instrumented by the Performance Schema, this table misses some rows. In this case, the `Performance_schema_thread_instances_lost` status variable is greater than zero.

- `performance_schema_session_connect_attrs_size`

Command-Line Format	`--performance-schema-session-connect-attrs-size=#`
System Variable	`performance_schema_session_connect_attrs_si`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autosizing; do not assign this literal value)
Minimum Value	`-1` (signifies autosizing; do not assign this literal value)
Maximum Value	`1048576`
Unit	bytes

The amount of preallocated memory per thread reserved to hold connection attribute key-value pairs. If the aggregate size of connection attribute data sent by a client is larger than this amount, the Performance Schema truncates the attribute data, increments the `Performance_schema_session_connect_attrs_lost` status variable, and writes a message to the error log indicating that truncation occurred if the `log_error_verbosity` system variable is greater than 1. A `_truncated` attribute is also added to the session attributes with a value indicating how many bytes were lost, if the attribute buffer has sufficient space. This enables the Performance Schema to expose per-connection truncation information in the connection attribute tables. This information can be examined without having to check the error log.

The default value of `performance_schema_session_connect_attrs_size` is autosized at server startup. This value may be small, so if truncation occurs (`Performance_schema_session_connect_attrs_lost` becomes nonzero), you may wish to set `performance_schema_session_connect_attrs_size` explicitly to a larger value.

Although the maximum permitted `performance_schema_session_connect_attrs_size` value is 1MB, the effective maximum is 64KB because the server imposes a limit of 64KB on the aggregate size of connection attribute data it accepts. If a client attempts to send more than 64KB of attribute data, the server rejects the connection. For more information, see Section 27.12.9, “Performance Schema Connection Attribute Tables”.

- `performance_schema_setup_actors_size`

Command-Line Format	`--performance-schema-setup-actors-size=#`
System Variable	`performance_schema_setup_actors_size`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autoscaling; do not assign this literal value)

Minimum Value	-1 (signifies autosizing; do not assign this literal value)
Maximum Value	1048576

The number of rows in the setup_actors table.

- performance_schema_setup_objects_size

Command-Line Format	--performance-schema-setup-objects-size=#
System Variable	performance_schema_setup_objects_size
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	-1 (signifies autoscaling; do not assign this literal value)
Minimum Value	-1 (signifies autoscaling; do not assign this literal value)
Maximum Value	1048576

The number of rows in the setup_objects table.

- performance_schema_show_processlist

Command-Line Format	--performance-schema-show-processlist[={OFF|ON}]
Introduced	8.0.22
System Variable	performance_schema_show_processlist
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

The SHOW PROCESSLIST statement provides process information by collecting thread data from all active threads. The performance_schema_show_processlist variable determines which SHOW PROCESSLIST implementation to use:

- The default implementation iterates across active threads from within the thread manager while holding a global mutex. This has negative performance consequences, particularly on busy systems.
- The alternative SHOW PROCESSLIST implementation is based on the Performance Schema processlist table. This implementation queries active thread data from the Performance Schema rather than the thread manager and does not require a mutex.

To enable the alternative implementation, enable the performance_schema_show_processlist system variable. To ensure that the default and alternative implementations yield the same information, certain configuration requirements must be met; see Section 27.12.21.6, “The processlist Table”.

- performance_schema_users_size

Command-Line Format	`--performance-schema-users-size=#`
System Variable	`performance_schema_users_size`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autoscaling; do not assign this literal value)
Minimum Value	`-1` (signifies autoscaling; do not assign this literal value)
Maximum Value	`1048576`

The number of rows in the `users` table. If this variable is 0, the Performance Schema does not maintain connection statistics in the `users` table or status variable information in the `status_by_user` table.

27.16 Performance Schema Status Variables

The Performance Schema implements several status variables that provide information about instrumentation that could not be loaded or created due to memory constraints:

```
mysql> SHOW STATUS LIKE 'perf%';
+-------------------------------------------+-------+
| Variable_name                             | Value |
+-------------------------------------------+-------+
| Performance_schema_accounts_lost          | 0     |
| Performance_schema_cond_classes_lost      | 0     |
| Performance_schema_cond_instances_lost    | 0     |
| Performance_schema_file_classes_lost      | 0     |
| Performance_schema_file_handles_lost      | 0     |
| Performance_schema_file_instances_lost    | 0     |
| Performance_schema_hosts_lost             | 0     |
| Performance_schema_locker_lost            | 0     |
| Performance_schema_mutex_classes_lost     | 0     |
| Performance_schema_mutex_instances_lost   | 0     |
| Performance_schema_rwlock_classes_lost    | 0     |
| Performance_schema_rwlock_instances_lost  | 0     |
| Performance_schema_socket_classes_lost    | 0     |
| Performance_schema_socket_instances_lost  | 0     |
| Performance_schema_stage_classes_lost     | 0     |
| Performance_schema_statement_classes_lost | 0     |
| Performance_schema_table_handles_lost     | 0     |
| Performance_schema_table_instances_lost   | 0     |
| Performance_schema_thread_classes_lost    | 0     |
| Performance_schema_thread_instances_lost  | 0     |
| Performance_schema_users_lost             | 0     |
+-------------------------------------------+-------+
```

For information on using these variables to check Performance Schema status, see Section 27.7, “Performance Schema Status Monitoring”.

Performance Schema status variables have the following meanings:

- `Performance_schema_accounts_lost`

  The number of times a row could not be added to the `accounts` table because it was full.

- `Performance_schema_cond_classes_lost`

  How many condition instruments could not be loaded.

- `Performance_schema_cond_instances_lost`

  How many condition instrument instances could not be created.

- `Performance_schema_digest_lost`

  The number of digest instances that could not be instrumented in the `events_statements_summary_by_digest` table. This can be nonzero if the value of `performance_schema_digests_size` is too small.

- `Performance_schema_file_classes_lost`

  How many file instruments could not be loaded.

- `Performance_schema_file_handles_lost`

  How many file instrument instances could not be opened.

- `Performance_schema_file_instances_lost`

  How many file instrument instances could not be created.

- `Performance_schema_hosts_lost`

  The number of times a row could not be added to the `hosts` table because it was full.

- `Performance_schema_index_stat_lost`

  The number of indexes for which statistics were lost. This can be nonzero if the value of `performance_schema_max_index_stat` is too small.

- `Performance_schema_locker_lost`

  How many events are “lost” or not recorded, due to the following conditions:

  - Events are recursive (for example, waiting for A caused a wait on B, which caused a wait on C).
  - The depth of the nested events stack is greater than the limit imposed by the implementation.

  Events recorded by the Performance Schema are not recursive, so this variable should always be 0.

- `Performance_schema_memory_classes_lost`

  The number of times a memory instrument could not be loaded.

- `Performance_schema_metadata_lock_lost`

  The number of metadata locks that could not be instrumented in the `metadata_locks` table. This can be nonzero if the value of `performance_schema_max_metadata_locks` is too small.

- `Performance_schema_mutex_classes_lost`

  How many mutex instruments could not be loaded.

- `Performance_schema_mutex_instances_lost`

  How many mutex instrument instances could not be created.

- `Performance_schema_nested_statement_lost`

  The number of stored program statements for which statistics were lost. This can be nonzero if the value of `performance_schema_max_statement_stack` is too small.

- `Performance_schema_prepared_statements_lost`

The number of prepared statements that could not be instrumented in the `prepared_statements_instances` table. This can be nonzero if the value of `performance_schema_max_prepared_statements_instances` is too small.

- `Performance_schema_program_lost`

  The number of stored programs for which statistics were lost. This can be nonzero if the value of `performance_schema_max_program_instances` is too small.

- `Performance_schema_rwlock_classes_lost`

  How many rwlock instruments could not be loaded.

- `Performance_schema_rwlock_instances_lost`

  How many rwlock instrument instances could not be created.

- `Performance_schema_session_connect_attrs_longest_seen`

  In addition to the connection attribute size-limit check performed by the Performance Schema against the value of the `performance_schema_session_connect_attrs_size` system variable, the server performs a preliminary check, imposing a limit of 64KB on the aggregate size of connection attribute data it accepts. If a client attempts to send more than 64KB of attribute data, the server rejects the connection. Otherwise, the server considers the attribute buffer valid and tracks the size of the longest such buffer in the `Performance_schema_session_connect_attrs_longest_seen` status variable. If this value is larger than `performance_schema_session_connect_attrs_size`, DBAs may wish to increase the latter value, or, alternatively, investigate which clients are sending large amounts of attribute data.

  For more information about connection attributes, see Section 27.12.9, “Performance Schema Connection Attribute Tables”.

- `Performance_schema_session_connect_attrs_lost`

  The number of connections for which connection attribute truncation has occurred. For a given connection, if the client sends connection attribute key-value pairs for which the aggregate size is larger than the reserved storage permitted by the value of the `performance_schema_session_connect_attrs_size` system variable, the Performance Schema truncates the attribute data and increments `Performance_schema_session_connect_attrs_lost`. If this value is nonzero, you may wish to set `performance_schema_session_connect_attrs_size` to a larger value.

  For more information about connection attributes, see Section 27.12.9, “Performance Schema Connection Attribute Tables”.

- `Performance_schema_socket_classes_lost`

  How many socket instruments could not be loaded.

- `Performance_schema_socket_instances_lost`

  How many socket instrument instances could not be created.

- `Performance_schema_stage_classes_lost`

  How many stage instruments could not be loaded.

- `Performance_schema_statement_classes_lost`

  How many statement instruments could not be loaded.

- `Performance_schema_table_handles_lost`

  How many table instrument instances could not be opened. This can be nonzero if the value of `performance_schema_max_table_handles` is too small.

- `Performance_schema_table_instances_lost`

  How many table instrument instances could not be created.

- `Performance_schema_table_lock_stat_lost`

  The number of tables for which lock statistics were lost. This can be nonzero if the value of `performance_schema_max_table_lock_stat` is too small.

- `Performance_schema_thread_classes_lost`

  How many thread instruments could not be loaded.

- `Performance_schema_thread_instances_lost`

  The number of thread instances that could not be instrumented in the `threads` table. This can be nonzero if the value of `performance_schema_max_thread_instances` is too small.

- `Performance_schema_users_lost`

  The number of times a row could not be added to the `users` table because it was full.

27.17 The Performance Schema Memory-Allocation Model

The Performance Schema uses this memory allocation model:

- May allocate memory at server startup
- May allocate additional memory during server operation
- Never free memory during server operation (although it might be recycled)
- Free all memory used at shutdown

The result is to relax memory constraints so that the Performance Schema can be used with less configuration, and to decrease the memory footprint so that consumption scales with server load. Memory used depends on the load actually seen, not the load estimated or explicitly configured for.

Several Performance Schema sizing parameters are autoscaled and need not be configured explicitly unless you want to establish an explicit limit on memory allocation:

```
performance_schema_accounts_size
performance_schema_hosts_size
performance_schema_max_cond_instances
performance_schema_max_file_instances
performance_schema_max_index_stat
performance_schema_max_metadata_locks
performance_schema_max_mutex_instances
performance_schema_max_prepared_statements_instances
performance_schema_max_program_instances
performance_schema_max_rwlock_instances
performance_schema_max_socket_instances
performance_schema_max_table_handles
performance_schema_max_table_instances
performance_schema_max_table_lock_stat
performance_schema_max_thread_instances
performance_schema_users_size
```

For an autoscaled parameter, configuration works like this:

- With the value set to -1 (the default), the parameter is autoscaled:
  - The corresponding internal buffer is empty initially and no memory is allocated.
  - As the Performance Schema collects data, memory is allocated in the corresponding buffer. The buffer size is unbounded, and may grow with the load.
- With the value set to 0:
  - The corresponding internal buffer is empty initially and no memory is allocated.
- With the value set to *N* > 0:
  - The corresponding internal buffer is empty initially and no memory is allocated.
  - As the Performance Schema collects data, memory is allocated in the corresponding buffer, until the buffer size reaches *N*.
  - Once the buffer size reaches *N*, no more memory is allocated. Data collected by the Performance Schema for this buffer is lost, and any corresponding “lost instance” counters are incremented.

To see how much memory the Performance Schema is using, check the instruments designed for that purpose. The Performance Schema allocates memory internally and associates each buffer with a dedicated instrument so that memory consumption can be traced to individual buffers. Instruments named with the prefix `memory/performance_schema/` expose how much memory is allocated for these internal buffers. The buffers are global to the server, so the instruments are displayed only in the `memory_summary_global_by_event_name` table, and not in other `memory_summary_by_xxx_by_event_name` tables.

This query shows the information associated with the memory instruments:

```
SELECT * FROM performance_schema.memory_summary_global_by_event_name
WHERE EVENT_NAME LIKE 'memory/performance_schema/%';
```

27.18 Performance Schema and Plugins

Removing a plugin with `UNINSTALL PLUGIN` does not affect information already collected for code in that plugin. Time spent executing the code while the plugin was loaded was still spent even if the plugin is unloaded later. The associated event information, including aggregate information, remains readable in `performance_schema` database tables. For additional information about the effect of plugin installation and removal, see Section 27.7, “Performance Schema Status Monitoring”.

A plugin implementor who instruments plugin code should document its instrumentation characteristics to enable those who load the plugin to account for its requirements. For example, a third-party storage engine should include in its documentation how much memory the engine needs for mutex and other instruments.

27.19 Using the Performance Schema to Diagnose Problems

The Performance Schema is a tool to help a DBA do performance tuning by taking real measurements instead of “wild guesses.” This section demonstrates some ways to use the Performance Schema for this purpose. The discussion here relies on the use of event filtering, which is described in Section 27.4.2, “Performance Schema Event Filtering”.

The following example provides one methodology that you can use to analyze a repeatable problem, such as investigating a performance bottleneck. To begin, you should have a repeatable use case where performance is deemed “too slow” and needs optimization, and you should enable all instrumentation (no pre-filtering at all).

1. Run the use case.

2. Using the Performance Schema tables, analyze the root cause of the performance problem. This analysis relies heavily on post-filtering.
3. For problem areas that are ruled out, disable the corresponding instruments. For example, if analysis shows that the issue is not related to file I/O in a particular storage engine, disable the file I/O instruments for that engine. Then truncate the history and summary tables to remove previously collected events.
4. Repeat the process at step 1.

   With each iteration, the Performance Schema output, particularly the `events_waits_history_long` table, contains less and less “noise” caused by nonsignificant instruments, and given that this table has a fixed size, contains more and more data relevant to the analysis of the problem at hand.

   With each iteration, investigation should lead closer and closer to the root cause of the problem, as the “signal/noise” ratio improves, making analysis easier.
5. Once a root cause of performance bottleneck is identified, take the appropriate corrective action, such as:
   - Tune the server parameters (cache sizes, memory, and so forth).
   - Tune a query by writing it differently,
   - Tune the database schema (tables, indexes, and so forth).
   - Tune the code (this applies to storage engine or server developers only).
6. Start again at step 1, to see the effects of the changes on performance.

The `mutex_instances.LOCKED_BY_THREAD_ID` and `rwlock_instances.WRITE_LOCKED_BY_THREAD_ID` columns are extremely important for investigating performance bottlenecks or deadlocks. This is made possible by Performance Schema instrumentation as follows:

1. Suppose that thread 1 is stuck waiting for a mutex.
2. You can determine what the thread is waiting for:

   ```
   SELECT * FROM performance_schema.events_waits_current
   WHERE THREAD_ID = thread_1;
   ```

   Say the query result identifies that the thread is waiting for mutex A, found in `events_waits_current.OBJECT_INSTANCE_BEGIN`.
3. You can determine which thread is holding mutex A:

   ```
   SELECT * FROM performance_schema.mutex_instances
   WHERE OBJECT_INSTANCE_BEGIN = mutex_A;
   ```

   Say the query result identifies that it is thread 2 holding mutex A, as found in `mutex_instances.LOCKED_BY_THREAD_ID`.
4. You can see what thread 2 is doing:

   ```
   SELECT * FROM performance_schema.events_waits_current
   WHERE THREAD_ID = thread_2;
   ```

27.19.1 Query Profiling Using Performance Schema

The following example demonstrates how to use Performance Schema statement events and stage events to retrieve data comparable to profiling information provided by `SHOW PROFILES` and `SHOW PROFILE` statements.

The `setup_actors` table can be used to limit the collection of historical events by host, user, or account to reduce runtime overhead and the amount of data collected in history tables. The first step of the example shows how to limit collection of historical events to a specific user.

Performance Schema displays event timer information in picoseconds (trillionths of a second) to normalize timing data to a standard unit. In the following example, `TIMER_WAIT` values are divided by 1000000000000 to show data in units of seconds. Values are also truncated to 6 decimal places to display data in the same format as `SHOW PROFILES` and `SHOW PROFILE` statements.

1. Limit the collection of historical events to the user that runs the query. By default, `setup_actors` is configured to allow monitoring and historical event collection for all foreground threads:

```
mysql> SELECT * FROM performance_schema.setup_actors;
+------+------+------+---------+---------+
| HOST | USER | ROLE | ENABLED | HISTORY |
+------+------+------+---------+---------+
| %    | %    | %    | YES     | YES     |
+------+------+------+---------+---------+
```

   Update the default row in the `setup_actors` table to disable historical event collection and monitoring for all foreground threads, and insert a new row that enables monitoring and historical event collection for the user that runs the query:

```
mysql> UPDATE performance_schema.setup_actors
       SET ENABLED = 'NO', HISTORY = 'NO'
       WHERE HOST = '%' AND USER = '%';

mysql> INSERT INTO performance_schema.setup_actors
       (HOST,USER,ROLE,ENABLED,HISTORY)
       VALUES('localhost','test_user','%','YES','YES');
```

   Data in the `setup_actors` table should now appear similar to the following:

```
mysql> SELECT * FROM performance_schema.setup_actors;
+-----------+-----------+------+---------+---------+
| HOST      | USER      | ROLE | ENABLED | HISTORY |
+-----------+-----------+------+---------+---------+
| %         | %         | %    | NO      | NO      |
| localhost | test_user | %    | YES     | YES     |
+-----------+-----------+------+---------+---------+
```

2. Ensure that statement and stage instrumentation is enabled by updating the `setup_instruments` table. Some instruments may already be enabled by default.

```
mysql> UPDATE performance_schema.setup_instruments
       SET ENABLED = 'YES', TIMED = 'YES'
       WHERE NAME LIKE '%statement/%';

mysql> UPDATE performance_schema.setup_instruments
       SET ENABLED = 'YES', TIMED = 'YES'
       WHERE NAME LIKE '%stage/%';
```

3. Ensure that `events_statements_*` and `events_stages_*` consumers are enabled. Some consumers may already be enabled by default.

```
mysql> UPDATE performance_schema.setup_consumers
       SET ENABLED = 'YES'
       WHERE NAME LIKE '%events_statements_%';

mysql> UPDATE performance_schema.setup_consumers
       SET ENABLED = 'YES'
       WHERE NAME LIKE '%events_stages_%';
```

4. Under the user account you are monitoring, run the statement that you want to profile. For example:

```
mysql> SELECT * FROM employees.employees WHERE emp_no = 10001;
+--------+------------+------------+-----------+--------+------------+
| emp_no | birth_date | first_name | last_name | gender | hire_date |
```

```
+--------+------------+------------+-----------+--------+------------+
|  10001 | 1953-09-02 | Georgi     | Facello   | M      | 1986-06-26 |
+--------+------------+------------+-----------+--------+------------+
```

5. Identify the `EVENT_ID` of the statement by querying the `events_statements_history_long` table. This step is similar to running `SHOW PROFILES` to identify the `Query_ID`. The following query produces output similar to `SHOW PROFILES`:

```
mysql> SELECT EVENT_ID, TRUNCATE(TIMER_WAIT/1000000000000,6) as Duration, SQL_TEXT
       FROM performance_schema.events_statements_history_long WHERE SQL_TEXT like '%10001%';
+----------+----------+-----------------------------------------------------------+
| event_id | duration | sql_text                                                  |
+----------+----------+-----------------------------------------------------------+
|       31 | 0.028310 | SELECT * FROM employees.employees WHERE emp_no = 10001 |
+----------+----------+-----------------------------------------------------------+
```

6. Query the `events_stages_history_long` table to retrieve the statement's stage events. Stages are linked to statements using event nesting. Each stage event record has a `NESTING_EVENT_ID` column that contains the `EVENT_ID` of the parent statement.

```
mysql> SELECT event_name AS Stage, TRUNCATE(TIMER_WAIT/1000000000000,6) AS Duration
       FROM performance_schema.events_stages_history_long WHERE NESTING_EVENT_ID=31;
+--------------------------------+----------+
| Stage                          | Duration |
+--------------------------------+----------+
| stage/sql/starting             | 0.000080 |
| stage/sql/checking permissions | 0.000005 |
| stage/sql/Opening tables       | 0.027759 |
| stage/sql/init                 | 0.000052 |
| stage/sql/System lock          | 0.000009 |
| stage/sql/optimizing           | 0.000006 |
| stage/sql/statistics           | 0.000082 |
| stage/sql/preparing            | 0.000008 |
| stage/sql/executing            | 0.000000 |
| stage/sql/Sending data         | 0.000017 |
| stage/sql/end                  | 0.000001 |
| stage/sql/query end            | 0.000004 |
| stage/sql/closing tables       | 0.000006 |
| stage/sql/freeing items        | 0.000272 |
| stage/sql/cleaning up          | 0.000001 |
+--------------------------------+----------+
```

27.19.2 Obtaining Parent Event Information

The `data_locks` table shows data locks held and requested. Rows of this table have a `THREAD_ID` column indicating the thread ID of the session that owns the lock, and an `EVENT_ID` column indicating the Performance Schema event that caused the lock. Tuples of (`THREAD_ID`, `EVENT_ID`) values implicitly identify a parent event in other Performance Schema tables:

- The parent wait event in the `events_waits_xxx` tables
- The parent stage event in the `events_stages_xxx` tables
- The parent statement event in the `events_statements_xxx` tables
- The parent transaction event in the `events_transactions_current` table

To obtain details about the parent event, join the `THREAD_ID` and `EVENT_ID` columns with the columns of like name in the appropriate parent event table. The relation is based on a nested set data model, so the join has several clauses. Given parent and child tables represented by `parent` and `child`, respectively, the join looks like this:

```
WHERE
  parent.THREAD_ID = child.THREAD_ID        /* 1 */
  AND parent.EVENT_ID < child.EVENT_ID      /* 2 */
  AND (
    child.EVENT_ID <= parent.END_EVENT_ID   /* 3a */
```

```
    OR parent.END_EVENT_ID IS NULL          /* 3b */
  )
```

The conditions for the join are:

1. The parent and child events are in the same thread.
2. The child event begins after the parent event, so its `EVENT_ID` value is greater than that of the parent.
3. The parent event has either completed or is still running.

