLES to release the locks and acquire other locks, or START TRANSACTION to release the locks and begin a new transaction.

While any of these statements is in effect within the session, attempts to use [FLUSH TABLES ...](#_bookmark26) [FOR EXPORT](#_bookmark26) produce an error:

FLUSH TABLES ... WITH READ LOCK

FLUSH TABLES ... FOR EXPORT

LOCK TABLES ... READ

LOCK TABLES ... WRITE

While [FLUSH TABLES ... FOR EXPORT](#_bookmark26) is in effect within the session, attempts to use any of these statements produce an error:

FLUSH TABLES WITH READ LOCK

FLUSH TABLES ... WITH READ LOCK

FLUSH TABLES ... FOR EXPORT

**13.7.8.4** **KILL** **Statement**

KILL [CONNECTION | QUERY] *processlist\_id*

Each connection to mysqld runs in a separate thread. You can kill a thread with the KILL *processlist\_id* statement.

Thread processlist identifiers can be determined from the ID column of the INFORMATION\_SCHEMA PROCESSLIST table, the Id column of [SHOW PROCESSLIST](#_bookmark1) output, and the PROCESSLIST\_ID column of the Performance Schema threads table. The value for the current thread is returned by the CONNECTION\_ID() function.

[KILL](#_bookmark2) permits an optional CONNECTION or QUERY modifier:

• [KILL CONNECTION](#_bookmark2) is the same as [KILL](#_bookmark2) with no modifier: It terminates the connection associated with the given *processlist\_id*, after terminating any statement the connection is executing.

• [KILL QUERY](#_bookmark2) terminates the statement the connection is currently executing, but leaves the connection itself intact.

The ability to see which threads are available to be killed depends on the PROCESS privilege:

• Without PROCESS, you can see only your own threads.

• With PROCESS, you can see all threads.

The ability to kill threads and statements depends on the CONNECTION\_ADMIN privilege and the deprecated SUPER privilege:

• Without CONNECTION\_ADMIN or SUPER, you can kill only your own threads and statements.

• With CONNECTION\_ADMIN or SUPER, you can kill all threads and statements, except that to affect a thread or statement that is executing with the SYSTEM\_USER privilege, your own session must additionally have the SYSTEM\_USER privilege.



You can also use the mysqladmin processlist and mysqladmin kill commands to examine and kill threads.

When you use [KILL](#_bookmark2), a thread-specific kill flag is set for the thread. In most cases, it might take some time for the thread to die because the kill flag is checked only at specific intervals:

• During SELECT operations, for ORDER BY and GROUP BY loops, the flag is checked after reading a block of rows. If the kill flag is set, the statement is aborted.

• ALTER TABLE operations that make a table copy check the kill flag periodically for each few copied rows read from the original table. If the kill flag was set, the statement is aborted and the temporary table is deleted.

The [KILL](#_bookmark2) statement returns without waiting for confirmation, but the kill flag check aborts the operation within a reasonably small amount of time. Aborting the operation to perform any necessary cleanup also takes some time.

• During UPDATE or DELETE operations, the kill flag is checked after each block read and after each updated or deleted row. If the kill flag is set, the statement is aborted. If you are not using transactions, the changes are not rolled back.

• GET\_LOCK() aborts and returns NULL.

• If the thread is in the table lock handler (state: Locked), the table lock is quickly aborted.

• If the thread is waiting for free disk space in a write call, the write is aborted with a “disk full” error message.

• [EXPLAIN ANALYZE](#_bookmark42) aborts and prints the first row of output. This works in MySQL 8.0.20 and later.

**Warning**

Killing a REPAIR TABLE or OPTIMIZE TABLE operation on a MyISAM table results in a table that is corrupted and unusable. Any reads or writes to such a table fail until you optimize or repair it again (without interruption).

**13.7.8.5** **LOAD** **INDEX** **INTO** **CACHE** **Statement**

LOAD INDEX INTO CACHE

*tbl\_index\_list* [, *tbl\_index\_list*] ...

*tbl\_index\_list*:

*tbl\_name*

[PARTITION (*partition\_list*)]

[{INDEX |KEY} (*index\_name* [, *index\_name*] ...)]

[IGNORE LEAVES]

*partition\_list*: {

*partition\_name* [, *partition\_name*] ...

| ALL

}

The [LOAD INDEX INTO CACHE](#_bookmark21) statement preloads a table index into the key cache to which it has been assigned by an explicit [CACHE INDEX](#_bookmark20) statement, or into the default key cache otherwise.

[LOAD INDEX INTO CACHE](#_bookmark21) applies only to MyISAM tables, including partitioned MyISAM tables. In addition, indexes on partitioned tables can be preloaded for one, several, or all partitions.

The IGNORE LEAVES modifier causes only blocks for the nonleaf nodes of the index to be preloaded. IGNORE LEAVES is also supported for partitioned MyISAM tables.

The following statement preloads nodes (index blocks) of indexes for the tables t1 and t2:

mysql> **LOAD** **INDEX** **INTO** **CACHE** **t1,** **t2** **IGNORE** **LEAVES;**

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

| Table | Op | Msg\_type | Msg\_text |

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

|

| test.t1 | preload\_keys | status

| test.t2 | preload\_keys | status

|

|

OK

OK

|

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

This statement preloads all index blocks from t1. It preloads only blocks for the nonleaf nodes from t2.

The syntax of [LOAD INDEX INTO CACHE](#_bookmark21) enables you to specify that only particular indexes from a table should be preloaded. However, the implementation preloads all the table's indexes into the cache, so there is no reason to specify anything other than the table name.

It is possible to preload indexes on specific partitions of partitioned MyISAM tables. For example, of the following 2 statements, the first preloads indexes for partition p0 of a partitioned table pt, while the second preloads the indexes for partitions p1 and p3 of the same table:

LOAD INDEX INTO CACHE pt PARTITION (p0);

LOAD INDEX INTO CACHE pt PARTITION (p1, p3);

To preload the indexes for all partitions in table pt, you can use either of the following two statements: LOAD INDEX INTO CACHE pt PARTITION (ALL);

LOAD INDEX INTO CACHE pt;

The two statements just shown are equivalent, and issuing either one has exactly the same effect. In other words, if you wish to preload indexes for all partitions of a partitioned table, the PARTITION (ALL) clause is optional.

When preloading indexes for multiple partitions, the partitions need not be contiguous, and you need not list their names in any particular order.

[LOAD INDEX INTO CACHE ... IGNORE LEAVES](#_bookmark21) fails unless all indexes in a table have the same block size. To determine index block sizes for a table, use myisamchk -dv and check the Blocksize column.

**13.7.8.6** **RESET** **Statement**

RESET *reset\_option* [, *reset\_option*] ...

*reset\_option*: {

MASTER

| REPLICA

| SLAVE

}

The [RESET](#_bookmark27) statement is used to clear the state of various server operations. You must have the RELOAD privilege to execute [RESET](#_bookmark27).

For information about the [RESET PERSIST](#_bookmark43) statement that removes persisted global system variables, see [Section 13.7.8.7, “RESET PERSIST Statement”](#_bookmark43) .

[RESET](#_bookmark27) acts as a stronger version of the [FLUSH](#_bookmark22) statement. See [Section 13.7.8.3, “FLUSH Statement”](#_bookmark22) .

The [RESET](#_bookmark27) statement causes an implicit commit. See Section 13.3.3, “Statements That Cause an Implicit Commit” .

The following list describes the permitted [RESET](#_bookmark27) statement *reset\_option* values:

• RESET MASTER

Deletes all binary logs listed in the index file, resets the binary log index file to be empty, and creates a new binary log file.

• RESET REPLICA

Makes the replica forget its replication position in the source binary logs. Also resets the relay log by deleting any existing relay log files and beginning a new one. Use RESET REPLICA in place of RESET SLAVE from MySQL 8.0.22.

**13.7.8.7** **RESET** **PERSIST** **Statement**

RESET PERSIST [[IF EXISTS] *system\_var\_name*]

[RESET PERSIST](#_bookmark43) removes persisted global system variable settings from the mysqld-auto.cnf option file in the data directory. Removing a persisted system variable causes the variable no longer to be initialized from mysqld-auto.cnf at server startup. For more information about persisting system variables and the mysqld-auto.cnf file, see Section 5.1.9.3, “Persisted System Variables” .

Prior to MySQL 8.0.32, this statement did not work with variables whose name contained a dot character ( .), such as MyISAM multiple key cache variables and variables registered by components. (Bug #33417357)

The privileges required for [RESET PERSIST](#_bookmark43) depend on the type of system variable to be removed:

• For dynamic system variables, this statement requires the SYSTEM\_VARIABLES\_ADMIN privilege (or the deprecated SUPER privilege).

• For read-only system variables, this statement requires the SYSTEM\_VARIABLES\_ADMIN and PERSIST\_RO\_VARIABLES\_ADMIN privileges.

See Section 5.1.9.1, “System Variable Privileges” .