To find lock information, `data_locks` is the table containing child events.

The `data_locks` table shows only existing locks, so these considerations apply regarding which table contains the parent event:

- For transactions, the only choice is `events_transactions_current`. If a transaction is completed, it may be in the transaction history tables, but the locks are gone already.
- For statements, it all depends on whether the statement that took a lock is a statement in a transaction that has already completed (use `events_statements_history`) or the statement is still running (use `events_statements_current`).
- For stages, the logic is similar to that for statements; use `events_stages_history` or `events_stages_current`.
- For waits, the logic is similar to that for statements; use `events_waits_history` or `events_waits_current`. However, so many waits are recorded that the wait that caused a lock is most likely gone from the history tables already.

Wait, stage, and statement events disappear quickly from the history. If a statement that executed a long time ago took a lock but is in a still-open transaction, it might not be possible to find the statement, but it is possible to find the transaction.

This is why the nested set data model works better for locating parent events. Following links in a parent/child relationship (data lock -> parent wait -> parent stage -> parent transaction) does not work well when intermediate nodes are already gone from the history tables.

The following scenario illustrates how to find the parent transaction of a statement in which a lock was taken:

Session A:

```
[1] START TRANSACTION;
[2] SELECT * FROM t1 WHERE pk = 1;
[3] SELECT 'Hello, world';
```

Session B:

```
SELECT ...
FROM performance_schema.events_transactions_current AS parent
  INNER JOIN performance_schema.data_locks AS child
WHERE
  parent.THREAD_ID = child.THREAD_ID
  AND parent.EVENT_ID < child.EVENT_ID
  AND (
    child.EVENT_ID <= parent.END_EVENT_ID
    OR parent.END_EVENT_ID IS NULL
  );
```

The query for session B should show statement [2] as owning a data lock on the record with `pk=1`.

If session A executes more statements, [2] fades out of the history table.

The query should show the transaction that started in [1], regardless of how many statements, stages, or waits were executed.

To see more data, you can also use the `events_xxx_history_long` tables, except for transactions, assuming no other query runs in the server (so that history is preserved).

27.20 Restrictions on Performance Schema

The Performance Schema avoids using mutexes to collect or produce data, so there are no guarantees of consistency and results can sometimes be incorrect. Event values in `performance_schema` tables are nondeterministic and nonrepeatable.

If you save event information in another table, you should not assume that the original events remain available later. For example, if you select events from a `performance_schema` table into a temporary table, intending to join that table with the original table later, there might be no matches.

`mysqldump` and `BACKUP DATABASE` ignore tables in the `performance_schema` database.

Tables in the `performance_schema` database cannot be locked with `LOCK TABLES`, except the `setup_xxx` tables.

Tables in the `performance_schema` database cannot be indexed.

Tables in the `performance_schema` database are not replicated.

The types of timers might vary per platform. The `performance_timers` table shows which event timers are available. If the values in this table for a given timer name are `NULL`, that timer is not supported on your platform.

Instruments that apply to storage engines might not be implemented for all storage engines. Instrumentation of each third-party engine is the responsibility of the engine maintainer.

Chapter 28 MySQL sys Schema

Table of Contents

MySQL 8.0 includes the `sys` schema, a set of objects that helps DBAs and developers interpret data collected by the Performance Schema. `sys` schema objects can be used for typical tuning and diagnosis use cases. Objects in this schema include:

- Views that summarize Performance Schema data into more easily understandable form.
- Stored procedures that perform operations such as Performance Schema configuration and generating diagnostic reports.
- Stored functions that query Performance Schema configuration and provide formatting services.

For new installations, the `sys` schema is installed by default during data directory initialization if you use `mysqld` with the `--initialize` or `--initialize-insecure` option. If this is not desired, you can drop the `sys` schema manually after initialization if it is unneeded.

The MySQL upgrade procedure produces an error if a `sys` schema exists but has no `version` view, on the assumption that absence of this view indicates a user-created `sys` schema. To upgrade in this case, remove or rename the existing `sys` schema first.

`sys` schema objects have a `DEFINER` of `'mysql.sys'@'localhost'`. Use of the dedicated `mysql.sys` account avoids problems that occur if a DBA renames or removes the `root` account.

28.1 Prerequisites for Using the sys Schema

Before using the `sys` schema, the prerequisites described in this section must be satisfied.

Because the `sys` schema provides an alternative means of accessing the Performance Schema, the Performance Schema must be enabled for the `sys` schema to work. See Section 27.3, “Performance Schema Startup Configuration”.

For full access to the `sys` schema, a user must have these privileges:

- `SELECT` on all `sys` tables and views
- `EXECUTE` on all `sys` stored procedures and functions
- `INSERT` and `UPDATE` for the `sys_config` table, if changes are to be made to it
- Additional privileges for certain `sys` schema stored procedures and functions, as noted in their descriptions (for example, the `ps_setup_save()` procedure)

It is also necessary to have privileges for the objects underlying the `sys` schema objects:

- `SELECT` on any Performance Schema tables accessed by `sys` schema objects, and `UPDATE` for any tables to be updated using `sys` schema objects
- `PROCESS` for the `INFORMATION_SCHEMA INNODB_BUFFER_PAGE` table

Certain Performance Schema instruments and consumers must be enabled and (for instruments) timed to take full advantage of `sys` schema capabilities:

- All `wait` instruments
- All `stage` instruments
- All `statement` instruments
- *`xxx`*`_current` and *`xxx`*`_history_long` consumers for all events

You can use the `sys` schema itself to enable all of the additional instruments and consumers:

```
CALL sys.ps_setup_enable_instrument('wait');
CALL sys.ps_setup_enable_instrument('stage');
CALL sys.ps_setup_enable_instrument('statement');
CALL sys.ps_setup_enable_consumer('current');
CALL sys.ps_setup_enable_consumer('history_long');
```

Note

For many uses of the `sys` schema, the default Performance Schema is sufficient for data collection. Enabling all the instruments and consumers just mentioned has a performance impact, so it is preferable to enable only the additional configuration you need. Also, remember that if you enable additional configuration, you can easily restore the default configuration like this:

```
CALL sys.ps_setup_reset_to_default(TRUE);
```

28.2 Using the sys Schema

You can make the `sys` schema the default schema so that references to its objects need not be qualified with the schema name:

```
mysql> USE sys;
Database changed
mysql> SELECT * FROM version;
+-------------+---------------+
| sys_version | mysql_version |
+-------------+---------------+
| 2.1.1       | 8.0.26-debug  |
+-------------+---------------+
```

(The `version` view shows the `sys` schema and MySQL server versions.)

To access `sys` schema objects while a different schema is the default (or simply to be explicit), qualify object references with the schema name:

```
mysql> SELECT * FROM sys.version;
+-------------+---------------+
| sys_version | mysql_version |
+-------------+---------------+
| 2.1.1       | 8.0.26-debug  |
+-------------+---------------+
```

The `sys` schema contains many views that summarize Performance Schema tables in various ways. Most of these views come in pairs, such that one member of the pair has the same name as the other member, plus a `x$` prefix. For example, the `host_summary_by_file_io` view summarizes file I/O grouped by host and displays latencies converted from picoseconds to more readable values (with units);

```
mysql> SELECT * FROM sys.host_summary_by_file_io;
+------------+-------+------------+
| host       | ios   | io_latency |
+------------+-------+------------+
| localhost  | 67570 | 5.38 s     |
| background |  3468 | 4.18 s     |
+------------+-------+------------+
```

The `x$host_summary_by_file_io` view summarizes the same data but displays unformatted picosecond latencies:

```
mysql> SELECT * FROM sys.x$host_summary_by_file_io;
+------------+-------+---------------+
| host       | ios   | io_latency    |
+------------+-------+---------------+
| localhost  | 67574 | 5380678125144 |
| background |  3474 | 4758696829416 |
+------------+-------+---------------+
```

The view without the `x$` prefix is intended to provide output that is more user friendly and easier for humans to read. The view with the `x$` prefix that displays the same values in raw form is intended more for use with other tools that perform their own processing on the data. For additional information about the differences between non-`x$` and `x$` views, see Section 28.4.3, “sys Schema Views”.

To examine `sys` schema object definitions, use the appropriate `SHOW` statement or `INFORMATION_SCHEMA` query. For example, to examine the definitions of the `session` view and `format_bytes()` function, use these statements:

```
mysql> SHOW CREATE VIEW sys.session;
mysql> SHOW CREATE FUNCTION sys.format_bytes;
```

However, those statements display the definitions in relatively unformatted form. To view object definitions with more readable formatting, access the individual `.sql` files found under the `scripts/sys_schema` in MySQL source distributions. Prior to MySQL 8.0.18, the sources are maintained in a separate distribution available from the `sys` schema development website at https://github.com/mysql/mysql-sys.

Neither `mysqldump` nor `mysqlpump` dump the `sys` schema by default. To generate a dump file, name the `sys` schema explicitly on the command line using either of these commands:

```
mysqldump --databases --routines sys > sys_dump.sql
mysqlpump sys > sys_dump.sql
```

To reinstall the schema from the dump file, use this command:

```
mysql < sys_dump.sql
```

28.3 sys Schema Progress Reporting

The following `sys` schema views provide progress reporting for long-running transactions:

```
processlist
session
x$processlist
x$session
```

Assuming that the required instruments and consumers are enabled, the `progress` column of these views shows the percentage of work completed for stages that support progress reporting.

Stage progress reporting requires that the `events_stages_current` consumer be enabled, as well as the instruments for which progress information is desired. Instruments for these stages currently support progress reporting:

```
stage/sql/Copying to tmp table
stage/innodb/alter table (end)
stage/innodb/alter table (flush)
stage/innodb/alter table (insert)
stage/innodb/alter table (log apply index)
stage/innodb/alter table (log apply table)
stage/innodb/alter table (merge sort)
stage/innodb/alter table (read PK and internal sort)
stage/innodb/buffer pool load
```

For stages that do not support estimated and completed work reporting, or if the required instruments or consumers are not enabled, the `progress` column is `NULL`.

28.4 sys Schema Object Reference

The sys schema includes tables and triggers, views, and stored procedures and functions. The following sections provide details for each of these objects.

28.4.1 sys Schema Object Index

The following tables list sys schema objects and provide a short description of each one.

Table 28.1 sys Schema Tables and Triggers

Table or Trigger Name	Description
sys_config	sys schema configuration options table
sys_config_insert_set_user	sys_config insert trigger
sys_config_update_set_user	sys_config update trigger

Table 28.2 sys Schema Views

View Name	Description	Deprecated
host_summary, x $host_summary	Statement activity, file I/O, and connections, grouped by host	
host_summary_by_file_io, x $host_summary_by_file_io	File I/O, grouped by host	
host_summary_by_file_io_type, x $host_summary_by_file_io_type	File I/O, grouped by host and event type	
host_summary_by_stages, x $host_summary_by_stages	Statement stages, grouped by host	
host_summary_by_statement_latency, x $host_summary_by_statement_latency	Statement statistics, grouped by host	
host_summary_by_statement_type, x $host_summary_by_statement_type	Statements executed, grouped by host and statement	
innodb_buffer_stats_by_schema, x $innodb_buffer_stats_by_schema	InnoDB buffer information, grouped by schema	
innodb_buffer_stats_by_table, x $innodb_buffer_stats_by_table	InnoDB buffer information, grouped by schema and table	
innodb_lock_waits, x $innodb_lock_waits	InnoDB lock information	
io_by_thread_by_latency, x $io_by_thread_by_latency	I/O consumers, grouped by thread	
io_global_by_file_by_bytes, x $io_global_by_file_by_bytes	Global I/O consumers, grouped by file and bytes	
io_global_by_file_by_latency, x $io_global_by_file_by_latency	Global I/O consumers, grouped by file and latency	

View Name	Description	Deprecated
io_global_by_wait_by_bytes, x $io_global_by_wait_by_bytes	Global I/O consumers, grouped by bytes	
io_global_by_wait_by_latency, x $io_global_by_wait_by_latency	Global I/O consumers, grouped by latency	
latest_file_io, x $latest_file_io	Most recent I/O, grouped by file and thread	
memory_by_host_by_current_bytes, x $memory_by_host_by_current_bytes	Memory use, grouped by host	
memory_by_thread_by_current_bytes, x $memory_by_thread_by_current_bytes	Memory use, grouped by thread	
memory_by_user_by_current_bytes, x $memory_by_user_by_current_bytes	Memory use, grouped by user	
memory_global_by_current_bytes, x $memory_global_by_current_bytes	Memory use, grouped by allocation type	
memory_global_total, x $memory_global_total	Total memory use	
metrics	Server metrics	
processlist, x $processlist	Processlist information	
ps_check_lost_instrumentation	Variables that have lost instruments	
schema_auto_increment_columns	AUTO_INCREMENT column information	
schema_index_statistics, x $schema_index_statistics	Index statistics	
schema_object_overview	Types of objects within each schema	
schema_redundant_indexes	Duplicate or redundant indexes	
schema_table_lock_waits, x $schema_table_lock_waits	Sessions waiting for metadata locks	
schema_table_statistics, x $schema_table_statistics	Table statistics	
schema_table_statistics_with_buffer, x $schema_table_statistics_with_buffer	Table statistics, including InnoDB buffer pool statistics	
schema_tables_with_full_table_scans, x $schema_tables_with_full_table_scans	Tables being accessed with full scans	
schema_unused_indexes	Indexes not in active use	

View Name	Description	Deprecated
session, x$session	Processlist information for user sessions	
session_ssl_status	Connection SSL information	
statement_analysis, x$statement_analysis	Statement aggregate statistics	
statements_with_errors_or_warnings, x$statements_with_errors_or_warnings	Statements that have produced errors or warnings	
statements_with_full_table_scans, x$statements_with_full_table_scans	Statements that have done full table scans	
statements_with_runtimes_in_95th_percentile, x$statements_with_runtimes_in_95th_percentile	Statements with highest average runtime	
statements_with_sorting, x$statements_with_sorting	Statements that performed sorts	
statements_with_temp_tables, x$statements_with_temp_tables	Statements that used temporary tables	
user_summary, x$user_summary	User statement and connection activity	
user_summary_by_file_io, x$user_summary_by_file_io	File I/O, grouped by user	
user_summary_by_file_io_type, x$user_summary_by_file_io_type	File I/O, grouped by user and event	
user_summary_by_stages, x$user_summary_by_stages	Stage events, grouped by user	
user_summary_by_statement_latency, x$user_summary_by_statement_latency	Statement statistics, grouped by user	
user_summary_by_statement_type, x$user_summary_by_statement_type	Statements executed, grouped by user and statement	
version	Current sys schema and MySQL server versions	8.0.18
wait_classes_global_by_avg_latency, x$wait_classes_global_by_avg_latency	Wait class average latency, grouped by event class	
wait_classes_global_by_latency, x$wait_classes_global_by_latency	Wait class total latency, grouped by event class	
waits_by_host_by_latency, x$waits_by_host_by_latency	Wait events, grouped by host and event	

View Name	Description	Deprecated
waits_by_user_by_latency, x $waits_by_user_by_latency	Wait events, grouped by user and event	
waits_global_by_latency, x $waits_global_by_latency	Wait events, grouped by event	
x $ps_digest_95th_percentile_by_avg_us	Helper view for 95th-percentile views	
x $ps_digest_avg_latency_distribution	Helper view for 95th-percentile views	
x $ps_schema_table_statistics_io	Helper view for table-statistics views	
x$schema_flattened_keys	Helper view for schema_redundant_indexes	

Table 28.3 sys Schema Stored Procedures

Procedure Name	Description
create_synonym_db()	Create synonym for schema
diagnostics()	Collect system diagnostic information
execute_prepared_stmt()	Execute prepared statement
ps_setup_disable_background_threads()	Disable background thread instrumentation
ps_setup_disable_consumer()	Disable consumers
ps_setup_disable_instrument()	Disable instruments
ps_setup_disable_thread()	Disable instrumentation for thread
ps_setup_enable_background_threads()	Enable background thread instrumentation
ps_setup_enable_consumer()	Enable consumers
ps_setup_enable_instrument()	Enable instruments
ps_setup_enable_thread()	Enable instrumentation for thread
ps_setup_reload_saved()	Reload saved Performance Schema configuration
ps_setup_reset_to_default()	Reset saved Performance Schema configuration
ps_setup_save()	Save Performance Schema configuration
ps_setup_show_disabled()	Display disabled Performance Schema configuration
ps_setup_show_disabled_consumers()	Display disabled Performance Schema consumers
ps_setup_show_disabled_instruments()	Display disabled Performance Schema instruments
ps_setup_show_enabled()	Display enabled Performance Schema configuration
ps_setup_show_enabled_consumers()	Display enabled Performance Schema consumers
ps_setup_show_enabled_instruments()	Display enabled Performance Schema instruments
ps_statement_avg_latency_histogram()	Display statement latency histogram
ps_trace_statement_digest()	Trace Performance Schema instrumentation for digest
ps_trace_thread()	Dump Performance Schema data for thread

Procedure Name	Description
ps_truncate_all_tables()	Truncate Performance Schema summary tables
statement_performance_analyzer()	Report of statements running on server
table_exists()	Whether a table exists

Table 28.4 sys Schema Stored Functions

Function Name	Description	Deprecated
extract_schema_from_file_name()	Extract schema name part of file name	
extract_table_from_file_name()	Extract table name part of file name	
format_bytes()	Convert byte count to value with units	8.0.16
format_path()	Replace directories in path name with symbolic system variable names	
format_statement()	Truncate long statement to fixed length	
format_time()	Convert picoseconds time to value with units	8.0.16
list_add()	Add item to list	
list_drop()	Remove item from list	
ps_is_account_enabled()	Whether Performance Schema instrumentation for account is enabled	
ps_is_consumer_enabled()	Whether Performance Schema consumer is enabled	
ps_is_instrument_default_enabled()	Whether Performance Schema instrument is enabled by default	
ps_is_instrument_default_timed()	Whether Performance Schema instrument is timed by default	
ps_is_thread_instrumented()	Whether Performance Schema instrumentation for connection ID is enabled	
ps_thread_account()	Account associated with Performance Schema thread ID	
ps_thread_id()	Performance Schema thread ID associated with connection ID	8.0.16
ps_thread_stack()	Event information for connection ID	
ps_thread_trx_info()	Transaction information for thread ID	
quote_identifier()	Quote string as identifier	
sys_get_config()	sys schema configuration option value	
version_major()	MySQL server major version number	

Function Name	Description	Deprecated
`version_minor()`	MySQL server minor version number	
`version_patch()`	MySQL server patch release version number	

28.4.2 sys Schema Tables and Triggers

The following sections describe `sys` schema tables and triggers.

28.4.2.1 The sys_config Table

This table contains `sys` schema configuration options, one row per option. Configuration changes made by updating this table persist across client sessions and server restarts.

The `sys_config` table has these columns:

- `variable`

  The configuration option name.

- `value`

  The configuration option value.

- `set_time`

  The timestamp of the most recent modification to the row.

- `set_by`

  The account that made the most recent modification to the row. The value is `NULL` if the row has not been changed since the `sys` schema was installed.

As an efficiency measure to minimize the number of direct reads from the `sys_config` table, `sys` schema functions that use a value from this table check for a user-defined variable with a corresponding name, which is the user-defined variable having the same name plus a `@sys.` prefix. (For example, the variable corresponding to the `diagnostics.include_raw` option is `@sys.diagnostics.include_raw`.) If the user-defined variable exists in the current session and is non-`NULL`, the function uses its value in preference to the value in the `sys_config` table. Otherwise, the function reads and uses the value from the table. In the latter case, the calling function conventionally also sets the corresponding user-defined variable to the table value so that further references to the configuration option within the same session use the variable and need not read the table again.

For example, the `statement_truncate_len` option controls the maximum length of statements returned by the `format_statement()` function. The default is 64. To temporarily change the value to 32 for your current session, set the corresponding `@sys.statement_truncate_len` user-defined variable:

```
mysql> SET @stmt = 'SELECT variable, value, set_time, set_by FROM sys_config';
mysql> SELECT sys.format_statement(@stmt);
+-------------------------------------------------------------+
| sys.format_statement(@stmt)                                 |
+-------------------------------------------------------------+
| SELECT variable, value, set_time, set_by FROM sys_config |
+-------------------------------------------------------------+
mysql> SET @sys.statement_truncate_len = 32;
mysql> SELECT sys.format_statement(@stmt);
+----------------------------------+
| sys.format_statement(@stmt)      |
+----------------------------------+
```

```
| SELECT variabl ... ROM sys_config |
+-----------------------------------+
```

Subsequent invocations of `format_statement()` within the session continue to use the user-defined variable value (32), rather than the value stored in the table (64).

To stop using the user-defined variable and revert to using the value in the table, set the variable to `NULL` within your session:

```
mysql> SET @sys.statement_truncate_len = NULL;
mysql> SELECT sys.format_statement(@stmt);
+-----------------------------------------------------+
| sys.format_statement(@stmt)                         |
+-----------------------------------------------------+
| SELECT variable, value, set_time, set_by FROM sys_config |
+-----------------------------------------------------+
```

Alternatively, end your current session (causing the user-defined variable to no longer exist) and begin a new session.

The conventional relationship just described between options in the `sys_config` table and user-defined variables can be exploited to make temporary configuration changes that end when your session ends. However, if you set a user-defined variable and then subsequently change the corresponding table value within the same session, the changed table value is not used in that session as long as the user-defined variable exists with a non-`NULL` value. (The changed table value *is* used in other sessions in which the user-defined variable is not assigned.)