Depending on whether the variable name and IF EXISTS clauses are present, the [RESET PERSIST](#_bookmark43) statement has these forms:

• To remove all persisted variables from mysqld-auto.cnf, use [RESET PERSIST](#_bookmark43) without naming any system variable:

RESET PERSIST;

You must have privileges for removing both dynamic and read-only system variables if mysqld- auto.cnf contains both kinds of variables.

• To remove a specific persisted variable from mysqld-auto.cnf, name it in the statement: RESET PERSIST *system\_var\_name*;

This includes plugin system variables, even if the plugin is not currently installed. If the variable is not present in the file, an error occurs.

• To remove a specific persisted variable from mysqld-auto.cnf, but produce a warning rather than an error if the variable is not present in the file, add an IF EXISTS clause to the previous syntax:

RESET PERSIST IF EXISTS *system\_var\_name*;

[RESET PERSIST](#_bookmark43) is not affected by the value of the persisted\_globals\_load system variable.

[RESET PERSIST](#_bookmark43) affects the contents of the Performance Schema persisted\_variables table because the table contents correspond to the contents of the mysqld-auto.cnf file. On the other hand, because [RESET PERSIST](#_bookmark43) does not change variable values, it has no effect on the contents of the Performance Schema variables\_info table until the server is restarted.

For information about [RESET](#_bookmark27) statement variants that clear the state of other server operations, see [Section 13.7.8.6, “RESET Statement”](#_bookmark27) .

**13.7.8.8** **RESTART** **Statement**

RESTART



This statement stops and restarts the MySQL server. It requires the SHUTDOWN privilege.

One use for [RESTART](#_bookmark44) is when it is not possible or convenient to gain command-line access to the MySQL server on the server host to restart it. For example, SET PERSIST\_ONLY can be used at runtime to make configuration changes to system variables that can be set only at server startup, but the server must still be restarted for those changes to take effect. The [RESTART](#_bookmark44) statement provides a way to do so from within client sessions, without requiring command-line access on the server host.

**Note**

After executing a [RESTART](#_bookmark44) statement, the client can expect the current connection to be lost. If auto-reconnect is enabled, the connection is reestablished after the server restarts. Otherwise, the connection must be reestablished manually.

A successful [RESTART](#_bookmark44) operation requires mysqld to be running in an environment that has a monitoring process available to detect a server shutdown performed for restart purposes:

• In the presence of a monitoring process, [RESTART](#_bookmark44) causes mysqld to terminate such that the monitoring process can determine that it should start a new mysqld instance.

• If no monitoring process is present, [RESTART](#_bookmark44) fails with an error.

These platforms provide the necessary monitoring support for the [RESTART](#_bookmark44) statement:

• Windows, when mysqld is started as a Windows service or standalone. (mysqld forks, and one process acts as a monitor to the other, which acts as the server.)

• Unix and Unix-like systems that use systemd or mysqld\_safe to manage mysqld. To configure a monitoring environment such that mysqld enables the [RESTART](#_bookmark44) statement:

1. Set the MYSQLD\_PARENT\_PID environment variable to the value of the process ID of the process that starts mysqld, before starting mysqld.

2. When mysqld performs a shutdown due to use of the [RESTART](#_bookmark44) statement, it returns exit code 16.

3. When the monitoring process detects an exit code of 16, it starts mysqld again. Otherwise, it exits.

Here is a minimal example as implemented in the bash shell:

#!/bin/bash

export MYSQLD\_PARENT\_PID=$$

export MYSQLD\_RESTART\_EXIT=16

while true ; do

bin/mysqld *mysqld* *options* *here*

if [ $? -ne $MYSQLD\_RESTART\_EXIT ]; then

break

fi

done

On Windows, the forking used to implement [RESTART](#_bookmark44) makes determining the server process to attach to for debugging more difficult. To alleviate this, starting the server with --gdb suppresses forking, in addition to its other actions done to set up a debugging environment. In non-debug settings, --no- monitor may be used for the sole purpose of suppressing forking the monitor process. For a server started with either --gdb or --no-monitor, executing [RESTART](#_bookmark44) causes the server to simply exit without restarting.

The Com\_restart status variable tracks the number of [RESTART](#_bookmark44) statements. Because status variables are initialized for each server startup and do not persist across restarts, Com\_restart normally has a value of zero, but can be nonzero if [RESTART](#_bookmark44) statements were executed but failed.

**13.7.8.9** **SHUTDOWN** **Statement**

SHUTDOWN

This statement stops the MySQL server. It requires the SHUTDOWN privilege.

[SHUTDOWN](#_bookmark45) provides an SQL-level interface to the same functionality available using the mysqladmin shutdown command or the [mysql\_shutdown()](https://dev.mysql.com/doc/c-api/8.0/en/mysql-shutdown.html) C API function. A successful [SHUTDOWN](#_bookmark45) sequence consists of checking the privileges, validating the arguments, and sending an OK packet to the client. Then the server is shut down.

The Com\_shutdown status variable tracks the number of [SHUTDOWN](#_bookmark45) statements. Because status variables are initialized for each server startup and do not persist across restarts, Com\_shutdown normally has a value of zero, but can be nonzero if [SHUTDOWN](#_bookmark45) statements were executed but failed.

Another way to stop the server is to send it a SIGTERM signal, which can be done by root or the account that owns the server process. SIGTERM enables server shutdown to be performed without having to connect to the server. See Section 4.10, “Unix Signal Handling in MySQL” .

**13.8** **Utility** **Statements**

**13.8.1** **DESCRIBE** **Statement**

The [DESCRIBE](#_bookmark46) and [EXPLAIN](#_bookmark16) statements are synonyms, used either to obtain information about table structure or query execution plans. For more information, see Section 13.7.7.5, “SHOW COLUMNS Statement” , and [Section 13.8.2, “EXPLAIN Statement”](#_bookmark16) .

**13.8.2** **EXPLAIN** **Statement**

{EXPLAIN | DESCRIBE | DESC}

*tbl\_name* [*col\_name* | *wild*]

{EXPLAIN | DESCRIBE | DESC}

[*explain\_type*]

{*explainable\_stmt* | FOR CONNECTION *connection\_id*}

{EXPLAIN | DESCRIBE | DESC} ANALYZE [FORMAT = TREE] *select\_statement*

*explain\_type*: {

FORMAT = *format\_name*

}

*format\_name*: {

TRADITIONAL

| JSON

| TREE

}

*explainable\_stmt*: {

SELECT statement

| TABLE statement

| DELETE statement

| INSERT statement

| REPLACE statement

| UPDATE statement

}

The [DESCRIBE](#_bookmark46) and [EXPLAIN](#_bookmark16) statements are synonyms. In practice, the [DESCRIBE](#_bookmark46) keyword is more often used to obtain information about table structure, whereas [EXPLAIN](#_bookmark16) is used to obtain a query execution plan (that is, an explanation of how MySQL would execute a query).

The following discussion uses the [DESCRIBE](#_bookmark46) and [EXPLAIN](#_bookmark16) keywords in accordance with those uses, but the MySQL parser treats them as completely synonymous.

• [Obtaining Table Structure Information](#_bookmark47)

• [Obtaining Execution Plan Information](#_bookmark48)

• [Obtaining Information with EXPLAIN ANALYZE](#_bookmark42)

**Obtaining** **Table** **Structure** **Information**

[DESCRIBE](#_bookmark46) provides information about the columns in a table:

mysql> **DESCRIBE** **City;**

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

| Field | Type | Null | Key | Default | Extra |

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

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| | Id | int(11) | NO  | Name | char(35) | NO | |  | | PRI | |  | | NULL | |  | | auto\_increment | |  | |
| | Country | char(3) | NO | | | UNI | | |  | | |  | | |
| | District | char(20) | YES | | | MUL | | |  | | |  | | |
| | Population | int(11) | NO | | |  | | | 0 | | |  | | |

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

[DESCRIBE](#_bookmark46) is a shortcut for SHOW COLUMNS. These statements also display information for views. The description for SHOW COLUMNS provides more information about the output columns. See Section 13.7.7.5, “SHOW COLUMNS Statement” .

By default, [DESCRIBE](#_bookmark46) displays information about all columns in the table. *col\_name*, if given, is the name of a column in the table. In this case, the statement displays information only for the named column. *wild*, if given, is a pattern string. It can contain the SQL % and \_ wildcard characters. In this case, the statement displays output only for the columns with names matching the string. There is no need to enclose the string within quotation marks unless it contains spaces or other special characters.

The [DESCRIBE](#_bookmark46) statement is provided for compatibility with Oracle.

The SHOW CREATE TABLE, [SHOW TABLE STATUS](#_bookmark11), and SHOW INDEX statements also provide information about tables. See Section 13.7.7, “SHOW Statements” .

The explain\_format system variable, added in MySQL 8.0.32, has no effect on the output of EXPLAIN when used to obtain information about table columns.

**Obtaining** **Execution** **Plan** **Information**

The [EXPLAIN](#_bookmark16) statement provides information about how MySQL executes statements:

• [EXPLAIN](#_bookmark16) works with SELECT, DELETE, INSERT, REPLACE, and UPDATE statements. In MySQL 8.0.19 and later, it also works with TABLE statements.