The following list describes the options in the `sys_config` table and the corresponding user-defined variables:

- `diagnostics.allow_i_s_tables`, `@sys.diagnostics.allow_i_s_tables`

  If this option is `ON`, the `diagnostics()` procedure is permitted to perform table scans on the Information Schema `TABLES` table. This can be expensive if there are many tables. The default is `OFF`.

- `diagnostics.include_raw`, `@sys.diagnostics.include_raw`

  If this option is `ON`, the `diagnostics()` procedure includes the raw output from querying the `metrics` view. The default is `OFF`.

- `ps_thread_trx_info.max_length`, `@sys.ps_thread_trx_info.max_length`

  The maximum length for JSON output produced by the `ps_thread_trx_info()` function. The default is 65535.

- `statement_performance_analyzer.limit`, `@sys.statement_performance_analyzer.limit`

  The maximum number of rows to return for views that have no built-in limit. (For example, the `statements_with_runtimes_in_95th_percentile` view has a built-in limit in the sense that it returns only statements with average execution time in the 95th percentile.) The default is 100.

- `statement_performance_analyzer.view`, `@sys.statement_performance_analyzer.view`

  The custom query or view to be used by the `statement_performance_analyzer()` procedure (which is itself invoked by the `diagnostics()` procedure). If the option value contains a space, it is interpreted as a query. Otherwise, it must be the name of an existing view that queries the Performance Schema `events_statements_summary_by_digest` table. There cannot be any `LIMIT` clause in the query or view definition if the `statement_performance_analyzer.limit` configuration option is greater than 0. The default is `NULL` (no custom view defined).

- `statement_truncate_len`, `@sys.statement_truncate_len`

  The maximum length of statements returned by the `format_statement()` function. Longer statements are truncated to this length. The default is 64.

Other options can be added to the `sys_config` table. For example, the `diagnostics()` and `execute_prepared_stmt()` procedures use the `debug` option if it exists, but this option is not part of the `sys_config` table by default because debug output normally is enabled only temporarily, by setting the corresponding `@sys.debug` user-defined variable. To enable debug output without having to set that variable in individual sessions, add the option to the table:

```
mysql> INSERT INTO sys.sys_config (variable, value) VALUES('debug', 'ON');
```

To change the debug setting in the table, do two things. First, modify the value in the table itself:

```
mysql> UPDATE sys.sys_config
       SET value = 'OFF'
       WHERE variable = 'debug';
```

Second, to also ensure that procedure invocations within the current session use the changed value from the table, set the corresponding user-defined variable to `NULL`:

```
mysql> SET @sys.debug = NULL;
```

28.4.2.2 The sys_config_insert_set_user Trigger

For rows added to the `sys_config` table by `INSERT` statements, the `sys_config_insert_set_user` trigger sets the `set_by` column to the current user.

28.4.2.3 The sys_config_update_set_user Trigger

The `sys_config_update_set_user` trigger for the `sys_config` table is similar to the `sys_config_insert_set_user` trigger, but for `UPDATE` statements.

28.4.3 sys Schema Views

The following sections describe `sys` schema views.

The `sys` schema contains many views that summarize Performance Schema tables in various ways. Most of these views come in pairs, such that one member of the pair has the same name as the other member, plus a `x$` prefix. For example, the `host_summary_by_file_io` view summarizes file I/O grouped by host and displays latencies converted from picoseconds to more readable values (with units);

```
mysql> SELECT * FROM sys.host_summary_by_file_io;
+------------+-------+------------+
| host       | ios   | io_latency |
+------------+-------+------------+
| localhost  | 67570 | 5.38 s     |
| background |  3468 | 4.18 s     |
+------------+-------+------------+
```

The `x$host_summary_by_file_io` view summarizes the same data but displays unformatted picosecond latencies:

```
mysql> SELECT * FROM sys.x$host_summary_by_file_io;
+------------+-------+---------------+
| host       | ios   | io_latency    |
+------------+-------+---------------+
| localhost  | 67574 | 5380678125144 |
| background |  3474 | 4758696829416 |
+------------+-------+---------------+
```

The view without the `x$` prefix is intended to provide output that is more user friendly and easier to read. The view with the `x$` prefix that displays the same values in raw form is intended more for use with other tools that perform their own processing on the data.

Views without the `x$` prefix differ from the corresponding `x$` views in these ways:

- Byte counts are formatted with size units using `format_bytes()`.
- Time values are formatted with temporal units using `format_time()`.
- SQL statements are truncated to a maximum display width using `format_statement()`.
- Path name are shortened using `format_path()`.

28.4.3.1 The host_summary and x$host_summary Views

These views summarize statement activity, file I/O, and connections, grouped by host.

The `host_summary` and `x$host_summary` views have these columns:

- `host`

  The host from which the client connected. Rows for which the `HOST` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `statements`

  The total number of statements for the host.

- `statement_latency`

  The total wait time of timed statements for the host.

- `statement_avg_latency`

  The average wait time per timed statement for the host.

- `table_scans`

  The total number of table scans for the host.

- `file_ios`

  The total number of file I/O events for the host.

- `file_io_latency`

  The total wait time of timed file I/O events for the host.

- `current_connections`

  The current number of connections for the host.

- `total_connections`

  The total number of connections for the host.

- `unique_users`

  The number of distinct users for the host.

- `current_memory`

  The current amount of allocated memory for the host.

- `total_memory_allocated`

The total amount of allocated memory for the host.

28.4.3.2 The host_summary_by_file_io and x$host_summary_by_file_io Views

These views summarize file I/O, grouped by host. By default, rows are sorted by descending total file I/O latency.

The `host_summary_by_file_io` and `x$host_summary_by_file_io` views have these columns:

- `host`

  The host from which the client connected. Rows for which the `HOST` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `ios`

  The total number of file I/O events for the host.

- `io_latency`

  The total wait time of timed file I/O events for the host.

28.4.3.3 The host_summary_by_file_io_type and x$host_summary_by_file_io_type Views

These views summarize file I/O, grouped by host and event type. By default, rows are sorted by host and descending total I/O latency.

The `host_summary_by_file_io_type` and `x$host_summary_by_file_io_type` views have these columns:

- `host`

  The host from which the client connected. Rows for which the `HOST` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `event_name`

  The file I/O event name.

- `total`

  The total number of occurrences of the file I/O event for the host.

- `total_latency`

  The total wait time of timed occurrences of the file I/O event for the host.

- `max_latency`

  The maximum single wait time of timed occurrences of the file I/O event for the host.

28.4.3.4 The host_summary_by_stages and x$host_summary_by_stages Views

These views summarize statement stages, grouped by host. By default, rows are sorted by host and descending total latency.

The `host_summary_by_stages` and `x$host_summary_by_stages` views have these columns:

- `host`

The host from which the client connected. Rows for which the `HOST` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `event_name`

  The stage event name.

- `total`

  The total number of occurrences of the stage event for the host.

- `total_latency`

  The total wait time of timed occurrences of the stage event for the host.

- `avg_latency`

  The average wait time per timed occurrence of the stage event for the host.

28.4.3.5 The host_summary_by_statement_latency and x$host_summary_by_statement_latency Views

These views summarize overall statement statistics, grouped by host. By default, rows are sorted by descending total latency.

The `host_summary_by_statement_latency` and `x$host_summary_by_statement_latency` views have these columns:

- `host`

  The host from which the client connected. Rows for which the `HOST` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `total`

  The total number of statements for the host.

- `total_latency`

  The total wait time of timed statements for the host.

- `max_latency`

  The maximum single wait time of timed statements for the host.

- `lock_latency`

  The total time waiting for locks by timed statements for the host.

- `cpu_latency`

  The time spent on CPU for the current thread.

- `rows_sent`

  The total number of rows returned by statements for the host.

- `rows_examined`

  The total number of rows read from storage engines by statements for the host.

- `rows_affected`

  The total number of rows affected by statements for the host.

- `full_scans`

  The total number of full table scans by statements for the host.

28.4.3.6 The host_summary_by_statement_type and x$host_summary_by_statement_type Views

These views summarize information about statements executed, grouped by host and statement type. By default, rows are sorted by host and descending total latency.

The `host_summary_by_statement_type` and `x$host_summary_by_statement_type` views have these columns:

- `host`

  The host from which the client connected. Rows for which the `HOST` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `statement`

  The final component of the statement event name.

- `total`

  The total number of occurrences of the statement event for the host.

- `total_latency`

  The total wait time of timed occurrences of the statement event for the host.

- `max_latency`

  The maximum single wait time of timed occurrences of the statement event for the host.

- `lock_latency`

  The total time waiting for locks by timed occurrences of the statement event for the host.

- `cpu_latency`

  The time spent on CPU for the current thread.

- `rows_sent`

  The total number of rows returned by occurrences of the statement event for the host.

- `rows_examined`

  The total number of rows read from storage engines by occurrences of the statement event for the host.

- `rows_affected`

  The total number of rows affected by occurrences of the statement event for the host.

- `full_scans`

  The total number of full table scans by occurrences of the statement event for the host.

28.4.3.7 The innodb_buffer_stats_by_schema and x$innodb_buffer_stats_by_schema Views

These views summarize the information in the `INFORMATION_SCHEMA INNODB_BUFFER_PAGE` table, grouped by schema. By default, rows are sorted by descending buffer size.

Warning

Querying views that access the `INNODB_BUFFER_PAGE` table can affect performance. Do not query these views on a production system unless you are aware of the performance impact and have determined it to be acceptable. To avoid impacting performance on a production system, reproduce the issue you want to investigate and query buffer pool statistics on a test instance.

The `innodb_buffer_stats_by_schema` and `x$innodb_buffer_stats_by_schema` views have these columns:

- `object_schema`

  The schema name for the object, or `InnoDB System` if the table belongs to the `InnoDB` storage engine.

- `allocated`

  The total number of bytes allocated for the schema.

- `data`

  The total number of data bytes allocated for the schema.

- `pages`

  The total number of pages allocated for the schema.

- `pages_hashed`

  The total number of hashed pages allocated for the schema.

- `pages_old`

  The total number of old pages allocated for the schema.

- `rows_cached`

  The total number of cached rows for the schema.

28.4.3.8 The innodb_buffer_stats_by_table and x$innodb_buffer_stats_by_table Views

These views summarize the information in the `INFORMATION_SCHEMA INNODB_BUFFER_PAGE` table, grouped by schema and table. By default, rows are sorted by descending buffer size.

Warning

Querying views that access the `INNODB_BUFFER_PAGE` table can affect performance. Do not query these views on a production system unless you are aware of the performance impact and have determined it to be acceptable. To avoid impacting performance on a production system, reproduce the issue you want to investigate and query buffer pool statistics on a test instance.

The `innodb_buffer_stats_by_table` and `x$innodb_buffer_stats_by_table` views have these columns:

- `object_schema`

The schema name for the object, or `InnoDB System` if the table belongs to the `InnoDB` storage engine.

- `object_name`

  The table name.

- `allocated`

  The total number of bytes allocated for the table.

- `data`

  The number of data bytes allocated for the table.

- `pages`

  The total number of pages allocated for the table.

- `pages_hashed`

  The number of hashed pages allocated for the table.

- `pages_old`

  The number of old pages allocated for the table.

- `rows_cached`

  The number of cached rows for the table.

28.4.3.9 The innodb_lock_waits and x$innodb_lock_waits Views

These views summarize the `InnoDB` locks that transactions are waiting for. By default, rows are sorted by descending lock age.

The `innodb_lock_waits` and `x$innodb_lock_waits` views have these columns:

- `wait_started`

  The time at which the lock wait started.

- `wait_age`

  How long the lock has been waited for, as a `TIME` value.

- `wait_age_secs`

  How long the lock has been waited for, in seconds.

- `locked_table_schema`

  The schema that contains the locked table.

- `locked_table_name`

  The name of the locked table.

- `locked_table_partition`

  The name of the locked partition, if any; `NULL` otherwise.

- `locked_table_subpartition`

The name of the locked subpartition, if any; `NULL` otherwise.

- `locked_index`

  The name of the locked index.

- `locked_type`

  The type of the waiting lock.

- `waiting_trx_id`

  The ID of the waiting transaction.

- `waiting_trx_started`

  The time at which the waiting transaction started.

- `waiting_trx_age`

  How long the waiting transaction has been waiting, as a `TIME` value.

- `waiting_trx_rows_locked`

  The number of rows locked by the waiting transaction.

- `waiting_trx_rows_modified`

  The number of rows modified by the waiting transaction.

- `waiting_pid`

  The processlist ID of the waiting transaction.

- `waiting_query`

  The statement that is waiting for the lock.

- `waiting_lock_id`

  The ID of the waiting lock.

- `waiting_lock_mode`

  The mode of the waiting lock.

- `blocking_trx_id`

  The ID of the transaction that is blocking the waiting lock.

- `blocking_pid`

  The processlist ID of the blocking transaction.

- `blocking_query`

  The statement the blocking transaction is executing. This field reports NULL if the session that issued the blocking query becomes idle. For more information, see Identifying a Blocking Query After the Issuing Session Becomes Idle.

- `blocking_lock_id`

  The ID of the lock that is blocking the waiting lock.

- `blocking_lock_mode`

  The mode of the lock that is blocking the waiting lock.

- `blocking_trx_started`

  The time at which the blocking transaction started.

- `blocking_trx_age`

  How long the blocking transaction has been executing, as a `TIME` value.

- `blocking_trx_rows_locked`

  The number of rows locked by the blocking transaction.

- `blocking_trx_rows_modified`

  The number of rows modified by the blocking transaction.

- `sql_kill_blocking_query`

  The `KILL` statement to execute to kill the blocking statement.

- `sql_kill_blocking_connection`

  The `KILL` statement to execute to kill the session running the blocking statement.

28.4.3.10 The io_by_thread_by_latency and x$io_by_thread_by_latency Views

These views summarize I/O consumers to display time waiting for I/O, grouped by thread. By default, rows are sorted by descending total I/O latency.

The `io_by_thread_by_latency` and `x$io_by_thread_by_latency` views have these columns:

- `user`

  For foreground threads, the account associated with the thread. For background threads, the thread name.

- `total`

  The total number of I/O events for the thread.

- `total_latency`

  The total wait time of timed I/O events for the thread.

- `min_latency`

  The minimum single wait time of timed I/O events for the thread.

- `avg_latency`

  The average wait time per timed I/O event for the thread.

- `max_latency`

  The maximum single wait time of timed I/O events for the thread.

- `thread_id`

  The thread ID.

- `processlist_id`

  For foreground threads, the processlist ID of the thread. For background threads, `NULL`.

28.4.3.11 The io_global_by_file_by_bytes and x$io_global_by_file_by_bytes Views

These views summarize global I/O consumers to display amount of I/O, grouped by file. By default, rows are sorted by descending total I/O (bytes read and written).

The `io_global_by_file_by_bytes` and `x$io_global_by_file_by_bytes` views have these columns:

- `file`

  The file path name.

- `count_read`

  The total number of read events for the file.

- `total_read`

  The total number of bytes read from the file.

- `avg_read`

  The average number of bytes per read from the file.

- `count_write`

  The total number of write events for the file.

- `total_written`

  The total number of bytes written to the file.

- `avg_write`

  The average number of bytes per write to the file.

- `total`

  The total number of bytes read and written for the file.

- `write_pct`

  The percentage of total bytes of I/O that were writes.

28.4.3.12 The io_global_by_file_by_latency and x$io_global_by_file_by_latency Views

These views summarize global I/O consumers to display time waiting for I/O, grouped by file. By default, rows are sorted by descending total latency.

The `io_global_by_file_by_latency` and `x$io_global_by_file_by_latency` views have these columns:

- `file`

  The file path name.

- `total`

  The total number of I/O events for the file.

- `total_latency`

  The total wait time of timed I/O events for the file.

- `count_read`

  The total number of read I/O events for the file.

- `read_latency`

  The total wait time of timed read I/O events for the file.

- `count_write`

  The total number of write I/O events for the file.

- `write_latency`

  The total wait time of timed write I/O events for the file.

- `count_misc`

  The total number of other I/O events for the file.

- `misc_latency`

  The total wait time of timed other I/O events for the file.

28.4.3.13 The io_global_by_wait_by_bytes and x$io_global_by_wait_by_bytes Views

These views summarize global I/O consumers to display amount of I/O and time waiting for I/O, grouped by event. By default, rows are sorted by descending total I/O (bytes read and written).

The `io_global_by_wait_by_bytes` and `x$io_global_by_wait_by_bytes` views have these columns:

- `event_name`

  The I/O event name, with the `wait/io/file/` prefix stripped.

- `total`

  The total number of occurrences of the I/O event.

- `total_latency`

  The total wait time of timed occurrences of the I/O event.

- `min_latency`

  The minimum single wait time of timed occurrences of the I/O event.

- `avg_latency`

  The average wait time per timed occurrence of the I/O event.

- `max_latency`

  The maximum single wait time of timed occurrences of the I/O event.

- `count_read`

  The number of read requests for the I/O event.

- `total_read`

  The number of bytes read for the I/O event.

- `avg_read`

  The average number of bytes per read for the I/O event.

- `count_write`

  The number of write requests for the I/O event.

- `total_written`

  The number of bytes written for the I/O event.

- `avg_written`

  The average number of bytes per write for the I/O event.

- `total_requested`

  The total number of bytes read and written for the I/O event.

28.4.3.14 The io_global_by_wait_by_latency and x$io_global_by_wait_by_latency Views

These views summarize global I/O consumers to display amount of I/O and time waiting for I/O, grouped by event. By default, rows are sorted by descending total latency.

The `io_global_by_wait_by_latency` and `x$io_global_by_wait_by_latency` views have these columns:

- `event_name`

  The I/O event name, with the `wait/io/file/` prefix stripped.

- `total`

  The total number of occurrences of the I/O event.

- `total_latency`

  The total wait time of timed occurrences of the I/O event.

- `avg_latency`

  The average wait time per timed occurrence of the I/O event.

- `max_latency`

  The maximum single wait time of timed occurrences of the I/O event.

- `read_latency`

  The total wait time of timed read occurrences of the I/O event.

- `write_latency`

  The total wait time of timed write occurrences of the I/O event.

- `misc_latency`

  The total wait time of timed other occurrences of the I/O event.

- `count_read`

  The number of read requests for the I/O event.

- `total_read`

  The number of bytes read for the I/O event.

- `avg_read`

  The average number of bytes per read for the I/O event.

- `count_write`

  The number of write requests for the I/O event.

- `total_written`

  The number of bytes written for the I/O event.

- `avg_written`

  The average number of bytes per write for the I/O event.

28.4.3.15 The latest_file_io and x$latest_file_io Views

These views summarize file I/O activity, grouped by file and thread. By default, rows are sorted with most recent I/O first.

The `latest_file_io` and `x$latest_file_io` views have these columns:

- `thread`

  For foreground threads, the account associated with the thread (and port number for TCP/IP connections). For background threads, the thread name and thread ID

- `file`

  The file path name.

- `latency`

  The wait time of the file I/O event.

- `operation`

  The type of operation.

- `requested`

  The number of data bytes requested for the file I/O event.

28.4.3.16 The memory_by_host_by_current_bytes and x$memory_by_host_by_current_bytes Views

These views summarize memory use, grouped by host. By default, rows are sorted by descending amount of memory used.

The `memory_by_host_by_current_bytes` and `x$memory_by_host_by_current_bytes` views have these columns:

- `host`

The host from which the client connected. Rows for which the `HOST` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `current_count_used`

  The current number of allocated memory blocks that have not been freed yet for the host.

- `current_allocated`

  The current number of allocated bytes that have not been freed yet for the host.

- `current_avg_alloc`

  The current number of allocated bytes per memory block for the host.

- `current_max_alloc`

  The largest single current memory allocation in bytes for the host.

- `total_allocated`

  The total memory allocation in bytes for the host.

28.4.3.17 The memory_by_thread_by_current_bytes and x$memory_by_thread_by_current_bytes Views

These views summarize memory use, grouped by thread. By default, rows are sorted by descending amount of memory used.

The `memory_by_thread_by_current_bytes` and `x$memory_by_thread_by_current_bytes` views have these columns:

- `thread_id`

  The thread ID.

- `user`

  The thread user or thread name.

- `current_count_used`

  The current number of allocated memory blocks that have not been freed yet for the thread.

- `current_allocated`

  The current number of allocated bytes that have not been freed yet for the thread.

- `current_avg_alloc`

  The current number of allocated bytes per memory block for the thread.

- `current_max_alloc`

  The largest single current memory allocation in bytes for the thread.

- `total_allocated`

  The total memory allocation in bytes for the thread.

28.4.3.18 The memory_by_user_by_current_bytes and x$memory_by_user_by_current_bytes Views

These views summarize memory use, grouped by user. By default, rows are sorted by descending amount of memory used.

The `memory_by_user_by_current_bytes` and `x$memory_by_user_by_current_bytes` views have these columns:

- `user`

  The client user name. Rows for which the `USER` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `current_count_used`

  The current number of allocated memory blocks that have not been freed yet for the user.

- `current_allocated`

  The current number of allocated bytes that have not been freed yet for the user.

- `current_avg_alloc`

  The current number of allocated bytes per memory block for the user.

- `current_max_alloc`

  The largest single current memory allocation in bytes for the user.

- `total_allocated`

  The total memory allocation in bytes for the user.

28.4.3.19 The memory_global_by_current_bytes and x$memory_global_by_current_bytes Views

These views summarize memory use, grouped by allocation type (that is, by event). By default, rows are sorted by descending amount of memory used.

The `memory_global_by_current_bytes` and `x$memory_global_by_current_bytes` views have these columns:

- `event_name`

  The memory event name.

- `current_count`

  The total number of occurrences of the event.

- `current_alloc`

  The current number of allocated bytes that have not been freed yet for the event.

- `current_avg_alloc`

  The current number of allocated bytes per memory block for the event.

- `high_count`

  The high-water mark for number of memory blocks allocated for the event.

- `high_alloc`

The high-water mark for number of bytes allocated for the event.

- `high_avg_alloc`

  The high-water mark for average number of bytes per memory block allocated for the event.

28.4.3.20 The memory_global_total and x$memory_global_total Views

These views summarize total memory use within the server.