• When [EXPLAIN](#_bookmark16) is used with an explainable statement, MySQL displays information from the optimizer about the statement execution plan. That is, MySQL explains how it would process the statement, including information about how tables are joined and in which order. For information about using [EXPLAIN](#_bookmark16) to obtain execution plan information, see Section 8.8.2, “EXPLAIN Output Format” .

• When [EXPLAIN](#_bookmark16) is used with FOR CONNECTION *connection\_id* rather than an explainable statement, it displays the execution plan for the statement executing in the named connection. See

Section 8.8.4, “Obtaining Execution Plan Information for a Named Connection” .

• For explainable statements, [EXPLAIN](#_bookmark16) produces additional execution plan information that can be displayed using [SHOW WARNINGS](#_bookmark15). See Section 8.8.3, “Extended EXPLAIN Output Format” .

• [EXPLAIN](#_bookmark16) is useful for examining queries involving partitioned tables. See Section 24.3.5, “Obtaining Information About Partitions” .

• The FORMAT option can be used to select the output format. TRADITIONAL presents the output in tabular format. This is the default if no FORMAT option is present. JSON format displays the

information in JSON format. In MySQL 8.0.16 and later, TREE provides tree-like output with more precise descriptions of query handling than the TRADITIONAL format; it is the only format which shows hash join usage (see Section 8.2.1.4, “Hash Join Optimization”) and is always used for

EXPLAIN ANALYZE.

As of MySQL 8.0.32, the default output format used by EXPLAIN (that is, when it has no FORMAT option) is determined by the value of the explain\_format system variable. The precise effects of this variable are described later in this section.

[EXPLAIN](#_bookmark16) requires the same privileges required to execute the explained statement. Additionally, [EXPLAIN](#_bookmark16) also requires the SHOW VIEW privilege for any explained view. [EXPLAIN ... FOR](#_bookmark16) [CONNECTION](#_bookmark16) also requires the PROCESS privilege if the specified connection belongs to a different user.

The explain\_format system variable introduced in MySQL 8.0.32 determines the format of the output from EXPLAIN when used to display a query execution plan. This variable can take any of the values used with the FORMAT option, with the addition of DEFAULT as a synonym for TRADITIONAL. The following example uses the country table from the world database which can be obtained from [MySQL: Other Downloads](https://dev.mysql.com/doc/index-other.html):

mysql> **USE** **world;** # Make world the current database

Database changed

Checking the value of explain\_format, we see that it has the default value, and that EXPLAIN (with no FORMAT option) therefore uses the traditional tabular output:

mysql> **SELECT** **@@explain\_format;**

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

| @@explain\_format |

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

| TRADITIONAL |

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

1 row in set (0.00 sec)

mysql> **EXPLAIN** **SELECT** **Name** **FROM** **country** **WHERE** **Code** **Like** **'A%';**

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

| id | select\_type | table | partitions | type | possible\_keys | key | key\_len | ref | rows | f

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

| 1 | SIMPLE | country | NULL | range | PRIMARY | PRIMARY | 12 | NULL | 17 |

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

1 row in set, 1 warning (0.00 sec)

If we set the value of explain\_format to TREE, then rerun the same EXPLAIN statement, the output uses the tree-like format:

mysql> **SET** **@@explain\_format=TREE;**

Query OK, 0 rows affected (0.00 sec)

mysql> **SELECT** **@@explain\_format;**

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

| @@explain\_format |

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

| TREE |

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

1 row in set (0.00 sec)

mysql> **EXPLAIN** **SELECT** **Name** **FROM** **country** **WHERE** **Code** **LIKE** **'A%';**

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

| EXPLAIN

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

| -> Filter: (country . `Code` like 'A%') (cost=3 .67 rows=17)

-> Index range scan on country using PRIMARY over ('A' <= Code <= 'A????????') (cost=3 .67 rows=17)

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

1 row in set, 1 warning (0.00 sec)

As stated previously, the FORMAT option overrides this setting. Executing the same EXPLAIN statement using FORMAT=JSON instead of FORMAT=TREE shows that this is the case:

mysql> **EXPLAIN** **FORMAT=JSON** **SELECT** **Name** **FROM** **country** **WHERE** **Code** **LIKE** **'A%';**

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

| EXPLAIN |

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

| {

"query\_block": {

"select\_id": 1,

"cost\_info": {

"query\_cost": "3 .67"