The `memory_global_total` and `x$memory_global_total` views have these columns:

- `total_allocated`

  The total bytes of memory allocated within the server.

28.4.3.21 The metrics View

This view summarizes MySQL server metrics to show variable names, values, types, and whether they are enabled. By default, rows are sorted by variable type and name.

The `metrics` view includes this information:

- Global status variables from the Performance Schema `global_status` table
- `InnoDB` metrics from the `INFORMATION_SCHEMA` `INNODB_METRICS` table
- Current and total memory allocation, based on the Performance Schema memory instrumentation
- The current time (human readable and Unix timestamp formats)

There is some duplication of information between the `global_status` and `INNODB_METRICS` tables, which the `metrics` view eliminates.

The `metrics` view has these columns:

- `Variable_name`

  The metric name. The metric type determines the source from which the name is taken:

  - For global status variables: The `VARIABLE_NAME` column of the `global_status` table
  - For `InnoDB` metrics: The `NAME` column of the `INNODB_METRICS` table
  - For other metrics: A view-provided descriptive string

- `Variable_value`

  The metric value. The metric type determines the source from which the value is taken:

  - For global status variables: The `VARIABLE_VALUE` column of the `global_status` table
  - For `InnoDB` metrics: The `COUNT` column of the `INNODB_METRICS` table
  - For memory metrics: The relevant column from the Performance Schema `memory_summary_global_by_event_name` table
  - For the current time: The value of `NOW(3)` or `UNIX_TIMESTAMP(NOW(3))`

- `Type`

  The metric type:

  - For global status variables: `Global Status`
  - For `InnoDB` metrics: `InnoDB Metrics - %`, where `%` is replaced by the value of the `SUBSYSTEM` column of the `INNODB_METRICS` table
  - For memory metrics: `Performance Schema`
  - For the current time: `System Time`
- `Enabled`

  Whether the metric is enabled:

  - For global status variables: `YES`
  - For `InnoDB` metrics: `YES` if the `STATUS` column of the `INNODB_METRICS` table is `enabled`, `NO` otherwise
  - For memory metrics: `NO`, `YES`, or `PARTIAL` (currently, `PARTIAL` occurs only for memory metrics and indicates that not all `memory/%` instruments are enabled; Performance Schema memory instruments are always enabled)
  - For the current time: `YES`

28.4.3.22 The processlist and x$processlist Views

The MySQL process list indicates the operations currently being performed by the set of threads executing within the server. The `processlist` and `x$processlist` views summarize process information. They provide more complete information than the `SHOW PROCESSLIST` statement and the `INFORMATION_SCHEMA PROCESSLIST` table, and are also nonblocking. By default, rows are sorted by descending process time and descending wait time. For a comparison of process information sources, see Sources of Process Information.

The column descriptions here are brief. For additional information, see the description of the Performance Schema `threads` table at Section 27.12.21.7, “The threads Table”.

The `processlist` and `x$processlist` views have these columns:

- `thd_id`

  The thread ID.
- `conn_id`

  The connection ID.
- `user`

  The thread user or thread name.
- `db`

  The default database for the thread, or `NULL` if there is none.
- `command`

  For foreground threads, the type of command the thread is executing on behalf of the client, or `Sleep` if the session is idle.
- `state`

  An action, event, or state that indicates what the thread is doing.

- `time`

  The time in seconds that the thread has been in its current state.

- `current_statement`

  The statement the thread is executing, or `NULL` if it is not executing any statement.

- `execution_engine`

  The query execution engine. The value is either `PRIMARY` or `SECONDARY`. For use with MySQL Database Service and HeatWave, where the `PRIMARY` engine is `InnoDB` and `SECONDARY` engine is HeatWave (`RAPID`). For MySQL Community Edition Server, MySQL Enterprise Edition Server (on-premise), and MySQL Database Service without HeatWave, the value is always `PRIMARY`. This column was added in MySQL 8.0.29.

- `statement_latency`

  How long the statement has been executing.

- `progress`

  The percentage of work completed for stages that support progress reporting. See Section 28.3, “sys Schema Progress Reporting”.

- `lock_latency`

  The time spent waiting for locks by the current statement.

- `cpu_latency`

  The time spent on CPU for the current thread.

- `rows_examined`

  The number of rows read from storage engines by the current statement.

- `rows_sent`

  The number of rows returned by the current statement.

- `rows_affected`

  The number of rows affected by the current statement.

- `tmp_tables`

  The number of internal in-memory temporary tables created by the current statement.

- `tmp_disk_tables`

  The number of internal on-disk temporary tables created by the current statement.

- `full_scan`

  The number of full table scans performed by the current statement.

- `last_statement`

  The last statement executed by the thread, if there is no currently executing statement or wait.

- `last_statement_latency`

  How long the last statement executed.

- `current_memory`

  The number of bytes allocated by the thread.

- `last_wait`

  The name of the most recent wait event for the thread.

- `last_wait_latency`

  The wait time of the most recent wait event for the thread.

- `source`

  The source file and line number containing the instrumented code that produced the event.

- `trx_latency`

  The wait time of the current transaction for the thread.

- `trx_state`

  The state for the current transaction for the thread.

- `trx_autocommit`

  Whether autocommit mode was enabled when the current transaction started.

- `pid`

  The client process ID.

- `program_name`

  The client program name.

28.4.3.23 The ps_check_lost_instrumentation View

This view returns information about lost Performance Schema instruments, to indicate whether the Performance Schema is unable to monitor all runtime data.

The `ps_check_lost_instrumentation` view has these columns:

- `variable_name`

  The Performance Schema status variable name indicating which type of instrument was lost.

- `variable_value`

  The number of instruments lost.

28.4.3.24 The schema_auto_increment_columns View

This view indicates which tables have `AUTO_INCREMENT` columns and provides information about those columns, such as the current and maximum column values and the usage ratio (ratio of used to possible values). By default, rows are sorted by descending usage ratio and maximum column value.

Tables in these schemas are excluded from view output: `mysql`, `sys`, `INFORMATION_SCHEMA`, `performance_schema`.

The `schema_auto_increment_columns` view has these columns:

- `table_schema`

The schema that contains the table.

- `table_name`

  The table that contains the `AUTO_INCREMENT` column.

- `column_name`

  The name of the `AUTO_INCREMENT` column.

- `data_type`

  The data type of the column.

- `column_type`

  The column type of the column, which is the data type plus possibly other information. For example, for a column with a `bigint(20) unsigned` column type, the data type is just `bigint`.

- `is_signed`

  Whether the column type is signed.

- `is_unsigned`

  Whether the column type is unsigned.

- `max_value`

  The maximum permitted value for the column.

- `auto_increment`

  The current `AUTO_INCREMENT` value for the column.

- `auto_increment_ratio`

  The ratio of used to permitted values for the column. This indicates how much of the sequence of values is “used up.”

28.4.3.25 The schema_index_statistics and x$schema_index_statistics Views

These views provide index statistics. By default, rows are sorted by descending total index latency.

The `schema_index_statistics` and `x$schema_index_statistics` views have these columns:

- `table_schema`

  The schema that contains the table.

- `table_name`

  The table that contains the index.

- `index_name`

  The name of the index.

- `rows_selected`

  The total number of rows read using the index.

- `select_latency`

The total wait time of timed reads using the index.

- `rows_inserted`

  The total number of rows inserted into the index.

- `insert_latency`

  The total wait time of timed inserts into the index.

- `rows_updated`

  The total number of rows updated in the index.

- `update_latency`

  The total wait time of timed updates in the index.

- `rows_deleted`

  The total number of rows deleted from the index.

- `delete_latency`

  The total wait time of timed deletes from the index.

28.4.3.26 The schema_object_overview View

This view summarizes the types of objects within each schema. By default, rows are sorted by schema and object type.

Note

For MySQL instances with a large number of objects, this view might take a long time to execute.

The `schema_object_overview` view has these columns:

- `db`

  The schema name.

- `object_type`

  The object type: `BASE TABLE`, `INDEX (index_type)`, `EVENT`, `FUNCTION`, `PROCEDURE`, `TRIGGER`, `VIEW`.

- `count`

  The number of objects in the schema of the given type.

28.4.3.27 The schema_redundant_indexes and x$schema_flattened_keys Views

The `schema_redundant_indexes` view displays indexes that duplicate other indexes or are made redundant by them. The `x$schema_flattened_keys` view is a helper view for `schema_redundant_indexes`.

In the following column descriptions, the dominant index is the one that makes the redundant index redundant.

The `schema_redundant_indexes` view has these columns:

- `table_schema`

The schema that contains the table.

- `table_name`

  The table that contains the index.

- `redundant_index_name`

  The name of the redundant index.

- `redundant_index_columns`

  The names of the columns in the redundant index.

- `redundant_index_non_unique`

  The number of nonunique columns in the redundant index.

- `dominant_index_name`

  The name of the dominant index.

- `dominant_index_columns`

  The names of the columns in the dominant index.

- `dominant_index_non_unique`

  The number of nonunique columns in the dominant index.

- `subpart_exists`

  Whether the index indexes only part of a column.

- `sql_drop_index`

  The statement to execute to drop the redundant index.

The `x$schema_flattened_keys` view has these columns:

- `table_schema`

  The schema that contains the table.

- `table_name`

  The table that contains the index.

- `index_name`

  An index name.

- `non_unique`

  The number of nonunique columns in the index.

- `subpart_exists`

  Whether the index indexes only part of a column.

- `index_columns`

  The name of the columns in the index.

28.4.3.28 The schema_table_lock_waits and x$schema_table_lock_waits Views

These views display which sessions are blocked waiting on metadata locks, and what is blocking them.

The column descriptions here are brief. For additional information, see the description of the Performance Schema `metadata_locks` table at Section 27.12.13.3, “The metadata_locks Table”.

The `schema_table_lock_waits` and `x$schema_table_lock_waits` views have these columns:

- `object_schema`

  The schema containing the object to be locked.

- `object_name`

  The name of the instrumented object.

- `waiting_thread_id`

  The thread ID of the thread that is waiting for the lock.

- `waiting_pid`

  The processlist ID of the thread that is waiting for the lock.

- `waiting_account`

  The account associated with the session that is waiting for the lock.

- `waiting_lock_type`

  The type of the waiting lock.

- `waiting_lock_duration`

  How long the waiting lock has been waiting.

- `waiting_query`

  The statement that is waiting for the lock.

- `waiting_query_secs`

  How long the statement has been waiting, in seconds.

- `waiting_query_rows_affected`

  The number of rows affected by the statement.

- `waiting_query_rows_examined`

  The number of rows read from storage engines by the statement.

- `blocking_thread_id`

  The thread ID of the thread that is blocking the waiting lock.

- `blocking_pid`

  The processlist ID of the thread that is blocking the waiting lock.

- `blocking_account`

  The account associated with the thread that is blocking the waiting lock.

- `blocking_lock_type`

  The type of lock that is blocking the waiting lock.

- `blocking_lock_duration`

  How long the blocking lock has been held.

- `sql_kill_blocking_query`

  The `KILL` statement to execute to kill the blocking statement.

- `sql_kill_blocking_connection`

  The `KILL` statement to execute to kill the session running the blocking statement.

28.4.3.29 The schema_table_statistics and x$schema_table_statistics Views

These views summarize table statistics. By default, rows are sorted by descending total wait time (tables with most contention first).

These views user a helper view, `x$ps_schema_table_statistics_io`.

The `schema_table_statistics` and `x$schema_table_statistics` views have these columns:

- `table_schema`

  The schema that contains the table.

- `table_name`

  The table name.

- `total_latency`

  The total wait time of timed I/O events for the table.

- `rows_fetched`

  The total number of rows read from the table.

- `fetch_latency`

  The total wait time of timed read I/O events for the table.

- `rows_inserted`

  The total number of rows inserted into the table.

- `insert_latency`

  The total wait time of timed insert I/O events for the table.

- `rows_updated`

  The total number of rows updated in the table.

- `update_latency`

  The total wait time of timed update I/O events for the table.

- `rows_deleted`

  The total number of rows deleted from the table.

- `delete_latency`

  The total wait time of timed delete I/O events for the table.

- `io_read_requests`

  The total number of read requests for the table.

- `io_read`

  The total number of bytes read from the table.

- `io_read_latency`

  The total wait time of reads from the table.

- `io_write_requests`

  The total number of write requests for the table.

- `io_write`

  The total number of bytes written to the table.

- `io_write_latency`

  The total wait time of writes to the table.

- `io_misc_requests`

  The total number of miscellaneous I/O requests for the table.

- `io_misc_latency`

  The total wait time of miscellaneous I/O requests for the table.

28.4.3.30 The schema_table_statistics_with_buffer and x $schema_table_statistics_with_buffer Views

These views summarize table statistics, including `InnoDB` buffer pool statistics. By default, rows are sorted by descending total wait time (tables with most contention first).

These views user a helper view, `x$ps_schema_table_statistics_io`.

The `schema_table_statistics_with_buffer` and `x $schema_table_statistics_with_buffer` views have these columns:

- `table_schema`

  The schema that contains the table.

- `table_name`

  The table name.

- `rows_fetched`

  The total number of rows read from the table.

- `fetch_latency`

  The total wait time of timed read I/O events for the table.

- `rows_inserted`

The total number of rows inserted into the table.

- `insert_latency`

  The total wait time of timed insert I/O events for the table.

- `rows_updated`

  The total number of rows updated in the table.

- `update_latency`

  The total wait time of timed update I/O events for the table.

- `rows_deleted`

  The total number of rows deleted from the table.

- `delete_latency`

  The total wait time of timed delete I/O events for the table.

- `io_read_requests`

  The total number of read requests for the table.

- `io_read`

  The total number of bytes read from the table.

- `io_read_latency`

  The total wait time of reads from the table.

- `io_write_requests`

  The total number of write requests for the table.

- `io_write`

  The total number of bytes written to the table.

- `io_write_latency`

  The total wait time of writes to the table.

- `io_misc_requests`

  The total number of miscellaneous I/O requests for the table.

- `io_misc_latency`

  The total wait time of miscellaneous I/O requests for the table.

- `innodb_buffer_allocated`

  The total number of `InnoDB` buffer bytes allocated for the table.

- `innodb_buffer_data`

  The total number of `InnoDB` data bytes allocated for the table.

- `innodb_buffer_free`

The total number of InnoDB nondata bytes allocated for the table (innodb_buffer_allocated − innodb_buffer_data).

- innodb_buffer_pages

  The total number of InnoDB pages allocated for the table.

- innodb_buffer_pages_hashed

  The total number of InnoDB hashed pages allocated for the table.

- innodb_buffer_pages_old

  The total number of InnoDB old pages allocated for the table.

- innodb_buffer_rows_cached

  The total number of InnoDB cached rows for the table.

28.4.3.31 The schema_tables_with_full_table_scans and x $schema_tables_with_full_table_scans Views

These views display which tables are being accessed with full table scans. By default, rows are sorted by descending rows scanned.

The schema_tables_with_full_table_scans and x $schema_tables_with_full_table_scans views have these columns:

- object_schema

  The schema name.

- object_name

  The table name.

- rows_full_scanned

  The total number of rows scanned by full scans of the table.

- latency

  The total wait time of full scans of the table.

28.4.3.32 The schema_unused_indexes View

These views display indexes for which there are no events, which indicates that they are not being used. By default, rows are sorted by schema and table.

This view is most useful when the server has been up and processing long enough that its workload is representative. Otherwise, presence of an index in this view may not be meaningful.

The schema_unused_indexes view has these columns:

- object_schema

  The schema name.

- object_name

  The table name.

- `index_name`

  The unused index name.

28.4.3.33 The session and x$session Views

These views are similar to `processlist` and `x$processlist`, but they filter out background processes to display only user sessions. For descriptions of the columns, see Section 28.4.3.22, “The processlist and x$processlist Views”.

28.4.3.34 The session_ssl_status View

For each connection, this view displays the SSL version, cipher, and count of reused SSL sessions.

The `session_ssl_status` view has these columns:

- `thread_id`

  The thread ID for the connection.

- `ssl_version`

  The version of SSL used for the connection.

- `ssl_cipher`

  The SSL cipher used for the connection.

- `ssl_sessions_reused`

  The number of reused SSL sessions for the connection.

28.4.3.35 The statement_analysis and x$statement_analysis Views

These views list normalized statements with aggregated statistics. The content mimics the MySQL Enterprise Monitor Query Analysis view. By default, rows are sorted by descending total latency.

The `statement_analysis` and `x$statement_analysis` views have these columns:

- `query`

  The normalized statement string.

- `db`

  The default database for the statement, or `NULL` if there is none.

- `full_scan`

  The total number of full table scans performed by occurrences of the statement.

- `exec_count`

  The total number of times the statement has executed.

- `err_count`

  The total number of errors produced by occurrences of the statement.

- `warn_count`

  The total number of warnings produced by occurrences of the statement.

- `total_latency`

The total wait time of timed occurrences of the statement.

- `max_latency`

  The maximum single wait time of timed occurrences of the statement.

- `avg_latency`

  The average wait time per timed occurrence of the statement.

- `lock_latency`

  The total time waiting for locks by timed occurrences of the statement.

- `cpu_latency`

  The time spent on CPU for the current thread.

- `rows_sent`

  The total number of rows returned by occurrences of the statement.

- `rows_sent_avg`

  The average number of rows returned per occurrence of the statement.

- `rows_examined`

  The total number of rows read from storage engines by occurrences of the statement.

- `rows_examined_avg`

  The average number of rows read from storage engines per occurrence of the statement.

- `rows_affected`

  The total number of rows affected by occurrences of the statement.

- `rows_affected_avg`

  The average number of rows affected per occurrence of the statement.

- `tmp_tables`

  The total number of internal in-memory temporary tables created by occurrences of the statement.

- `tmp_disk_tables`

  The total number of internal on-disk temporary tables created by occurrences of the statement.

- `rows_sorted`

  The total number of rows sorted by occurrences of the statement.

- `sort_merge_passes`

  The total number of sort merge passes by occurrences of the statement.

- `max_controlled_memory`

  The maximum amount of controlled memory (bytes) used by the statement.

  This column was added in MySQL 8.0.31

- `max_total_memory`

  The maximum amount of memory (bytes) used by the statement.

  This column was added in MySQL 8.0.31

- `digest`

  The statement digest.

- `first_seen`

  The time at which the statement was first seen.

- `last_seen`

  The time at which the statement was most recently seen.

28.4.3.36 The statements_with_errors_or_warnings and x$statements_with_errors_or_warnings Views

These views display normalized statements that have produced errors or warnings. By default, rows are sorted by descending error and warning counts.

The `statements_with_errors_or_warnings` and `x$statements_with_errors_or_warnings` views have these columns:

- `query`

  The normalized statement string.

- `db`

  The default database for the statement, or `NULL` if there is none.

- `exec_count`

  The total number of times the statement has executed.

- `errors`

  The total number of errors produced by occurrences of the statement.

- `error_pct`

  The percentage of statement occurrences that produced errors.

- `warnings`

  The total number of warnings produced by occurrences of the statement.

- `warning_pct`

  The percentage of statement occurrences that produced warnings.

- `first_seen`

  The time at which the statement was first seen.

- `last_seen`

  The time at which the statement was most recently seen.

- `digest`

The statement digest.

28.4.3.37 The statements_with_full_table_scans and x$statements_with_full_table_scans Views

These views display normalized statements that have done full table scans. By default, rows are sorted by descending percentage of time a full scan was done and descending total latency.

The `statements_with_full_table_scans` and `x$statements_with_full_table_scans` views have these columns:

- `query`

  The normalized statement string.

- `db`

  The default database for the statement, or `NULL` if there is none.

- `exec_count`

  The total number of times the statement has executed.

- `total_latency`

  The total wait time of timed statement events for the statement.

- `no_index_used_count`

  The total number of times no index was used to scan the table.

- `no_good_index_used_count`

  The total number of times no good index was used to scan the table.

- `no_index_used_pct`

  The percentage of the time no index was used to scan the table.

- `rows_sent`

  The total number of rows returned from the table.

- `rows_examined`

  The total number of rows read from the storage engine for the table.

- `rows_sent_avg`

  The average number of rows returned from the table.

- `rows_examined_avg`

  The average number of rows read from the storage engine for the table.

- `first_seen`

  The time at which the statement was first seen.

- `last_seen`

  The time at which the statement was most recently seen.

- `digest`

  The statement digest.

28.4.3.38 The statements_with_runtimes_in_95th_percentile and x$statements_with_runtimes_in_95th_percentile Views

These views list statements with runtimes in the 95th percentile. By default, rows are sorted by descending average latency.

Both views use two helper views, `x$ps_digest_avg_latency_distribution` and `x$ps_digest_95th_percentile_by_avg_us`.

The `statements_with_runtimes_in_95th_percentile` and `x$statements_with_runtimes_in_95th_percentile` views have these columns:

- `query`

  The normalized statement string.

- `db`

  The default database for the statement, or `NULL` if there is none.

- `full_scan`

  The total number of full table scans performed by occurrences of the statement.

- `exec_count`

  The total number of times the statement has executed.

- `err_count`

  The total number of errors produced by occurrences of the statement.

- `warn_count`

  The total number of warnings produced by occurrences of the statement.

- `total_latency`

  The total wait time of timed occurrences of the statement.

- `max_latency`

  The maximum single wait time of timed occurrences of the statement.

- `avg_latency`

  The average wait time per timed occurrence of the statement.

- `rows_sent`

  The total number of rows returned by occurrences of the statement.

- `rows_sent_avg`

  The average number of rows returned per occurrence of the statement.

- `rows_examined`

  The total number of rows read from storage engines by occurrences of the statement.

- `rows_examined_avg`

  The average number of rows read from storage engines per occurrence of the statement.

- `first_seen`

  The time at which the statement was first seen.

- `last_seen`

  The time at which the statement was most recently seen.

- `digest`

  The statement digest.

28.4.3.39 The statements_with_sorting and x$statements_with_sorting Views

These views list normalized statements that have performed sorts. By default, rows are sorted by descending total latency.

The `statements_with_sorting` and `x$statements_with_sorting` views have these columns:

- `query`

  The normalized statement string.

- `db`

  The default database for the statement, or `NULL` if there is none.