},

"table": {

"table\_name": "country",

"access\_type": "range",

"possible\_keys": [

"PRIMARY"

],

"key": "PRIMARY",

"used\_key\_parts": [

"Code"

],

"key\_length": "12",

"rows\_examined\_per\_scan": 17,

"rows\_produced\_per\_join": 17,

"filtered": "100 .00",

"cost\_info": {

"read\_cost": "1 .97",

"eval\_cost": "1 .70",

"prefix\_cost": "3 .67",

"data\_read\_per\_join": "16K"

},

"used\_columns": [

"Code",

"Name"

],

"attached\_condition": "(`world` . `country` . `Code` like 'A%')"

}

}

} |

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

1 row in set, 1 warning (0.00 sec)

To return the default output of EXPLAIN to the tabular format, set explain\_format to TRADITIONAL. Alternatively, you can set it to DEFAULT, which has the same effect, as shown here:

mysql> **SET** **@@explain\_format=DEFAULT;**

Query OK, 0 rows affected (0.00 sec)

mysql> **SELECT** **@@explain\_format;**

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

| @@explain\_format |

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

| TRADITIONAL |

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

1 row in set (0.00 sec)

With the help of [EXPLAIN](#_bookmark16), you can see where you should add indexes to tables so that the statement executes faster by using indexes to find rows. You can also use [EXPLAIN](#_bookmark16) to check whether the optimizer joins the tables in an optimal order. To give a hint to the optimizer to use a join order corresponding to the order in which the tables are named in a SELECT statement, begin the statement with SELECT STRAIGHT\_JOIN rather than just SELECT. (See Section 13.2.13, “SELECT Statement” .)

The optimizer trace may sometimes provide information complementary to that of [EXPLAIN](#_bookmark16). However, the optimizer trace format and content are subject to change between versions. For details, see [MySQL Internals: Tracing the Optimizer](https://dev.mysql.com/doc/internals/en/optimizer-tracing.html).

If you have a problem with indexes not being used when you believe that they should be, run ANALYZE TABLE to update table statistics, such as cardinality of keys, that can affect the choices the optimizer makes. See Section 13.7.3.1, “ANALYZE TABLE Statement” .



**Note**

MySQL Workbench has a Visual Explain capability that provides a visual representation of [EXPLAIN](#_bookmark16) output. See [Tutorial: Using Explain Improve](https://dev.mysql.com/doc/workbench/en/wb-tutorial-visual-explain-dbt3.html)to [Query Performance](https://dev.mysql.com/doc/workbench/en/wb-tutorial-visual-explain-dbt3.html).

**Obtaining** **Information** **with** **EXPLAIN** **ANALYZE**

MySQL 8.0.18 introduces EXPLAIN ANALYZE, which runs a statement and produces [EXPLAIN](#_bookmark48) output along with timing and additional, iterator-based, information about how the optimizer's expectations matched the actual execution. For each iterator, the following information is provided:

• Estimated execution cost

(Some iterators are not accounted for by the cost model, and so are not included in the estimate.)

• Estimated number of returned rows

• Time to return first row

• Time spent executing this iterator (including child iterators, but not parent iterators), in milliseconds. (When there are multiple loops, this figure shows the average time per loop.)

• Number of rows returned by the iterator

• Number of loops

The query execution information is displayed using the TREE output format, in which nodes represent iterators. EXPLAIN ANALYZE always uses the TREE output format. In MySQL 8.0.21 and later, this can optionally be specified explicitly using FORMAT=TREE; formats other than TREE remain unsupported.

EXPLAIN ANALYZE can be used with SELECT statements, as well as with multi-table UPDATE and DELETE statements. Beginning with MySQL 8.0.19, it can also be used with TABLE statements.

Beginning with MySQL 8.0.20, you can terminate this statement using [KILL QUERY](#_bookmark2) or **CTRL-C**. EXPLAIN ANALYZE cannot be used with FOR CONNECTION.

Example output:

mysql> **EXPLAIN** **ANALYZE** **SELECT** **\*** **FROM** **t1** **JOIN** **t2** **ON** **(t1** **.c1** **=** **t2** **.c2)\G**

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

EXPLAIN: -> Inner hash join (t2 .c2 = t1 .c1) (cost=4 .70 rows=6)

(actual time=0 .032 . .0 .035 rows=6 loops=1)

-> Table scan on t2 (cost=0 .06 rows=6)

(actual time=0 .003 . .0 .005 rows=6 loops=1)

-> Hash

-> Table scan on t1 (cost=0 .85 rows=6)

(actual time=0.018..0.022 rows=6 loops=1)

mysql> **EXPLAIN** **ANALYZE** **SELECT** **\*** **FROM** **t3** **WHERE** **i** **>** **8\G**

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

EXPLAIN: -> Filter: (t3 .i > 8) (cost=1 .75 rows=5)

(actual time=0 .019 . .0 .021 rows=6 loops=1)

-> Table scan on t3 (cost=1 .75 rows=15)

(actual time=0.017..0.019 rows=15 loops=1)

mysql> **EXPLAIN** **ANALYZE** **SELECT** **\*** **FROM** **t3** **WHERE** **pk** **>** **17\G**

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

|  |  |
| --- | --- |
| EXPLAIN: -> Filter: (t3 .pk > 17) (cost=1 .26  (actual time=0 .013 . .0 .016 rows=5 loops=1)  -> Index range scan on t3 using PRIMARY  (actual time=0.012..0.014 rows=5 loops=1) | rows=5)  (cost=1.26 rows=5) |

The tables used in the example output were created by the statements shown here:

CREATE TABLE t1 (

c1 INTEGER DEFAULT NULL,

c2 INTEGER DEFAULT NULL

);

CREATE TABLE t2 (

c1 INTEGER DEFAULT NULL,

c2 INTEGER DEFAULT NULL

);

CREATE TABLE t3 (

pk INTEGER NOT NULL PRIMARY KEY,

i INTEGER DEFAULT NULL

);

Values shown for actual time in the output of this statement are expressed in milliseconds.

As of MySQL 8.0.32, the explain\_format system variable has the following effects on EXPLAIN ANALYZE:

• If this value of this variable is TRADITIONAL or TREE, EXPLAIN ANALYZE uses the TREE format. This ensures that this statement continues to use the TREE format by default, as it did prior to the introduction of explain\_format.

• If the value of explain\_format is JSON, EXPLAIN ANALYZE returns an error unless FORMAT=TREE is specified as part of the statement. This is due to the fact that EXPLAIN ANALYZE supports only the TREE output format.

We illustrate the behavior described in the second point here, re-using the last EXPLAIN ANALYZE statement from the previous example:

mysql> **SET** **@@explain\_format=JSON;**

Query OK, 0 rows affected (0.00 sec)

mysql> **SELECT** **@@explain\_format;**

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

| @@explain\_format |

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

| JSON |

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

1 row in set (0.00 sec)

mysql> **EXPLAIN** **ANALYZE** **SELECT** **\*** **FROM** **t3** **WHERE** **pk** **>** **17\G**

ERROR 1235 (42000): This version of MySQL doesn't yet support 'EXPLAIN ANALYZE with JSON format'

mysql> **EXPLAIN** **ANALYZE** **FORMAT=TRADITIONAL** **SELECT** **\*** **FROM** **t3** **WHERE** **pk** **>** **17\G**

ERROR 1235 (42000): This version of MySQL doesn't yet support 'EXPLAIN ANALYZE with TRADITIONAL format'

mysql> **EXPLAIN** **ANALYZE** **FORMAT=TREE** **SELECT** **\*** **FROM** **t3** **WHERE** **pk** **>** **17\G**

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

EXPLAIN: -> Filter: (t3 .pk > 17) (cost=1 .26

rows=5)

(cost=1.26 rows=5)

(actual time=0 .013 . .0 .016 rows=5 loops=1)

-> Index range scan on t3 using PRIMARY

(actual time=0.012..0.014 rows=5 loops=1)

Using FORMAT=TRADITIONAL or FORMAT=JSON with EXPLAIN ANALYZE always raises an error, regardless of the value of explain\_format.

Beginning with MySQL 8.0.33, numbers in the output of EXPLAIN ANALYZE and EXPLAIN FORMAT=TREE are formatted according to the following rules:

• Numbers in the range 0.001-999999.5 are printed as decimal numbers.

Decimal numbers less than 1000 have three significant digits; the remainder have four, five, or six.

• Numbers outside the range 0.001-999999.5 are printed in engineering format. Examples of such values are 1.23e+9 and 934e-6.

• No trailing zeros are printed. For example, we print 2.3 rather than 2.30, and 1.2e+6 rather than 1.20e+6.

• Numbers less than 1e-12 are printed as 0.

**13.8.3** **HELP** **Statement**

HELP '*search\_string*'

The [HELP](#_bookmark49) statement returns online information from the MySQL Reference Manual. Its proper operation requires that the help tables in the mysql database be initialized with help topic information (see Section 5.1.17, “Server-Side Help Support”).

The [HELP](#_bookmark49) statement searches the help tables for the given search string and displays the result of the search. The search string is not case-sensitive.