- `exec_count`

  The total number of times the statement has executed.

- `total_latency`

  The total wait time of timed occurrences of the statement.

- `sort_merge_passes`

  The total number of sort merge passes by occurrences of the statement.

- `avg_sort_merges`

  The average number of sort merge passes per occurrence of the statement.

- `sorts_using_scans`

  The total number of sorts using table scans by occurrences of the statement.

- `sort_using_range`

  The total number of sorts using range accesses by occurrences of the statement.

- `rows_sorted`

  The total number of rows sorted by occurrences of the statement.

- `avg_rows_sorted`

  The average number of rows sorted per occurrence of the statement.

- `first_seen`

The time at which the statement was first seen.

- `last_seen`

  The time at which the statement was most recently seen.

- `digest`

  The statement digest.

28.4.3.40 The statements_with_temp_tables and x$statements_with_temp_tables Views

These views list normalized statements that have used temporary tables. By default, rows are sorted by descending number of on-disk temporary tables used and descending number of in-memory temporary tables used.

The `statements_with_temp_tables` and `x$statements_with_temp_tables` views have these columns:

- `query`

  The normalized statement string.

- `db`

  The default database for the statement, or `NULL` if there is none.

- `exec_count`

  The total number of times the statement has executed.

- `total_latency`

  The total wait time of timed occurrences of the statement.

- `memory_tmp_tables`

  The total number of internal in-memory temporary tables created by occurrences of the statement.

- `disk_tmp_tables`

  The total number of internal on-disk temporary tables created by occurrences of the statement.

- `avg_tmp_tables_per_query`

  The average number of internal temporary tables created per occurrence of the statement.

- `tmp_tables_to_disk_pct`

  The percentage of internal in-memory temporary tables that were converted to on-disk tables.

- `first_seen`

  The time at which the statement was first seen.

- `last_seen`

  The time at which the statement was most recently seen.

- `digest`

  The statement digest.

28.4.3.41 The user_summary and x$user_summary Views

These views summarize statement activity, file I/O, and connections, grouped by user. By default, rows are sorted by descending total latency.

The `user_summary` and `x$user_summary` views have these columns:

- `user`

  The client user name. Rows for which the `USER` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `statements`

  The total number of statements for the user.

- `statement_latency`

  The total wait time of timed statements for the user.

- `statement_avg_latency`

  The average wait time per timed statement for the user.

- `table_scans`

  The total number of table scans for the user.

- `file_ios`

  The total number of file I/O events for the user.

- `file_io_latency`

  The total wait time of timed file I/O events for the user.

- `current_connections`

  The current number of connections for the user.

- `total_connections`

  The total number of connections for the user.

- `unique_hosts`

  The number of distinct hosts from which connections for the user have originated.

- `current_memory`

  The current amount of allocated memory for the user.

- `total_memory_allocated`

  The total amount of allocated memory for the user.

28.4.3.42 The user_summary_by_file_io and x$user_summary_by_file_io Views

These views summarize file I/O, grouped by user. By default, rows are sorted by descending total file I/O latency.

The `user_summary_by_file_io` and `x$user_summary_by_file_io` views have these columns:

- `user`

  The client user name. Rows for which the `USER` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `ios`

  The total number of file I/O events for the user.

- `io_latency`

  The total wait time of timed file I/O events for the user.

28.4.3.43 The user_summary_by_file_io_type and x$user_summary_by_file_io_type Views

These views summarize file I/O, grouped by user and event type. By default, rows are sorted by user and descending total latency.

The `user_summary_by_file_io_type` and `x$user_summary_by_file_io_type` views have these columns:

- `user`

  The client user name. Rows for which the `USER` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `event_name`

  The file I/O event name.

- `total`

  The total number of occurrences of the file I/O event for the user.

- `latency`

  The total wait time of timed occurrences of the file I/O event for the user.

- `max_latency`

  The maximum single wait time of timed occurrences of the file I/O event for the user.

28.4.3.44 The user_summary_by_stages and x$user_summary_by_stages Views

These views summarize stages, grouped by user. By default, rows are sorted by user and descending total stage latency.

The `user_summary_by_stages` and `x$user_summary_by_stages` views have these columns:

- `user`

  The client user name. Rows for which the `USER` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `event_name`

  The stage event name.

- `total`

The total number of occurrences of the stage event for the user.

- `total_latency`

  The total wait time of timed occurrences of the stage event for the user.

- `avg_latency`

  The average wait time per timed occurrence of the stage event for the user.

28.4.3.45 The user_summary_by_statement_latency and x$user_summary_by_statement_latency Views

These views summarize overall statement statistics, grouped by user. By default, rows are sorted by descending total latency.

The `user_summary_by_statement_latency` and `x$user_summary_by_statement_latency` views have these columns:

- `user`

  The client user name. Rows for which the `USER` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `total`

  The total number of statements for the user.

- `total_latency`

  The total wait time of timed statements for the user.

- `max_latency`

  The maximum single wait time of timed statements for the user.

- `lock_latency`

  The total time waiting for locks by timed statements for the user.

- `cpu_latency`

  The time spent on CPU for the current thread.

- `rows_sent`

  The total number of rows returned by statements for the user.

- `rows_examined`

  The total number of rows read from storage engines by statements for the user.

- `rows_affected`

  The total number of rows affected by statements for the user.

- `full_scans`

  The total number of full table scans by statements for the user.

28.4.3.46 The user_summary_by_statement_type and x$user_summary_by_statement_type Views

These views summarize information about statements executed, grouped by user and statement type. By default, rows are sorted by user and descending total latency.

The `user_summary_by_statement_type` and `x$user_summary_by_statement_type` views have these columns:

- `user`

  The client user name. Rows for which the `USER` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `statement`

  The final component of the statement event name.

- `total`

  The total number of occurrences of the statement event for the user.

- `total_latency`

  The total wait time of timed occurrences of the statement event for the user.

- `max_latency`

  The maximum single wait time of timed occurrences of the statement event for the user.

- `lock_latency`

  The total time waiting for locks by timed occurrences of the statement event for the user.

- `cpu_latency`

  The time spent on CPU for the current thread.

- `rows_sent`

  The total number of rows returned by occurrences of the statement event for the user.

- `rows_examined`

  The total number of rows read from storage engines by occurrences of the statement event for the user.

- `rows_affected`

  The total number of rows affected by occurrences of the statement event for the user.

- `full_scans`

  The total number of full table scans by occurrences of the statement event for the user.

28.4.3.47 The version View

This view provides the current `sys` schema and MySQL server versions.

Note

As of MySQL 8.0.18, this view is deprecated and subject to removal in a future MySQL version. Applications that use it should be migrated to use an alternative instead. For example, use the `VERSION()` function to retrieve the MySQL server version.

The `version` view has these columns:

- `sys_version`

  The `sys` schema version.

- `mysql_version`

  The MySQL server version.

28.4.3.48 The wait_classes_global_by_avg_latency and x$wait_classes_global_by_avg_latency Views

These views summarize wait class average latencies, grouped by event class. By default, rows are sorted by descending average latency. Idle events are ignored.

An event class is determined by stripping from the event name everything after the first three components. For example, the class for `wait/io/file/sql/slow_log` is `wait/io/file`.

The `wait_classes_global_by_avg_latency` and `x$wait_classes_global_by_avg_latency` views have these columns:

- `event_class`

  The event class.

- `total`

  The total number of occurrences of events in the class.

- `total_latency`

  The total wait time of timed occurrences of events in the class.

- `min_latency`

  The minimum single wait time of timed occurrences of events in the class.

- `avg_latency`

  The average wait time per timed occurrence of events in the class.

- `max_latency`

  The maximum single wait time of timed occurrences of events in the class.

28.4.3.49 The wait_classes_global_by_latency and x$wait_classes_global_by_latency Views

These views summarize wait class total latencies, grouped by event class. By default, rows are sorted by descending total latency. Idle events are ignored.

An event class is determined by stripping from the event name everything after the first three components. For example, the class for `wait/io/file/sql/slow_log` is `wait/io/file`.

The `wait_classes_global_by_latency` and `x$wait_classes_global_by_latency` views have these columns:

- `event_class`

  The event class.

- `total`

The total number of occurrences of events in the class.

- `total_latency`

  The total wait time of timed occurrences of events in the class.

- `min_latency`

  The minimum single wait time of timed occurrences of events in the class.

- `avg_latency`

  The average wait time per timed occurrence of events in the class.

- `max_latency`

  The maximum single wait time of timed occurrences of events in the class.

28.4.3.50 The waits_by_host_by_latency and x$waits_by_host_by_latency Views

These views summarize wait events, grouped by host and event. By default, rows are sorted by host and descending total latency. Idle events are ignored.

The `waits_by_host_by_latency` and `x$waits_by_host_by_latency` views have these columns:

- `host`

  The host from which the connection originated.

- `event`

  The event name.

- `total`

  The total number of occurrences of the event for the host.

- `total_latency`

  The total wait time of timed occurrences of the event for the host.

- `avg_latency`

  The average wait time per timed occurrence of the event for the host.

- `max_latency`

  The maximum single wait time of timed occurrences of the event for the host.

28.4.3.51 The waits_by_user_by_latency and x$waits_by_user_by_latency Views

These views summarize wait events, grouped by user and event. By default, rows are sorted by user and descending total latency. Idle events are ignored.

The `waits_by_user_by_latency` and `x$waits_by_user_by_latency` views have these columns:

- `user`

  The user associated with the connection.

- `event`

The event name.

- `total`

  The total number of occurrences of the event for the user.

- `total_latency`

  The total wait time of timed occurrences of the event for the user.

- `avg_latency`

  The average wait time per timed occurrence of the event for the user.

- `max_latency`

  The maximum single wait time of timed occurrences of the event for the user.

28.4.3.52 The waits_global_by_latency and x$waits_global_by_latency Views

These views summarize wait events, grouped by event. By default, rows are sorted by descending total latency. Idle events are ignored.

The `waits_global_by_latency` and `x$waits_global_by_latency` views have these columns:

- `events`

  The event name.

- `total`

  The total number of occurrences of the event.

- `total_latency`

  The total wait time of timed occurrences of the event.

- `avg_latency`

  The average wait time per timed occurrence of the event.

- `max_latency`

  The maximum single wait time of timed occurrences of the event.

28.4.4 sys Schema Stored Procedures

The following sections describe `sys` schema stored procedures.

28.4.4.1 The create_synonym_db() Procedure

Given a schema name, this procedure creates a synonym schema containing views that refer to all the tables and views in the original schema. This can be used, for example, to create a shorter name by which to refer to a schema with a long name (such as `info` rather than `INFORMATION_SCHEMA`).

Parameters

- `in_db_name VARCHAR(64)`: The name of the schema for which to create the synonym.
- `in_synonym VARCHAR(64)`: The name to use for the synonym schema. This schema must not already exist.

Example

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| world              |
+--------------------+
mysql> CALL sys.create_synonym_db('INFORMATION_SCHEMA', 'info');
+--------------------------------------+
| summary                              |
+--------------------------------------+
| Created 63 views in the info database |
+--------------------------------------+
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| info               |
| mysql              |
| performance_schema |
| sys                |
| world              |
+--------------------+
mysql> SHOW FULL TABLES FROM info;
+---------------------------------------+------------+
| Tables_in_info                        | Table_type |
+---------------------------------------+------------+
| character_sets                        | VIEW       |
| collation_character_set_applicability | VIEW       |
| collations                            | VIEW       |
| column_privileges                     | VIEW       |
| columns                               | VIEW       |
...
```

28.4.4.2 The diagnostics() Procedure

Creates a report of the current server status for diagnostic purposes.

This procedure disables binary logging during its execution by manipulating the session value of the `sql_log_bin` system variable. That is a restricted operation, so the procedure requires privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

Data collected for `diagnostics()` includes this information:

- Information from the `metrics` view (see Section 28.4.3.21, “The metrics View”)
- Information from other relevant `sys` schema views, such as the one that determines queries in the 95th percentile
- Information from the `ndbinfo` schema, if the MySQL server is part of NDB Cluster
- Replication status (both source and replica)

Some of the sys schema views are calculated as initial (optional), overall, and delta values:

- The initial view is the content of the view at the start of the `diagnostics()` procedure. This output is the same as the start values used for the delta view. The initial view is included if the `diagnostics.include_raw` configuration option is `ON`.
- The overall view is the content of the view at the end of the `diagnostics()` procedure. This output is the same as the end values used for the delta view. The overall view is always included.

- The delta view is the difference from the beginning to the end of procedure execution. The minimum and maximum values are the minimum and maximum values from the end view, respectively. They do not necessarily reflect the minimum and maximum values in the monitored period. Except for the `metrics` view, the delta is calculated only between the first and last outputs.

Parameters

- `in_max_runtime INT UNSIGNED`: The maximum data collection time in seconds. Use `NULL` to collect data for the default of 60 seconds. Otherwise, use a value greater than 0.
- `in_interval INT UNSIGNED`: The sleep time between data collections in seconds. Use `NULL` to sleep for the default of 30 seconds. Otherwise, use a value greater than 0.
- `in_auto_config ENUM('current', 'medium', 'full')`: The Performance Schema configuration to use. Permitted values are:
  - `current`: Use the current instrument and consumer settings.
  - `medium`: Enable some instruments and consumers.
  - `full`: Enable all instruments and consumers.

Note

The more instruments and consumers enabled, the more impact on MySQL server performance. Be careful with the `medium` setting and especially the `full` setting, which has a large performance impact.

Use of the `medium` or `full` setting requires the `SUPER` privilege.

If a setting other than `current` is chosen, the current settings are restored at the end of the procedure.

Configuration Options

`diagnostics()` operation can be modified using the following configuration options or their corresponding user-defined variables (see Section 28.4.2.1, “The sys_config Table”):

- `debug`, `@sys.debug`

  If this option is `ON`, produce debugging output. The default is `OFF`.

- `diagnostics.allow_i_s_tables`, `@sys.diagnostics.allow_i_s_tables`

  If this option is `ON`, the `diagnostics()` procedure is permitted to perform table scans on the Information Schema `TABLES` table. This can be expensive if there are many tables. The default is `OFF`.

- `diagnostics.include_raw`, `@sys.diagnostics.include_raw`

  If this option is `ON`, the `diagnostics()` procedure output includes the raw output from querying the `metrics` view. The default is `OFF`.

- `statement_truncate_len`, `@sys.statement_truncate_len`

  The maximum length of statements returned by the `format_statement()` function. Longer statements are truncated to this length. The default is 64.

Example

Create a diagnostics report that starts an iteration every 30 seconds and runs for at most 120 seconds using the current Performance Schema settings:

```
mysql> CALL sys.diagnostics(120, 30, 'current');
```

To capture the output from the `diagnostics()` procedure in a file as it runs, use the `mysql` client `tee filename` and `notee` commands (see Section 4.5.1.2, “mysql Client Commands”):

```
mysql> tee diag.out;
mysql> CALL sys.diagnostics(120, 30, 'current');
mysql> notee;
```

28.4.4.3 The execute_prepared_stmt() Procedure

Given an SQL statement as a string, executes it as a prepared statement. The prepared statement is deallocated after execution, so it is not subject to reuse. Thus, this procedure is useful primarily for executing dynamic statements on a one-time basis.

This procedure uses `sys_execute_prepared_stmt` as the prepared statement name. If that statement name exists when the procedure is called, its previous content is destroyed.

Parameters

- `in_query LONGTEXT CHARACTER SET utf8mb3`: The statement string to execute.

Configuration Options

`execute_prepared_stmt()` operation can be modified using the following configuration options or their corresponding user-defined variables (see Section 28.4.2.1, “The sys_config Table”):

- `debug`, `@sys.debug`

  If this option is `ON`, produce debugging output. The default is `OFF`.

Example

```
mysql> CALL sys.execute_prepared_stmt('SELECT COUNT(*) FROM mysql.user');
+----------+
| COUNT(*) |
+----------+
|       15 |
+----------+
```

28.4.4.4 The ps_setup_disable_background_threads() Procedure

Disables Performance Schema instrumentation for all background threads. Produces a result set indicating how many background threads were disabled. Already disabled threads do not count.

Parameters

None.

Example

```
mysql> CALL sys.ps_setup_disable_background_threads();
+--------------------------------+
| summary                        |
+--------------------------------+
| Disabled 24 background threads |
+--------------------------------+
```

28.4.4.5 The ps_setup_disable_consumer() Procedure

Disables Performance Schema consumers with names that contain the argument. Produces a result set indicating how many consumers were disabled. Already disabled consumers do not count.

Parameters

- `consumer VARCHAR(128)`: The value used to match consumer names, which are identified by using `%consumer%` as an operand for a `LIKE` pattern match.

  A value of `''` matches all consumers.

Example

Disable all statement consumers:

```
mysql> CALL sys.ps_setup_disable_consumer('statement');
+----------------------+
| summary              |
+----------------------+
| Disabled 4 consumers |
+----------------------+
```

28.4.4.6 The ps_setup_disable_instrument() Procedure

Disables Performance Schema instruments with names that contain the argument. Produces a result set indicating how many instruments were disabled. Already disabled instruments do not count.

Parameters

- `in_pattern VARCHAR(128)`: The value used to match instrument names, which are identified by using `%in_pattern%` as an operand for a `LIKE` pattern match.

  A value of `''` matches all instruments.

Example

Disable a specific instrument:

```
mysql> CALL sys.ps_setup_disable_instrument('wait/lock/metadata/sql/mdl');
+-----------------------+
| summary               |
+-----------------------+
| Disabled 1 instrument |
+-----------------------+
```

Disable all mutex instruments:

```
mysql> CALL sys.ps_setup_disable_instrument('mutex');
+--------------------------+
| summary                  |
+--------------------------+
| Disabled 177 instruments |
+--------------------------+
```

28.4.4.7 The ps_setup_disable_thread() Procedure

Given a connection ID, disables Performance Schema instrumentation for the thread. Produces a result set indicating how many threads were disabled. Already disabled threads do not count.

Parameters

- `in_connection_id BIGINT`: The connection ID. This is a value of the type given in the `PROCESSLIST_ID` column of the Performance Schema `threads` table or the `Id` column of `SHOW PROCESSLIST` output.

Example

Disable a specific connection by its connection ID:

```
mysql> CALL sys.ps_setup_disable_thread(225);
+-------------------+
| summary           |
+-------------------+
| Disabled 1 thread |
+-------------------+
```

Disable the current connection:

```
mysql> CALL sys.ps_setup_disable_thread(CONNECTION_ID());
+-------------------+
| summary           |
+-------------------+
| Disabled 1 thread |
+-------------------+
```

28.4.4.8 The ps_setup_enable_background_threads() Procedure

Enables Performance Schema instrumentation for all background threads. Produces a result set indicating how many background threads were enabled. Already enabled threads do not count.

Parameters

None.

Example

```
mysql> CALL sys.ps_setup_enable_background_threads();
+------------------------------+
| summary                      |
+------------------------------+
| Enabled 24 background threads |
+------------------------------+
```

28.4.4.9 The ps_setup_enable_consumer() Procedure

Enables Performance Schema consumers with names that contain the argument. Produces a result set indicating how many consumers were enabled. Already enabled consumers do not count.

Parameters

- `consumer VARCHAR(128)`: The value used to match consumer names, which are identified by using `%consumer%` as an operand for a `LIKE` pattern match.

  A value of `''` matches all consumers.

Example

Enable all statement consumers:

```
mysql> CALL sys.ps_setup_enable_consumer('statement');
+---------------------+
| summary             |
+---------------------+
| Enabled 4 consumers |
+---------------------+
```

28.4.4.10 The ps_setup_enable_instrument() Procedure

Enables Performance Schema instruments with names that contain the argument. Produces a result set indicating how many instruments were enabled. Already enabled instruments do not count.

Parameters

- `in_pattern VARCHAR(128)`: The value used to match instrument names, which are identified by using `%in_pattern%` as an operand for a `LIKE` pattern match.

  A value of `''` matches all instruments.

Example

Enable a specific instrument:

```
mysql> CALL sys.ps_setup_enable_instrument('wait/lock/metadata/sql/mdl');
+---------------------+
| summary             |
+---------------------+
| Enabled 1 instrument |
+---------------------+
```

Enable all mutex instruments:

```
mysql> CALL sys.ps_setup_enable_instrument('mutex');
+-------------------------+
| summary                 |
+-------------------------+
| Enabled 177 instruments |
+-------------------------+
```

28.4.4.11 The ps_setup_enable_thread() Procedure

Given a connection ID, enables Performance Schema instrumentation for the thread. Produces a result set indicating how many threads were enabled. Already enabled threads do not count.

Parameters

- `in_connection_id BIGINT`: The connection ID. This is a value of the type given in the `PROCESSLIST_ID` column of the Performance Schema `threads` table or the `Id` column of `SHOW PROCESSLIST` output.

Example

Enable a specific connection by its connection ID:

```
mysql> CALL sys.ps_setup_enable_thread(225);
+------------------+
| summary          |
+------------------+
| Enabled 1 thread |
+------------------+
```

Enable the current connection:

```
mysql> CALL sys.ps_setup_enable_thread(CONNECTION_ID());
+------------------+
| summary          |
+------------------+
| Enabled 1 thread |
+------------------+
```

28.4.4.12 The ps_setup_reload_saved() Procedure

Reloads a Performance Schema configuration saved earlier within the same session using `ps_setup_save()`. For more information, see the description of `ps_setup_save()`.

This procedure disables binary logging during its execution by manipulating the session value of the `sql_log_bin` system variable. That is a restricted operation, so the procedure requires privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

Parameters

None.

28.4.4.13 The ps_setup_reset_to_default() Procedure

Resets the Performance Schema configuration to its default settings.

Parameters

- `in_verbose BOOLEAN`: Whether to display information about each setup stage during procedure execution. This includes the SQL statements executed.

Example

```
mysql> CALL sys.ps_setup_reset_to_default(TRUE)\G
*************************** 1. row ***************************
status: Resetting: setup_actors
DELETE
FROM performance_schema.setup_actors
WHERE NOT (HOST = '%' AND USER = '%' AND ROLE = '%')

*************************** 1. row ***************************
status: Resetting: setup_actors
INSERT IGNORE INTO performance_schema.setup_actors
VALUES ('%', '%', '%')

...
```

28.4.4.14 The ps_setup_save() Procedure

Saves the current Performance Schema configuration. This enables you to alter the configuration temporarily for debugging or other purposes, then restore it to the previous state by invoking the `ps_setup_reload_saved()` procedure.

To prevent other simultaneous calls to save the configuration, `ps_setup_save()` acquires an advisory lock named `sys.ps_setup_save` by calling the `GET_LOCK()` function. `ps_setup_save()` takes a timeout parameter to indicate how many seconds to wait if the lock already exists (which indicates that some other session has a saved configuration outstanding). If the timeout expires without obtaining the lock, `ps_setup_save()` fails.

It is intended you call `ps_setup_reload_saved()` later within the *same* session as `ps_setup_save()` because the configuration is saved in `TEMPORARY` tables. `ps_setup_save()` drops the temporary tables and releases the lock. If you end your session without invoking `ps_setup_save()`, the tables and lock disappear automatically.

This procedure disables binary logging during its execution by manipulating the session value of the `sql_log_bin` system variable. That is a restricted operation, so the procedure requires privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

Parameters

- `in_timeout INT`: How many seconds to wait to obtain the `sys.ps_setup_save` lock. A negative timeout value means infinite timeout.

Example

```
mysql> CALL sys.ps_setup_save(10);
```

```
... make Performance Schema configuration changes ...

mysql> CALL sys.ps_setup_reload_saved();
```

28.4.4.15 The ps_setup_show_disabled() Procedure

Displays all currently disabled Performance Schema configuration.

Parameters

- `in_show_instruments BOOLEAN`: Whether to display disabled instruments. This might be a long list.
- `in_show_threads BOOLEAN`: Whether to display disabled threads.

Example

```
mysql> CALL sys.ps_setup_show_disabled(TRUE, TRUE);
+----------------------------+
| performance_schema_enabled |
+----------------------------+
|                          1 |
+----------------------------+

+---------------+
| enabled_users |
+---------------+
| '%'@'%'       |
+---------------+

+-------------+---------------------+---------+-------+
| object_type | objects             | enabled | timed |
+-------------+---------------------+---------+-------+
| EVENT       | mysql.%             | NO      | NO    |
| EVENT       | performance_schema.% | NO      | NO    |
| EVENT       | information_schema.% | NO      | NO    |
| FUNCTION    | mysql.%             | NO      | NO    |
| FUNCTION    | performance_schema.% | NO      | NO    |
| FUNCTION    | information_schema.% | NO      | NO    |
| PROCEDURE   | mysql.%             | NO      | NO    |
| PROCEDURE   | performance_schema.% | NO      | NO    |
| PROCEDURE   | information_schema.% | NO      | NO    |
| TABLE       | mysql.%             | NO      | NO    |
| TABLE       | performance_schema.% | NO      | NO    |
| TABLE       | information_schema.% | NO      | NO    |
| TRIGGER     | mysql.%             | NO      | NO    |
| TRIGGER     | performance_schema.% | NO      | NO    |
| TRIGGER     | information_schema.% | NO      | NO    |
+-------------+---------------------+---------+-------+

...
```

28.4.4.16 The ps_setup_show_disabled_consumers() Procedure

Displays all currently disabled Performance Schema consumers.

Parameters

None.

Example

```
mysql> CALL sys.ps_setup_show_disabled_consumers();
+---------------------------------+
| disabled_consumers              |
+---------------------------------+
| events_stages_current           |
| events_stages_history           |
```

```
| events_stages_history_long       |
| events_statements_history        |
| events_statements_history_long   |
| events_transactions_history      |
| events_transactions_history_long |
| events_waits_current             |
| events_waits_history             |
| events_waits_history_long        |
+----------------------------------+
```

28.4.4.17 The ps_setup_show_disabled_instruments() Procedure

Displays all currently disabled Performance Schema instruments. This might be a long list.

Parameters

None.

Example

```
mysql> CALL sys.ps_setup_show_disabled_instruments()\G
*************************** 1. row ***************************
disabled_instruments: wait/synch/mutex/sql/TC_LOG_MMAP::LOCK_tc
               timed: NO
*************************** 2. row ***************************
disabled_instruments: wait/synch/mutex/sql/THD::LOCK_query_plan
               timed: NO
*************************** 3. row ***************************
disabled_instruments: wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_commit
               timed: NO
...
```

28.4.4.18 The ps_setup_show_enabled() Procedure

Displays all currently enabled Performance Schema configuration.

Parameters

- `in_show_instruments BOOLEAN`: Whether to display enabled instruments. This might be a long list.
- `in_show_threads BOOLEAN`: Whether to display enabled threads.

Example

```
mysql> CALL sys.ps_setup_show_enabled(FALSE, FALSE);
+---------------------------+
| performance_schema_enabled |
+---------------------------+
|                         1 |
+---------------------------+
1 row in set (0.01 sec)

+---------------+
| enabled_users |
+---------------+
| '%'@'%'       |
+---------------+
1 row in set (0.01 sec)

+-------------+---------+---------+-------+
| object_type | objects | enabled | timed |
+-------------+---------+---------+-------+
| EVENT       | %.%     | YES     | YES   |
| FUNCTION    | %.%     | YES     | YES   |
| PROCEDURE   | %.%     | YES     | YES   |
| TABLE       | %.%     | YES     | YES   |
| TRIGGER     | %.%     | YES     | YES   |
```

```
+------------+---------+---------+-------+
5 rows in set (0.02 sec)

+----------------------------+
| enabled_consumers          |
+----------------------------+
| events_statements_current  |
| events_statements_history  |
| events_transactions_current |
| events_transactions_history |
| global_instrumentation     |
| statements_digest          |
| thread_instrumentation     |
+----------------------------+
```

28.4.4.19 The ps_setup_show_enabled_consumers() Procedure

Displays all currently enabled Performance Schema consumers.

Parameters

None.

Example

```
mysql> CALL sys.ps_setup_show_enabled_consumers();
+----------------------------+
| enabled_consumers          |
+----------------------------+
| events_statements_current  |
| events_statements_history  |
| events_transactions_current |
| events_transactions_history |
| global_instrumentation     |
| statements_digest          |
| thread_instrumentation     |
+----------------------------+
```

28.4.4.20 The ps_setup_show_enabled_instruments() Procedure

Displays all currently enabled Performance Schema instruments. This might be a long list.

Parameters

None.

Example

```
mysql> CALL sys.ps_setup_show_enabled_instruments()\G
*************************** 1. row ***************************
enabled_instruments: wait/io/file/sql/map
              timed: YES
*************************** 2. row ***************************
enabled_instruments: wait/io/file/sql/binlog
              timed: YES
*************************** 3. row ***************************
enabled_instruments: wait/io/file/sql/binlog_cache
              timed: YES
...
```

28.4.4.21 The ps_statement_avg_latency_histogram() Procedure

Displays a textual histogram graph of the average latency values across all normalized statements tracked within the Performance Schema `events_statements_summary_by_digest` table.

This procedure can be used to display a very high-level picture of the latency distribution of statements running within this MySQL instance.

Parameters

None.

Example

The histogram output in statement units. For example, `* = 2 units` in the histogram legend means that each `*` character represents 2 statements.

```
mysql> CALL sys.ps_statement_avg_latency_histogram()\G
*************************** 1. row ***************************
Performance Schema Statement Digest Average Latency Histogram:

  . = 1 unit
  * = 2 units
  # = 3 units

(0 - 66ms)     88  | ##############################
(66 - 133ms)   14  | ..............
(133 - 199ms)  4   | ....
(199 - 265ms)  5   | **
(265 - 332ms)  1   | .
(332 - 398ms)  0   |
(398 - 464ms)  1   | .
(464 - 531ms)  0   |
(531 - 597ms)  0   |
(597 - 663ms)  0   |
(663 - 730ms)  0   |
(730 - 796ms)  0   |
(796 - 863ms)  0   |
(863 - 929ms)  0   |
(929 - 995ms)  0   |
(995 - 1062ms) 0   |

  Total Statements: 114; Buckets: 16; Bucket Size: 66 ms;
```

28.4.4.22 The ps_trace_statement_digest() Procedure

Traces all Performance Schema instrumentation for a specific statement digest.

If you find a statement of interest within the Performance Schema `events_statements_summary_by_digest` table, specify its `DIGEST` column MD5 value to this procedure and indicate the polling duration and interval. The result is a report of all statistics tracked within Performance Schema for that digest for the interval.

The procedure also attempts to execute `EXPLAIN` for the longest running example of the digest during the interval. This attempt might fail because the Performance Schema truncates long `SQL_TEXT` values. Consequently, `EXPLAIN` fails, due to parse errors.

This procedure disables binary logging during its execution by manipulating the session value of the `sql_log_bin` system variable. That is a restricted operation, so the procedure requires privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

Parameters

- `in_digest VARCHAR(32)`: The statement digest identifier to analyze.
- `in_runtime INT`: How long to run the analysis in seconds.
- `in_interval DECIMAL(2,2)`: The analysis interval in seconds (which can be fractional) at which to try to take snapshots.
- `in_start_fresh BOOLEAN`: Whether to truncate the Performance Schema `events_statements_history_long` and `events_stages_history_long` tables before starting.
- `in_auto_enable BOOLEAN`: Whether to automatically enable required consumers.

Example

```
mysql> CALL sys.ps_trace_statement_digest('891ec6860f98ba46d89dd20b0c03652c', 10, 0.1, TRUE, TRUE);
+--------------------+
| SUMMARY STATISTICS |
+--------------------+
| SUMMARY STATISTICS |
+--------------------+
1 row in set (9.11 sec)

+------------+-----------+-----------+-----------+---------------+------------+------------+
| executions | exec_time | lock_time | rows_sent | rows_examined | tmp_tables | full_scans |
+------------+-----------+-----------+-----------+---------------+------------+------------+
|         21 | 4.11 ms   | 2.00 ms   |         0 |            21 |          0 |          0 |
+------------+-----------+-----------+-----------+---------------+------------+------------+
1 row in set (9.11 sec)

+----------------------------------------+-------+-----------+
| event_name                             | count | latency   |
+----------------------------------------+-------+-----------+
| stage/sql/statistics                   |    16 | 546.92 us |
| stage/sql/freeing items                |    18 | 520.11 us |
| stage/sql/init                         |    51 | 466.80 us |
...
| stage/sql/cleaning up                  |    18 | 11.92 us  |
| stage/sql/executing                    |    16 | 6.95 us   |
+----------------------------------------+-------+-----------+
17 rows in set (9.12 sec)

+--------------------------+
| LONGEST RUNNING STATEMENT |
+--------------------------+
| LONGEST RUNNING STATEMENT |
+--------------------------+
1 row in set (9.16 sec)

+-----------+-----------+-----------+-----------+---------------+------------+-----------+
| thread_id | exec_time | lock_time | rows_sent | rows_examined | tmp_tables | full_scan |
+-----------+-----------+-----------+-----------+---------------+------------+-----------+
|    166646 | 618.43 us | 1.00 ms   |         0 |             1 |          0 |         0 |
+-----------+-----------+-----------+-----------+---------------+------------+-----------+
1 row in set (9.16 sec)

# Truncated for clarity...
+-----------------------------------------------------------------+
| sql_text                                                        |
+-----------------------------------------------------------------+
| select hibeventhe0_.id as id1382_, hibeventhe0_.createdTime ... |
+-----------------------------------------------------------------+
1 row in set (9.17 sec)

+----------------------------------------+-----------+
| event_name                             | latency   |
+----------------------------------------+-----------+
| stage/sql/init                         | 8.61 us   |
| stage/sql/init                         | 331.07 ns |
...
| stage/sql/freeing items                | 30.46 us  |
| stage/sql/cleaning up                  | 662.13 ns |
+----------------------------------------+-----------+
18 rows in set (9.23 sec)

+----+-------------+--------------+-------+---------------+-----------+---------+-------------+------+-
| id | select_type | table        | type  | possible_keys | key       | key_len | ref         | rows |
+----+-------------+--------------+-------+---------------+-----------+---------+-------------+------+-
|  1 | SIMPLE      | hibeventhe0_ | const | fixedTime     | fixedTime | 775     | const,const |    1 |
+----+-------------+--------------+-------+---------------+-----------+---------+-------------+------+-
1 row in set (9.27 sec)

Query OK, 0 rows affected (9.28 sec)
```

28.4.4.23 The ps_trace_thread() Procedure

Dumps all Performance Schema data for an instrumented thread to a `.dot` formatted graph file (for the DOT graph description language). Each result set returned from the procedure should be used for a complete graph.

This procedure disables binary logging during its execution by manipulating the session value of the `sql_log_bin` system variable. That is a restricted operation, so the procedure requires privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

Parameters

- `in_thread_id INT`: The thread to trace.
- `in_outfile VARCHAR(255)`: The name to use for the `.dot` output file.
- `in_max_runtime DECIMAL(20,2)`: The maximum number of seconds (which can be fractional) to collect data. Use `NULL` to collect data for the default of 60 seconds.
- `in_interval DECIMAL(20,2)`: The number of seconds (which can be fractional) to sleep between data collections. Use `NULL` to sleep for the default of 1 second.
- `in_start_fresh BOOLEAN`: Whether to reset all Performance Schema data before tracing.
- `in_auto_setup BOOLEAN`: Whether to disable all other threads and enable all instruments and consumers. This also resets the settings at the end of the run.
- `in_debug BOOLEAN`: Whether to include `file:lineno` information in the graph.

Example

```
mysql> CALL sys.ps_trace_thread(25, CONCAT('/tmp/stack-', REPLACE(NOW(), ' ', '-'), '.dot'), NULL, NULL, TR
+--------------------+
| summary            |
+--------------------+
| Disabled 1 thread |
+--------------------+
1 row in set (0.00 sec)

+-----------------------------------------+
| Info                                    |
+-----------------------------------------+
| Data collection starting for THREAD_ID = 25 |
+-----------------------------------------+
1 row in set (0.03 sec)

+------------------------------------------------------------+
| Info                                                       |
+------------------------------------------------------------+
| Stack trace written to /tmp/stack-2014-02-16-21:18:41.dot |
+------------------------------------------------------------+
1 row in set (60.07 sec)

+-------------------------------------------------------------------+
| Convert to PDF                                                    |
+-------------------------------------------------------------------+
| dot -Tpdf -o /tmp/stack_25.pdf /tmp/stack-2014-02-16-21:18:41.dot |
+-------------------------------------------------------------------+
1 row in set (60.07 sec)

+-------------------------------------------------------------------+
| Convert to PNG                                                    |
+-------------------------------------------------------------------+
| dot -Tpng -o /tmp/stack_25.png /tmp/stack-2014-02-16-21:18:41.dot |
+-------------------------------------------------------------------+
1 row in set (60.07 sec)

+------------------+
| summary          |
+------------------+
```

```
| Enabled 1 thread |
+------------------+
1 row in set (60.32 sec)
```

28.4.4.24 The ps_truncate_all_tables() Procedure

Truncates all Performance Schema summary tables, resetting all aggregated instrumentation as a snapshot. Produces a result set indicating how many tables were truncated.

Parameters

- `in_verbose BOOLEAN`: Whether to display each `TRUNCATE TABLE` statement before executing it.

Example

```
mysql> CALL sys.ps_truncate_all_tables(FALSE);
+--------------------+
| summary            |
+--------------------+
| Truncated 49 tables |
+--------------------+
```

28.4.4.25 The statement_performance_analyzer() Procedure

Creates a report of the statements running on the server. The views are calculated based on the overall and/or delta activity.

This procedure disables binary logging during its execution by manipulating the session value of the `sql_log_bin` system variable. That is a restricted operation, so the procedure requires privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

Parameters

- `in_action ENUM('snapshot', 'overall', 'delta', 'create_tmp', 'create_table', 'save', 'cleanup')`: The action to take. These values are permitted:
  - `snapshot`: Store a snapshot. The default is to make a snapshot of the current content of the Performance Schema `events_statements_summary_by_digest` table. By setting `in_table`, this can be overwritten to copy the content of the specified table. The snapshot is stored in the `sys` schema `tmp_digests` temporary table.
  - `overall`: Generate an analysis based on the content of the table specified by `in_table`. For the overall analysis, `in_table` can be `NOW()` to use a fresh snapshot. This overwrites an existing snapshot. Use `NULL` for `in_table` to use the existing snapshot. If `in_table` is `NULL` and no snapshot exists, a new snapshot is created. The `in_views` parameter and the `statement_performance_analyzer.limit` configuration option affect the operation of this procedure.
  - `delta`: Generate a delta analysis. The delta is calculated between the reference table specified by `in_table` and the snapshot, which must exist. This action uses the `sys` schema `tmp_digests_delta` temporary table. The `in_views` parameter and the `statement_performance_analyzer.limit` configuration option affect the operation of this procedure.
  - `create_table`: Create a regular table suitable for storing the snapshot for later use (for example, for calculating deltas).
  - `create_tmp`: Create a temporary table suitable for storing the snapshot for later use (for example, for calculating deltas).
  - `save`: Save the snapshot in the table specified by `in_table`. The table must exist and have the correct structure. If no snapshot exists, a new snapshot is created.

    - `cleanup`: Remove the temporary tables used for the snapshot and delta.

- `in_table VARCHAR(129)`: The table parameter used for some of the actions specified by the `in_action` parameter. Use the format *`db_name.tbl_name`* or *`tbl_name`* without using any backtick (`` ` ``) identifier-quoting characters. Periods (`.`) are not supported in database and table names.

  The meaning of the `in_table` value for each `in_action` value is detailed in the individual `in_action` value descriptions.

- `in_views SET ('with_runtimes_in_95th_percentile', 'analysis', 'with_errors_or_warnings', 'with_full_table_scans', 'with_sorting', 'with_temp_tables', 'custom')`: Which views to include. This parameter is a `SET` value, so it can contain multiple view names, separated by commas. The default is to include all views except `custom`. The following values are permitted:

  - `with_runtimes_in_95th_percentile`: Use the `statements_with_runtimes_in_95th_percentile` view.
  - `analysis`: Use the `statement_analysis` view.
  - `with_errors_or_warnings`: Use the `statements_with_errors_or_warnings` view.
  - `with_full_table_scans`: Use the `statements_with_full_table_scans` view.
  - `with_sorting`: Use the `statements_with_sorting` view.
  - `with_temp_tables`: Use the `statements_with_temp_tables` view.
  - `custom`: Use a custom view. This view must be specified using the `statement_performance_analyzer.view` configuration option to name a query or an existing view.

Configuration Options

`statement_performance_analyzer()` operation can be modified using the following configuration options or their corresponding user-defined variables (see Section 28.4.2.1, “The sys_config Table”):

- `debug`, `@sys.debug`

  If this option is `ON`, produce debugging output. The default is `OFF`.

- `statement_performance_analyzer.limit`, `@sys.statement_performance_analyzer.limit`

  The maximum number of rows to return for views that have no built-in limit. The default is 100.

- `statement_performance_analyzer.view`, `@sys.statement_performance_analyzer.view`

  The custom query or view to be used. If the option value contains a space, it is interpreted as a query. Otherwise, it must be the name of an existing view that queries the Performance Schema `events_statements_summary_by_digest` table. There cannot be any `LIMIT` clause in the query or view definition if the `statement_performance_analyzer.limit` configuration option is greater than 0. If specifying a view, use the same format as for the `in_table` parameter. The default is `NULL` (no custom view defined).

Example

To create a report with the queries in the 95th percentile since the last truncation of `events_statements_summary_by_digest` and with a one-minute delta period:

1. Create a temporary table to store the initial snapshot.
2. Create the initial snapshot.
3. Save the initial snapshot in the temporary table.
4. Wait one minute.
5. Create a new snapshot.
6. Perform analysis based on the new snapshot.
7. Perform analysis based on the delta between the initial and new snapshots.

```
mysql> CALL sys.statement_performance_analyzer('create_tmp', 'mydb.tmp_digests_ini', NULL);
Query OK, 0 rows affected (0.08 sec)

mysql> CALL sys.statement_performance_analyzer('snapshot', NULL, NULL);
Query OK, 0 rows affected (0.02 sec)

mysql> CALL sys.statement_performance_analyzer('save', 'mydb.tmp_digests_ini', NULL);
Query OK, 0 rows affected (0.00 sec)

mysql> DO SLEEP(60);
Query OK, 0 rows affected (1 min 0.00 sec)

mysql> CALL sys.statement_performance_analyzer('snapshot', NULL, NULL);
Query OK, 0 rows affected (0.02 sec)

mysql> CALL sys.statement_performance_analyzer('overall', NULL, 'with_runtimes_in_95th_percentile');
+----------------------------------------+
| Next Output                            |
+----------------------------------------+
| Queries with Runtime in 95th Percentile |
+----------------------------------------+
1 row in set (0.05 sec)

...

mysql> CALL sys.statement_performance_analyzer('delta', 'mydb.tmp_digests_ini', 'with_runtimes_in_95th_
+----------------------------------------+
| Next Output                            |
+----------------------------------------+
| Queries with Runtime in 95th Percentile |
+----------------------------------------+
1 row in set (0.03 sec)

...
```

Create an overall report of the 95th percentile queries and the top 10 queries with full table scans:

```
mysql> CALL sys.statement_performance_analyzer('snapshot', NULL, NULL);
Query OK, 0 rows affected (0.01 sec)

mysql> SET @sys.statement_performance_analyzer.limit = 10;
Query OK, 0 rows affected (0.00 sec)

mysql> CALL sys.statement_performance_analyzer('overall', NULL, 'with_runtimes_in_95th_percentile,with_
+----------------------------------------+
| Next Output                            |
+----------------------------------------+
| Queries with Runtime in 95th Percentile |
+----------------------------------------+
1 row in set (0.01 sec)

...