The search string can contain the wildcard characters % and \_. These have the same meaning as for pattern-matching operations performed with the LIKE operator. For example, HELP 'rep%' returns a list of topics that begin with rep.

The HELP statement understands several types of search strings:

• At the most general level, use contents to retrieve a list of the top-level help categories: HELP 'contents'

• For a list of topics in a given help category, such as Data Types, use the category name: HELP 'data types'

• For help on a specific help topic, such as the ASCII() function or the CREATE TABLE statement, use the associated keyword or keywords:

HELP 'ascii'

HELP 'create table'

In other words, the search string matches a category, many topics, or a single topic. You cannot necessarily tell in advance whether a given search string returns a list of items or the help information for a single help topic. However, you can tell what kind of response [HELP](#_bookmark49) returned by examining the number of rows and columns in the result set.

The following descriptions indicate the forms that the result set can take. Output for the example statements is shown using the familiar “tabular” or “vertical” format that you see when using the mysql client, but note that mysql itself reformats [HELP](#_bookmark49) result sets in a different way.

• Empty result set

No match could be found for the search string.

• Result set containing a single row with three columns

This means that the search string yielded a hit for the help topic. The result has three columns:

• name: The topic name.

• description: Descriptive help text for the topic.

• example: Usage example or examples. This column might be blank. Example: HELP 'replace'

Yields:

name: REPLACE

description: Syntax:

REPLACE(str,from\_str,to\_str)

Returns the string str with all occurrences of the string from\_str

replaced by the string to\_str . REPLACE() performs a case-sensitive

match when searching for from\_str .

example: mysql> SELECT REPLACE('www .mysql .com', 'w', 'Ww');

-> 'WwWwWw.mysql.com'

• Result set containing multiple rows with two columns

This means that the search string matched many help topics. The result set indicates the help topic names:

• name: The help topic name.

• is\_it\_category: Y if the name represents a help category, N if it does not. If it does not, the name value when specified as the argument to the [HELP](#_bookmark49) statement should yield a single-row result set containing a description for the named item.

Example: HELP 'status'

Yields:

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

| name | is\_it\_category |

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  | | SHOW  SHOW  SHOW  SHOW  SHOW  SHOW  SHOW | ENGINE  MASTER STATUS  PROCEDURE STATUS  SLAVE STATUS  STATUS  TABLE STATUS | |  |  |  |  |  |  | | N  N  N  N  N  N  N | |  |  |  |  |  |  | |

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

• Result set containing multiple rows with three columns

This means the search string matches a category. The result set contains category entries:

• source\_category\_name: The help category name.

• name: The category or topic name

• is\_it\_category: Y if the name represents a help category, N if it does not. If it does not, the name value when specified as the argument to the [HELP](#_bookmark49) statement should yield a single-row result set containing a description for the named item.

Example: HELP 'functions'

Yields:

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

| source\_category\_name | name | is\_it\_category |

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| | | Functions | | | CREATE FUNCTION | | | N | | |
| | | Functions | | | DROP FUNCTION | | | N | | |
| | | Functions | | | Bit Functions | | | Y | | |
| | | Functions | | | Comparison operators | | | Y | | |
| | | Functions | | | Control flow functions | | | Y | | |
| | | Functions | | | Date and Time Functions | | | Y | | |
| | | Functions | | | Encryption Functions | | | Y | | |
| | | Functions | | | Information Functions | | | Y | | |
| | | Functions | | | Logical operators | | | Y | | |
| | | Functions | | | Miscellaneous Functions | | | Y | | |
| | | Functions | | | Numeric Functions | | | Y | | |

| Functions | String Functions | Y |

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

**13.8.4** **USE** **Statement**

USE *db\_name*

The [USE](#_bookmark50) statement tells MySQL to use the named database as the default (current) database for subsequent statements. This statement requires some privilege for the database or some object within it.

The named database remains the default until the end of the session or another [USE](#_bookmark50) statement is issued:

|  |  |
| --- | --- |
| USE db1;  SELECT COUNT(\*) FROM mytable;  USE db2;  SELECT COUNT(\*) FROM mytable; | # selects from db1.mytable  # selects from db2.mytable |

The database name must be specified on a single line. Newlines in database names are not supported.

Making a particular database the default by means of the [USE](#_bookmark50) statement does not preclude accessing tables in other databases. The following example accesses the author table from the db1 database and the editor table from the db2 database:

USE db1;

SELECT author\_name,editor\_name FROM author,db2.editor

WHERE author.editor\_id = db2.editor.editor\_id;