+-----------------------------------+
| Next Output                       |
+-----------------------------------+
| Top 10 Queries with Full Table Scan |
```

```
+-----------------------------------+
1 row in set (0.09 sec)

...
```

Use a custom view showing the top 10 queries sorted by total execution time, refreshing the view every minute using the `watch` command in Linux:

```
mysql> CREATE OR REPLACE VIEW mydb.my_statements AS
       SELECT sys.format_statement(DIGEST_TEXT) AS query,
              SCHEMA_NAME AS db,
              COUNT_STAR AS exec_count,
              sys.format_time(SUM_TIMER_WAIT) AS total_latency,
              sys.format_time(AVG_TIMER_WAIT) AS avg_latency,
              ROUND(IFNULL(SUM_ROWS_SENT / NULLIF(COUNT_STAR, 0), 0)) AS rows_sent_avg,
              ROUND(IFNULL(SUM_ROWS_EXAMINED / NULLIF(COUNT_STAR, 0), 0)) AS rows_examined_avg,
              ROUND(IFNULL(SUM_ROWS_AFFECTED / NULLIF(COUNT_STAR, 0), 0)) AS rows_affected_avg,
              DIGEST AS digest
         FROM performance_schema.events_statements_summary_by_digest
       ORDER BY SUM_TIMER_WAIT DESC;
Query OK, 0 rows affected (0.10 sec)

mysql> CALL sys.statement_performance_analyzer('create_table', 'mydb.digests_prev', NULL);
Query OK, 0 rows affected (0.10 sec)

$> watch -n 60 "mysql sys --table -e \"
> SET @sys.statement_performance_analyzer.view = 'mydb.my_statements';
> SET @sys.statement_performance_analyzer.limit = 10;
> CALL statement_performance_analyzer('snapshot', NULL, NULL);
> CALL statement_performance_analyzer('delta', 'mydb.digests_prev', 'custom');
> CALL statement_performance_analyzer('save', 'mydb.digests_prev', NULL);
> \""

Every 60.0s: mysql sys --table -e "        ...  Mon Dec 22 10:58:51 2014

+----------------------------------+
| Next Output                      |
+----------------------------------+
| Top 10 Queries Using Custom View |
+----------------------------------+
+------------------+-------+------------+---------------+-------------+---------------+-------------------
| query            | db    | exec_count | total_latency | avg_latency | rows_sent_avg | rows_examined_avg
+------------------+-------+------------+---------------+-------------+---------------+-------------------
...
```

28.4.4.26 The table_exists() Procedure

Tests whether a given table exists as a regular table, a `TEMPORARY` table, or a view. The procedure returns the table type in an `OUT` parameter. If both a temporary and a permanent table exist with the given name, `TEMPORARY` is returned.

Parameters

- `in_db VARCHAR(64)`: The name of the database in which to check for table existence.
- `in_table VARCHAR(64)`: The name of the table to check the existence of.
- `out_exists ENUM('', 'BASE TABLE', 'VIEW', 'TEMPORARY')`: The return value. This is an `OUT` parameter, so it must be a variable into which the table type can be stored. When the procedure returns, the variable has one of the following values to indicate whether the table exists:
  - `''`: The table name does not exist as a base table, `TEMPORARY` table, or view.
  - `BASE TABLE`: The table name exists as a base (permanent) table.
  - `VIEW`: The table name exists as a view.
  - `TEMPORARY`: The table name exists as a `TEMPORARY` table.

Example

```
mysql> CREATE DATABASE db1;
Query OK, 1 row affected (0.01 sec)

mysql> USE db1;
Database changed

mysql> CREATE TABLE t1 (id INT PRIMARY KEY);
Query OK, 0 rows affected (0.03 sec)

mysql> CREATE TABLE t2 (id INT PRIMARY KEY);
Query OK, 0 rows affected (0.20 sec)

mysql> CREATE view v_t1 AS SELECT * FROM t1;
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TEMPORARY TABLE t1 (id INT PRIMARY KEY);
Query OK, 0 rows affected (0.00 sec)

mysql> CALL sys.table_exists('db1', 't1', @exists); SELECT @exists;
Query OK, 0 rows affected (0.01 sec)

+-----------+
| @exists   |
+-----------+
| TEMPORARY |
+-----------+
1 row in set (0.00 sec)

mysql> CALL sys.table_exists('db1', 't2', @exists); SELECT @exists;
Query OK, 0 rows affected (0.02 sec)

+------------+
| @exists    |
+------------+
| BASE TABLE |
+------------+
1 row in set (0.00 sec)

mysql> CALL sys.table_exists('db1', 'v_t1', @exists); SELECT @exists;
Query OK, 0 rows affected (0.02 sec)

+---------+
| @exists |
+---------+
| VIEW    |
+---------+
1 row in set (0.00 sec)

mysql> CALL sys.table_exists('db1', 't3', @exists); SELECT @exists;
Query OK, 0 rows affected (0.00 sec)

+---------+
| @exists |
+---------+
|         |
+---------+
1 row in set (0.00 sec)
```

28.4.5 sys Schema Stored Functions

The following sections describe `sys` schema stored functions.

28.4.5.1 The extract_schema_from_file_name() Function

Given a file path name, returns the path component that represents the schema name. This function assumes that the file name lies within the schema directory. For this reason, it does not work with partitions or tables defined using their own `DATA_DIRECTORY` table option.

This function is useful when extracting file I/O information from the Performance Schema that includes file path names. It provides a convenient way to display schema names, which can be more easily understood than full path names, and can be used in joins against object schema names.

Parameters

- `path VARCHAR(512)`: The full path to a data file from which to extract the schema name.

Return Value

A `VARCHAR(64)` value.

Example

```
mysql> SELECT sys.extract_schema_from_file_name('/usr/local/mysql/data/world/City.ibd');
+-----------------------------------------------------------------------------+
| sys.extract_schema_from_file_name('/usr/local/mysql/data/world/City.ibd') |
+-----------------------------------------------------------------------------+
| world                                                                       |
+-----------------------------------------------------------------------------+
```

28.4.5.2 The extract_table_from_file_name() Function

Given a file path name, returns the path component that represents the table name.

This function is useful when extracting file I/O information from the Performance Schema that includes file path names. It provides a convenient way to display table names, which can be more easily understood than full path names, and can be used in joins against object table names.

Parameters

- `path VARCHAR(512)`: The full path to a data file from which to extract the table name.

Return Value

A `VARCHAR(64)` value.

Example

```
mysql> SELECT sys.extract_table_from_file_name('/usr/local/mysql/data/world/City.ibd');
+-----------------------------------------------------------------------------+
| sys.extract_table_from_file_name('/usr/local/mysql/data/world/City.ibd') |
+-----------------------------------------------------------------------------+
| City                                                                        |
+-----------------------------------------------------------------------------+
```

28.4.5.3 The format_bytes() Function

Note

As of MySQL 8.0.16, `format_bytes()` is deprecated and subject to removal in a future MySQL version. Applications that use it should be migrated to use the built-in `FORMAT_BYTES()` function instead. See Section 12.22, “Performance Schema Functions”

Given a byte count, converts it to human-readable format and returns a string consisting of a value and a units indicator. Depending on the size of the value, the units part is `bytes`, `KiB` (kibibytes), `MiB` (mebibytes), `GiB` (gibibytes), `TiB` (tebibytes), or `PiB` (pebibytes).

Parameters

- `bytes TEXT`: The byte count to format.

Return Value

A `TEXT` value.

Example

```
mysql> SELECT sys.format_bytes(512), sys.format_bytes(18446644073709551615);
+-----------------------+---------------------------------------+
| sys.format_bytes(512) | sys.format_bytes(18446644073709551615) |
+-----------------------+---------------------------------------+
| 512 bytes             | 16383.91 PiB                          |
+-----------------------+---------------------------------------+
```

28.4.5.4 The format_path() Function

Given a path name, returns the modified path name after replacing subpaths that match the values of the following system variables, in order:

```
datadir
tmpdir
slave_load_tmpdir or replica_load_tmpdir
innodb_data_home_dir
innodb_log_group_home_dir
innodb_undo_directory
basedir
```

A value that matches the value of system variable *`sysvar`* is replaced with the string `@@GLOBAL.`*`sysvar`*.

Parameters

- `path VARCHAR(512)`: The path name to format.

Return Value

A `VARCHAR(512) CHARACTER SET utf8mb3` value.

Example

```
mysql> SELECT sys.format_path('/usr/local/mysql/data/world/City.ibd');
+-----------------------------------------------------+
| sys.format_path('/usr/local/mysql/data/world/City.ibd') |
+-----------------------------------------------------+
| @@datadir/world/City.ibd                            |
+-----------------------------------------------------+
```

28.4.5.5 The format_statement() Function

Given a string (normally representing an SQL statement), reduces it to the length given by the `statement_truncate_len` configuration option, and returns the result. No truncation occurs if the string is shorter than `statement_truncate_len`. Otherwise, the middle part of the string is replaced by an ellipsis (`...`).

This function is useful for formatting possibly lengthy statements retrieved from Performance Schema tables to a known fixed maximum length.

Parameters

- `statement LONGTEXT`: The statement to format.

Configuration Options

`format_statement()` operation can be modified using the following configuration options or their corresponding user-defined variables (see Section 28.4.2.1, “The sys_config Table”):

- `statement_truncate_len`, `@sys.statement_truncate_len`

  The maximum length of statements returned by the `format_statement()` function. Longer statements are truncated to this length. The default is 64.

Return Value

A `LONGTEXT` value.

Example

By default, `format_statement()` truncates statements to be no more than 64 characters. Setting `@sys.statement_truncate_len` changes the truncation length for the current session:

```
mysql> SET @stmt = 'SELECT variable, value, set_time, set_by FROM sys_config';
mysql> SELECT sys.format_statement(@stmt);
+-----------------------------------------------------------+
| sys.format_statement(@stmt)                               |
+-----------------------------------------------------------+
| SELECT variable, value, set_time, set_by FROM sys_config |
+-----------------------------------------------------------+
mysql> SET @sys.statement_truncate_len = 32;
mysql> SELECT sys.format_statement(@stmt);
+----------------------------------+
| sys.format_statement(@stmt)      |
+----------------------------------+
| SELECT variabl ... ROM sys_config |
+----------------------------------+
```

28.4.5.6 The format_time() Function

Note

As of MySQL 8.0.16, `format_time()` is deprecated and subject to removal in a future MySQL version. Applications that use it should be migrated to use the built-in `FORMAT_PICO_TIME()` function instead. See Section 12.22, "Performance Schema Functions"

Given a Performance Schema latency or wait time in picoseconds, converts it to human-readable format and returns a string consisting of a value and a units indicator. Depending on the size of the value, the units part is `ps` (picoseconds), `ns` (nanoseconds), `us` (microseconds), `ms` (milliseconds), `s` (seconds), `m` (minutes), `h` (hours), `d` (days), or `w` (weeks).

Parameters

- `picoseconds TEXT`: The picoseconds value to format.

Return Value

A `TEXT` value.

Example

```
mysql> SELECT sys.format_time(3501), sys.format_time(188732396662000);
+----------------------+---------------------------------+
| sys.format_time(3501) | sys.format_time(188732396662000) |
+----------------------+---------------------------------+
| 3.50 ns               | 3.15 m                          |
+----------------------+---------------------------------+
```

28.4.5.7 The list_add() Function

Adds a value to a comma-separated list of values and returns the result.

This function and `list_drop()` can be useful for manipulating the value of system variables such as `sql_mode` and `optimizer_switch` that take a comma-separated list of values.

Parameters

- `in_list TEXT`: The list to be modified.
- `in_add_value TEXT`: The value to add to the list.

Return Value

A `TEXT` value.

Example

```
mysql> SELECT @@sql_mode;
+-------------------------------------+
| @@sql_mode                          |
+-------------------------------------+
| ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES |
+-------------------------------------+
mysql> SET @@sql_mode = sys.list_add(@@sql_mode, 'NO_ENGINE_SUBSTITUTION');
mysql> SELECT @@sql_mode;
+---------------------------------------------------------------+
| @@sql_mode                                                    |
+---------------------------------------------------------------+
| ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION |
+---------------------------------------------------------------+
mysql> SET @@sql_mode = sys.list_drop(@@sql_mode, 'ONLY_FULL_GROUP_BY');
mysql> SELECT @@sql_mode;
+-----------------------------------------+
| @@sql_mode                              |
+-----------------------------------------+
| STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION |
+-----------------------------------------+
```

28.4.5.8 The list_drop() Function

Removes a value from a comma-separated list of values and returns the result. For more information, see the description of `list_add()`

Parameters

- `in_list TEXT`: The list to be modified.
- `in_drop_value TEXT`: The value to drop from the list.

Return Value

A `TEXT` value.

28.4.5.9 The ps_is_account_enabled() Function

Returns `YES` or `NO` to indicate whether Performance Schema instrumentation for a given account is enabled.

Parameters

- `in_host VARCHAR(60)`: The host name of the account to check.
- `in_user VARCHAR(32)`: The user name of the account to check.

Return Value

An `ENUM('YES','NO')` value.

Example

```
mysql> SELECT sys.ps_is_account_enabled('localhost', 'root');
+-----------------------------------------------+
| sys.ps_is_account_enabled('localhost', 'root') |
+-----------------------------------------------+
| YES                                           |
+-----------------------------------------------+
```

28.4.5.10 The ps_is_consumer_enabled() Function

Returns `YES` or `NO` to indicate whether a given Performance Schema consumer is enabled, or `NULL` if the argument is `NULL`. If the argument is not a valid consumer name, an error occurs. (Prior to MySQL 8.0.18, the function returns `NULL` if the argument is not a valid consumer name.)

This function accounts for the consumer hierarchy, so a consumer is not considered enabled unless all consumers on which depends are also enabled. For information about the consumer hierarchy, see Section 27.4.7, “Pre-Filtering by Consumer”.

Parameters

- `in_consumer VARCHAR(64)`: The name of the consumer to check.

Return Value

An `ENUM('YES','NO')` value.

Example

```
mysql> SELECT sys.ps_is_consumer_enabled('thread_instrumentation');
+-----------------------------------------------------+
| sys.ps_is_consumer_enabled('thread_instrumentation') |
+-----------------------------------------------------+
| YES                                                 |
+-----------------------------------------------------+
```

28.4.5.11 The ps_is_instrument_default_enabled() Function

Returns `YES` or `NO` to indicate whether a given Performance Schema instrument is enabled by default.

Parameters

- `in_instrument VARCHAR(128)`: The name of the instrument to check.

Return Value

An `ENUM('YES','NO')` value.

Example

```
mysql> SELECT sys.ps_is_instrument_default_enabled('memory/innodb/row_log_buf');
+-----------------------------------------------------------------+
| sys.ps_is_instrument_default_enabled('memory/innodb/row_log_buf') |
+-----------------------------------------------------------------+
| NO                                                              |
+-----------------------------------------------------------------+
mysql> SELECT sys.ps_is_instrument_default_enabled('statement/sql/alter_user');
+-------------------------------------------------------------------+
| sys.ps_is_instrument_default_enabled('statement/sql/alter_user') |
+-------------------------------------------------------------------+
| YES                                                               |
+-------------------------------------------------------------------+
```

28.4.5.12 The ps_is_instrument_default_timed() Function

Returns `YES` or `NO` to indicate whether a given Performance Schema instrument is timed by default.

Parameters

- `in_instrument VARCHAR(128)`: The name of the instrument to check.

Return Value

An `ENUM('YES','NO')` value.

Example

```
mysql> SELECT sys.ps_is_instrument_default_timed('memory/innodb/row_log_buf');
+-------------------------------------------------------------+
| sys.ps_is_instrument_default_timed('memory/innodb/row_log_buf') |
+-------------------------------------------------------------+
| NO                                                          |
+-------------------------------------------------------------+
mysql> SELECT sys.ps_is_instrument_default_timed('statement/sql/alter_user');
+-------------------------------------------------------------+
| sys.ps_is_instrument_default_timed('statement/sql/alter_user') |
+-------------------------------------------------------------+
| YES                                                         |
+-------------------------------------------------------------+
```

28.4.5.13 The ps_is_thread_instrumented() Function

Returns `YES` or `NO` to indicate whether Performance Schema instrumentation for a given connection ID is enabled, `UNKNOWN` if the ID is unknown, or `NULL` if the ID is `NULL`.

Parameters

- `in_connection_id BIGINT UNSIGNED`: The connection ID. This is a value of the type given in the `PROCESSLIST_ID` column of the Performance Schema `threads` table or the `Id` column of `SHOW PROCESSLIST` output.

Return Value

An `ENUM('YES','NO','UNKNOWN')` value.

Example

```
mysql> SELECT sys.ps_is_thread_instrumented(43);
+-----------------------------------+
| sys.ps_is_thread_instrumented(43) |
+-----------------------------------+
| UNKNOWN                           |
+-----------------------------------+
mysql> SELECT sys.ps_is_thread_instrumented(CONNECTION_ID());
+-----------------------------------------------+
| sys.ps_is_thread_instrumented(CONNECTION_ID()) |
+-----------------------------------------------+
| YES                                           |
+-----------------------------------------------+
```

28.4.5.14 The ps_thread_account() Function

Given a Performance Schema thread ID, returns the *`user_name@host_name`* account associated with the thread.

Parameters

- `in_thread_id BIGINT UNSIGNED`: The thread ID for which to return the account. The value should match the `THREAD_ID` column from some Performance Schema `threads` table row.

Return Value

A `TEXT` value.

Example

```
mysql> SELECT sys.ps_thread_account(sys.ps_thread_id(CONNECTION_ID()));
+-----------------------------------------------------------+
| sys.ps_thread_account(sys.ps_thread_id(CONNECTION_ID())) |
+-----------------------------------------------------------+
| root@localhost                                            |
+-----------------------------------------------------------+
```

28.4.5.15 The ps_thread_id() Function

Note

As of MySQL 8.0.16, `ps_thread_id()` is deprecated and subject to removal in a future MySQL version. Applications that use it should be migrated to use the built-in `PS_THREAD_ID()` and `PS_CURRENT_THREAD_ID()` functions instead. See Section 12.22, “Performance Schema Functions”

Returns the Performance Schema thread ID assigned to a given connection ID, or the thread ID for the current connection if the connection ID is `NULL`.

Parameters

- `in_connection_id BIGINT UNSIGNED`: The ID of the connection for which to return the thread ID. This is a value of the type given in the `PROCESSLIST_ID` column of the Performance Schema `threads` table or the `Id` column of `SHOW PROCESSLIST` output.

Return Value

A `BIGINT UNSIGNED` value.

Example

```
mysql> SELECT sys.ps_thread_id(260);
+-----------------------+
| sys.ps_thread_id(260) |
+-----------------------+
|                   285 |
+-----------------------+
```

28.4.5.16 The ps_thread_stack() Function

Returns a JSON formatted stack of all statements, stages, and events within the Performance Schema for a given thread ID.

Parameters

- `in_thread_id BIGINT`: The ID of the thread to trace. The value should match the `THREAD_ID` column from some Performance Schema `threads` table row.
- `in_verbose BOOLEAN`: Whether to include `file:lineno` information in the events.

Return Value

A `LONGTEXT CHARACTER SET latin1` value.

Example

```
mysql> SELECT sys.ps_thread_stack(37, FALSE) AS thread_stack\G
```

```
*************************** 1. row ***************************
thread_stack: {"rankdir": "LR","nodesep": "0.10",
"stack_created": "2014-02-19 13:39:03", "mysql_version": "8.0.2-dmr-debug-log",
"mysql_user": "root@localhost","events": [{"nesting_event_id": "0",
"event_id": "10", "timer_wait": 256.35, "event_info": "sql/select",
"wait_info": "select @@version_comment limit 1\nerrors: 0\nwarnings: 0\nlock time:
...
```

28.4.5.17 The ps_thread_trx_info() Function

Returns a JSON object containing information about a given thread. The information includes the current transaction, and the statements it has already executed, derived from the Performance Schema `events_transactions_current` and `events_statements_history` tables. (The consumers for those tables must be enabled to obtain full data in the JSON object.)

If the output exceeds the truncation length (65535 by default), a JSON error object is returned, such as:

```
{ "error": "Trx info truncated: Row 6 was cut by GROUP_CONCAT()" }
```

Similar error objects are returned for other warnings and exceptions raised during function execution.

Parameters

- `in_thread_id BIGINT UNSIGNED`: The thread ID for which to return transaction information. The value should match the `THREAD_ID` column from some Performance Schema `threads` table row.

Configuration Options

`ps_thread_trx_info()` operation can be modified using the following configuration options or their corresponding user-defined variables (see Section 28.4.2.1, “The sys_config Table”):

- `ps_thread_trx_info.max_length`, `@sys.ps_thread_trx_info.max_length`

  The maximum length of the output. The default is 65535.

Return Value

A `LONGTEXT` value.

Example

```
mysql> SELECT sys.ps_thread_trx_info(48)\G
*************************** 1. row ***************************
sys.ps_thread_trx_info(48): [
  {
    "time": "790.70 us",
    "state": "COMMITTED",
    "mode": "READ WRITE",
    "autocommitted": "NO",
    "gtid": "AUTOMATIC",
    "isolation": "REPEATABLE READ",
    "statements_executed": [
      {
        "sql_text": "INSERT INTO info VALUES (1, \'foo\')",
        "time": "471.02 us",
        "schema": "trx",
        "rows_examined": 0,
        "rows_affected": 1,
        "rows_sent": 0,
        "tmp_tables": 0,
        "tmp_disk_tables": 0,
        "sort_rows": 0,
        "sort_merge_passes": 0
      },
      {
        "sql_text": "COMMIT",
        "time": "254.42 us",
```

```
        "schema": "trx",
        "rows_examined": 0,
        "rows_affected": 0,
        "rows_sent": 0,
        "tmp_tables": 0,
        "tmp_disk_tables": 0,
        "sort_rows": 0,
        "sort_merge_passes": 0
      }
    ]
  },
  {
    "time": "426.20 us",
    "state": "COMMITTED",
    "mode": "READ WRITE",
    "autocommitted": "NO",
    "gtid": "AUTOMATIC",
    "isolation": "REPEATABLE READ",
    "statements_executed": [
      {
        "sql_text": "INSERT INTO info VALUES (2, \'bar\')",
        "time": "107.33 us",
        "schema": "trx",
        "rows_examined": 0,
        "rows_affected": 1,
        "rows_sent": 0,
        "tmp_tables": 0,
        "tmp_disk_tables": 0,
        "sort_rows": 0,
        "sort_merge_passes": 0
      },
      {
        "sql_text": "COMMIT",
        "time": "213.23 us",
        "schema": "trx",
        "rows_examined": 0,
        "rows_affected": 0,
        "rows_sent": 0,
        "tmp_tables": 0,
        "tmp_disk_tables": 0,
        "sort_rows": 0,
        "sort_merge_passes": 0
      }
    ]
  }
]
```


28.4.5.18 The quote_identifier() Function

Given a string argument, this function produces a quoted identifier suitable for inclusion in SQL statements. This is useful when a value to be used as an identifier is a reserved word or contains backtick (`) characters.

Parameters

`in_identifier TEXT`: The identifier to quote.

Return Value

A `TEXT` value.

Example

```
mysql> SELECT sys.quote_identifier('plain');
+------------------------------+
| sys.quote_identifier('plain') |
+------------------------------+
| `plain`                      |
+------------------------------+
```

```
mysql> SELECT sys.quote_identifier('trick`ier');
+----------------------------------+
| sys.quote_identifier('trick`ier') |
+----------------------------------+
| `trick``ier`                      |
+----------------------------------+
mysql> SELECT sys.quote_identifier('integer');
+--------------------------------+
| sys.quote_identifier('integer') |
+--------------------------------+
| `integer`                      |
+--------------------------------+
```

28.4.5.19 The sys_get_config() Function

Given a configuration option name, returns the option value from the `sys_config` table, or the provided default value (which may be `NULL`) if the option does not exist in the table.

If `sys_get_config()` returns the default value and that value is `NULL`, it is expected that the caller is able to handle `NULL` for the given configuration option.

By convention, routines that call `sys_get_config()` first check whether the corresponding user-defined variable exists and is non-`NULL`. If so, the routine uses the variable value without reading the `sys_config` table. If the variable does not exist or is `NULL`, the routine reads the option value from the table and sets the user-defined variable to that value. For more information about the relationship between configuration options and their corresponding user-defined variables, see Section 28.4.2.1, “The sys_config Table”.

If you want to check whether the configuration option has already been set and, if not, use the return value of `sys_get_config()`, you can use `IFNULL(...)` (see example later). However, this should not be done inside a loop (for example, for each row in a result set) because for repeated calls where the assignment is needed only in the first iteration, using `IFNULL(...)` is expected to be significantly slower than using an `IF (...) THEN ... END IF;` block (see example later).

Parameters

- `in_variable_name VARCHAR(128)`: The name of the configuration option for which to return the value.
- `in_default_value VARCHAR(128)`: The default value to return if the configuration option is not found in the `sys_config` table.

Return Value

A `VARCHAR(128)` value.

Example

Get a configuration value from the `sys_config` table, falling back to 128 as the default if the option is not present in the table:

```
mysql> SELECT sys.sys_get_config('statement_truncate_len', 128) AS Value;
+-------+
| Value |
+-------+
| 64    |
+-------+
```

One-liner example: Check whether the option is already set; if not, assign the `IFNULL(...)` result (using the value from the `sys_config` table):

```
mysql> SET @sys.statement_truncate_len =
       IFNULL(@sys.statement_truncate_len,
              sys.sys_get_config('statement_truncate_len', 64));
```

`IF (...) THEN ... END IF;` block example: Check whether the option is already set; if not, assign the value from the `sys_config` table:

```
IF (@sys.statement_truncate_len IS NULL) THEN
  SET @sys.statement_truncate_len = sys.sys_get_config('statement_truncate_len', 64);
END IF;
```

28.4.5.20 The version_major() Function

This function returns the major version of the MySQL server.

Parameters

None.

Return Value

A `TINYINT UNSIGNED` value.

Example

```
mysql> SELECT VERSION(), sys.version_major();
+-------------+--------------------+
| VERSION()   | sys.version_major() |
+-------------+--------------------+
| 8.0.26-debug |                  8 |
+-------------+--------------------+
```

28.4.5.21 The version_minor() Function

This function returns the minor version of the MySQL server.

Parameters

None.

Return Value

A `TINYINT UNSIGNED` value.

Example

```
mysql> SELECT VERSION(), sys.version_minor();
+-------------+--------------------+
| VERSION()   | sys.version_minor() |
+-------------+--------------------+
| 8.0.26-debug |                  0 |
+-------------+--------------------+
```

28.4.5.22 The version_patch() Function

This function returns the patch release version of the MySQL server.

Parameters

None.

Return Value

A `TINYINT UNSIGNED` value.

Example

```
mysql> SELECT VERSION(), sys.version_patch();
```

```
+-------------+--------------------+
| VERSION()   | sys.version_patch() |
+-------------+--------------------+
| 8.0.26-debug |                 26 |
+-------------+--------------------+
```

Chapter 29 Connectors and APIs

Table of Contents

MySQL Connectors provide connectivity to the MySQL server for client programs. APIs provide low-level access to MySQL resources using either the classic MySQL protocol or X Protocol. Both Connectors and the APIs enable you to connect and execute MySQL statements from another language or environment, including ODBC, Java (JDBC), C++, Python, Node.js, PHP, Perl, Ruby, and C.

MySQL Connectors

Oracle develops a number of connectors:

- Connector/C++ enables C++ applications to connect to MySQL.
- Connector/J provides driver support for connecting to MySQL from Java applications using the standard Java Database Connectivity (JDBC) API.
- Connector/NET enables developers to create .NET applications that connect to MySQL. Connector/NET implements a fully functional ADO.NET interface and provides support for use with ADO.NET aware tools. Applications that use Connector/NET can be written in any supported .NET language.

  MySQL for Visual Studio works with Connector/NET and Microsoft Visual Studio 2012, 2013, 2015, and 2017. MySQL for Visual Studio provides access to MySQL objects and data from Visual Studio. As a Visual Studio package, it integrates directly into Server Explorer providing the ability to create new connections and work with MySQL database objects.
- Connector/ODBC provides driver support for connecting to MySQL using the Open Database Connectivity (ODBC) API. Support is available for ODBC connectivity from Windows, Unix, and macOS platforms.
- Connector/Python provides driver support for connecting to MySQL from Python applications using an API that is compliant with the Python DB API version 2.0. No additional Python modules or MySQL client libraries are required.
- `Connector/Node.js` provides an asynchronous API for connecting to MySQL from Node.js applications using X Protocol. Connector/Node.js supports managing database sessions and schemas, working with MySQL Document Store collections and using raw SQL statements.

The MySQL C API

For direct access to using MySQL natively within a C application, the C API provides low-level access to the MySQL client/server protocol through the `libmysqlclient` client library. This is the primary method used to connect to an instance of the MySQL server, and is used both by MySQL command-line clients and many of the MySQL Connectors and third-party APIs detailed here.

`libmysqlclient` is included in MySQL distributions distributions.

See also MySQL C API Implementations.

To access MySQL from a C application, or to build an interface to MySQL for a language not supported by the Connectors or APIs in this chapter, the C API is where to start. A number of programmer's utilities are available to help with the process; see Section 4.7, “Program Development Utilities”.

Third-Party MySQL APIs

The remaining APIs described in this chapter provide an interface to MySQL from specific application languages. These third-party solutions are not developed or supported by Oracle. Basic information on their usage and abilities is provided here for reference purposes only.

All the third-party language APIs are developed using one of two methods, using `libmysqlclient` or by implementing a *native driver*. The two solutions offer different benefits:

- Using *`libmysqlclient`* offers complete compatibility with MySQL because it uses the same libraries as the MySQL client applications. However, the feature set is limited to the implementation and interfaces exposed through `libmysqlclient` and the performance may be lower as data is copied between the native language, and the MySQL API components.
- *Native drivers* are an implementation of the MySQL network protocol entirely within the host language or environment. Native drivers are fast, as there is less copying of data between components, and they can offer advanced functionality not available through the standard MySQL API. Native drivers are also easier for end users to build and deploy because no copy of the MySQL client libraries is needed to build the native driver components.

Table 29.1, “MySQL APIs and Interfaces” lists many of the libraries and interfaces available for MySQL.

Table 29.1 MySQL APIs and Interfaces

Environment	API	Type	Notes
Ada	GNU Ada MySQL Bindings	`libmysqlclient`	See MySQL Bindings for GNU Ada
C	C API	`libmysqlclient`	See MySQL 8.0 C API Developer Guide.
C++	Connector/C++	`libmysqlclient`	See MySQL Connector/C++ 8.0 Developer Guide.
	MySQL++	`libmysqlclient`	See MySQL++ website.
	MySQL wrapped	`libmysqlclient`	See MySQL wrapped.
Cocoa	MySQL-Cocoa	`libmysqlclient`	Compatible with the Objective-C Cocoa environment. See http://mysql-cocoa.sourceforge.net/
D	MySQL for D	`libmysqlclient`	See MySQL for D.
Eiffel	Eiffel MySQL	`libmysqlclient`	See Section 29.13, “MySQL Eiffel Wrapper”.

Environment	API	Type	Notes
Erlang	`erlang-mysql-driver`	`libmysqlclient`	See `erlang-mysql-driver`.
Haskell	Haskell MySQL Bindings	Native Driver	See Brian O'Sullivan's pure Haskell MySQL bindings.
	`hsql-mysql`	`libmysqlclient`	See MySQL driver for Haskell.
Java/JDBC	Connector/J	Native Driver	See MySQL Connector/J 8.0 Developer Guide.
Kaya	MyDB	`libmysqlclient`	See MyDB.
Lua	LuaSQL	`libmysqlclient`	See LuaSQL.
.NET/Mono	Connector/NET	Native Driver	See MySQL Connector/NET Developer Guide.
Objective Caml	OBjective Caml MySQL Bindings	`libmysqlclient`	See MySQL Bindings for Objective Caml.
Octave	Database bindings for GNU Octave	`libmysqlclient`	See Database bindings for GNU Octave.
ODBC	Connector/ODBC	`libmysqlclient`	See MySQL Connector/ODBC Developer Guide.
Perl	`DBI`/`DBD::mysql`	`libmysqlclient`	See Section 29.9, “MySQL Perl API”.
	`Net::MySQL`	Native Driver	See `Net::MySQL` at CPAN
PHP	`mysql`, `ext/mysql` interface (deprecated)	`libmysqlclient`	See MySQL and PHP.
	`mysqli`, `ext/mysqli` interface	`libmysqlclient`	See MySQL and PHP.
	`PDO_MYSQL`	`libmysqlclient`	See MySQL and PHP.
	PDO mysqlnd	Native Driver	
Python	Connector/Python	Native Driver	See MySQL Connector/Python Developer Guide.
Python	Connector/Python C Extension	`libmysqlclient`	See MySQL Connector/Python Developer Guide.
	MySQLdb	`libmysqlclient`	See Section 29.10, “MySQL Python API”.
Ruby	mysql2	`libmysqlclient`	Uses `libmysqlclient`. See Section 29.11, “MySQL Ruby APIs”.
Scheme	`Myscsh`	`libmysqlclient`	See `Myscsh`.
SPL	`sql_mysql`	`libmysqlclient`	See `sql_mysql` for SPL.
Tcl	MySQLtcl	`libmysqlclient`	See Section 29.12, “MySQL Tcl API”.

29.1 MySQL Connector/C++

The MySQL Connector/C++ manual is published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/C++ 8.0 Developer Guide
- Release notes: MySQL Connector/C++ Release Notes

29.2 MySQL Connector/J

The MySQL Connector/J manual is published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/J Developer Guide
- Release notes: MySQL Connector/J Release Notes

29.3 MySQL Connector/NET

The MySQL Connector/NET manual is published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/NET Developer Guide
- Release notes: MySQL Connector/NET Release Notes

29.4 MySQL Connector/ODBC

The MySQL Connector/ODBC manual is published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/ODBC Developer Guide
- Release notes: MySQL Connector/ODBC Release Notes

29.5 MySQL Connector/Python

The MySQL Connector/Python manual is published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/Python Developer Guide
- Release notes: MySQL Connector/Python Release Notes

29.6 MySQL Connector/Node.js

The MySQL Connector/Node.js manual is published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Release notes: MySQL Connector/Node.js Release Notes

29.7 MySQL C API

The MySQL C API Developer Guide is published in standalone form, not as part of the MySQL Reference Manual. See MySQL 8.0 C API Developer Guide.

29.8 MySQL PHP API

The MySQL PHP API manual is now published in standalone form, not as part of the MySQL Reference Manual. See MySQL and PHP.

29.9 MySQL Perl API

The Perl `DBI` module provides a generic interface for database access. You can write a DBI script that works with many different database engines without change. To use DBI with MySQL, install the following:

1. The `DBI` module.
2. The `DBD::mysql` module. This is the DataBase Driver (DBD) module for Perl.
3. Optionally, the DBD module for any other type of database server you want to access.

Perl DBI is the recommended Perl interface. It replaces an older interface called `mysqlperl`, which should be considered obsolete.

These sections contain information about using Perl with MySQL and writing MySQL applications in Perl:

- For installation instructions for Perl DBI support, see Section 2.12, “Perl Installation Notes”.
- For an example of reading options from option files, see Section 5.8.4, “Using Client Programs in a Multiple-Server Environment”.
- For secure coding tips, see Section 6.1.1, “Security Guidelines”.
- For debugging tips, see Section 5.9.1.4, “Debugging mysqld under gdb”.
- For some Perl-specific environment variables, see Section 4.9, “Environment Variables”.
- For considerations for running on macOS, see Section 2.4, “Installing MySQL on macOS”.
- For ways to quote string literals, see Section 9.1.1, “String Literals”.

DBI information is available at the command line, online, or in printed form:

- Once you have the `DBI` and `DBD::mysql` modules installed, you can get information about them at the command line with the `perldoc` command:

  ```
  $> perldoc DBI
  $> perldoc DBI::FAQ
  $> perldoc DBD::mysql
  ```

  You can also use `pod2man`, `pod2html`, and so on to translate this information into other formats.

- For online information about Perl DBI, visit the DBI website, http://dbi.perl.org/. That site hosts a general DBI mailing list.
- For printed information, the official DBI book is *Programming the Perl DBI* (Alligator Descartes and Tim Bunce, O'Reilly & Associates, 2000). Information about the book is available at the DBI website, http://dbi.perl.org/.

29.10 MySQL Python API

`MySQLdb` is a third-party driver that provides MySQL support for Python, compliant with the Python DB API version 2.0. It can be found at http://sourceforge.net/projects/mysql-python/.

The new MySQL Connector/Python component provides an interface to the same Python API, and is built into the MySQL Server and supported by Oracle. See MySQL Connector/Python Developer Guide for details on the Connector, as well as coding guidelines for Python applications and sample Python code.

29.11 MySQL Ruby APIs

The `mysql2` Ruby gem provides an API for connecting to MySQL, performing queries, and iterating through results; it is intended to support MySQL 5.7 and MySQL 8.0. For more information, see the `mysql2` page at RubyGems.org or the project's GitHub page.

For background and syntax information about the Ruby language, see Ruby Programming Language.

29.11.1 The MySQL/Ruby API

The MySQL/Ruby module provides access to MySQL databases using Ruby through `libmysqlclient`.

For information on installing the module, and the functions exposed, see MySQL/Ruby.

29.11.2 The Ruby/MySQL API

The Ruby/MySQL module provides access to MySQL databases using Ruby through a native driver interface using the MySQL network protocol.

For information on installing the module, and the functions exposed, see Ruby/MySQL.

29.12 MySQL Tcl API

`MySQLtcl` is a simple API for accessing a MySQL database server from the Tcl programming language. It can be found at http://www.xdobry.de/mysqltcl/.

29.13 MySQL Eiffel Wrapper

Eiffel MySQL is an interface to the MySQL database server using the Eiffel programming language, written by Michael Ravits. It can be found at http://efsa.sourceforge.net/archive/ravits/mysql.htm.

Chapter 30 MySQL Enterprise Edition

Table of Contents

MySQL Enterprise Edition is a commercial product. Like MySQL Community Edition, MySQL Enterprise Edition includes MySQL Server, a fully integrated transaction-safe, ACID-compliant database with full commit, rollback, crash-recovery, and row-level locking capabilities. In addition, MySQL Enterprise Edition includes the following components designed to provide monitoring and online backup, as well as improved security and scalability:

The following sections briefly discuss each of these components and indicate where to find more detailed information. To learn more about commercial products, see https://www.mysql.com/products/.

- MySQL Enterprise Monitor
- MySQL Enterprise Backup
- MySQL Enterprise Security
- MySQL Enterprise Encryption
- MySQL Enterprise Audit
- MySQL Enterprise Firewall
- MySQL Enterprise Thread Pool
- MySQL Enterprise Data Masking and De-Identification

30.1 MySQL Enterprise Monitor Overview

MySQL Enterprise Monitor is an enterprise monitoring system for MySQL that keeps an eye on your MySQL servers, notifies you of potential issues and problems, and advises you how to fix the issues. MySQL Enterprise Monitor can monitor all kinds of configurations, from a single MySQL server that is important to your business, all the way up to a huge farm of MySQL servers powering a busy website.

The following discussion briefly summarizes the basic components that make up the MySQL Enterprise Monitor product. For more information, see the MySQL Enterprise Monitor manual, available at https://dev.mysql.com/doc/mysql-monitor/en/.

MySQL Enterprise Monitor components can be installed in various configurations depending on your database and network topology, to give you the best combination of reliable and responsive monitoring data, with minimal overhead on the database server machines. A typical MySQL Enterprise Monitor installation consists of:

- One or more MySQL servers to monitor. MySQL Enterprise Monitor can monitor both Community and Enterprise MySQL server releases.
- A MySQL Enterprise Monitor Agent for each monitored host.

- A single MySQL Enterprise Service Manager, which collates information from the agents and provides the user interface to the collected data.

MySQL Enterprise Monitor is designed to monitor one or more MySQL servers. The monitoring information is collected by using an agent, *MySQL Enterprise Monitor Agent*. The agent communicates with the hosts and MySQL servers that it monitors, collecting variables, status and health information, and sending this information to the MySQL Enterprise Service Manager.

The information collected by the agent about each MySQL server and host you are monitoring is sent to the *MySQL Enterprise Service Manager*. This server collates all of the information from the agents. As it collates the information sent by the agents, the MySQL Enterprise Service Manager continually tests the collected data, comparing the status of the server to reasonable values. When thresholds are reached, the server can trigger an event (including an alarm and notification) to highlight a potential issue, such as low memory, high CPU usage, or more complex conditions such insufficient buffer sizes and status information. We call each test, with its associated threshold value, a *rule*.

These rules, and the alarms and notifications, are each known as a *MySQL Enterprise Advisors*. Advisors form a critical part of the MySQL Enterprise Service Manager, as they provide warning information and troubleshooting advice about potential problems.

The MySQL Enterprise Service Manager includes a web server, and you interact with it through any web browser. This interface, the MySQL Enterprise Monitor User Interface, displays all of the information collected by the agents, and lets you view all of your servers and their current status as a group or individually. You control and configure all aspects of the service using the MySQL Enterprise Monitor User Interface.

The information supplied by the MySQL Enterprise Monitor Agent processes also includes statistical and query information, which you can view in the form of graphs. For example, you can view aspects such as server load, query numbers, or index usage information as a graph over time. The graph lets you pinpoint problems or potential issues on your server, and can help diagnose the impact from database or external problems (such as external system or network failure) by examining the data from a specific time interval.

The MySQL Enterprise Monitor Agent can also be configured to collect detailed information about the queries executed on your server, including the row counts and performance times for executing each query. You can correlate the detailed query data with the graphical information to identify which queries were executing when you experienced a particularly high load, index or other issue. The query data is supported by a system called Query Analyzer, and the data can be presented in different ways depending on your needs.

30.2 MySQL Enterprise Backup Overview

MySQL Enterprise Backup performs hot backup operations for MySQL databases. The product is architected for efficient and reliable backups of tables created by the InnoDB storage engine. For completeness, it can also back up tables from MyISAM and other storage engines.

The following discussion briefly summarizes MySQL Enterprise Backup. For more information, see the MySQL Enterprise Backup manual, available at https://dev.mysql.com/doc/mysql-enterprise-backup/en/.

Hot backups are performed while the database is running and applications are reading and writing to it. This type of backup does not block normal database operations, and it captures even changes that occur while the backup is happening. For these reasons, hot backups are desirable when your database “grows up” -- when the data is large enough that the backup takes significant time, and when your data is important enough to your business that you must capture every last change, without taking your application, website, or web service offline.

MySQL Enterprise Backup does a hot backup of all tables that use the InnoDB storage engine. For tables using MyISAM or other non-InnoDB storage engines, it does a “warm” backup, where the database continues to run, but those tables cannot be modified while being backed up. For efficient

backup operations, you can designate InnoDB as the default storage engine for new tables, or convert existing tables to use the InnoDB storage engine.

30.3 MySQL Enterprise Security Overview

MySQL Enterprise Edition provides plugins that implement security features using external services:

- MySQL Enterprise Edition includes an authentication plugin that enables MySQL Server to use LDAP (Lightweight Directory Access Protocol) to authenticate MySQL users. LDAP Authentications supports user name and password, SASL, and GSSAPI/Kerberos authentication methods to LDAP services. For more information, see Section 6.4.1.7, “LDAP Pluggable Authentication”.
- MySQL Enterprise Edition includes an authentication plugin that enables MySQL Server to use Native Kerberos to authenticate MySQL users using there Kerberos Principals. For more information, see Section 6.4.1.8, “Kerberos Pluggable Authentication”.
- MySQL Enterprise Edition includes an authentication plugin that enables MySQL Server to use PAM (Pluggable Authentication Modules) to authenticate MySQL users. PAM enables a system to use a standard interface to access various kinds of authentication methods, such as Unix passwords or an LDAP directory. For more information, see Section 6.4.1.5, “PAM Pluggable Authentication”.
- MySQL Enterprise Edition includes an authentication plugin that performs external authentication on Windows, enabling MySQL Server to use native Windows services to authenticate client connections. Users who have logged in to Windows can connect from MySQL client programs to the server based on the information in their environment without specifying an additional password. For more information, see Section 6.4.1.6, “Windows Pluggable Authentication”.
- MySQL Enterprise Edition includes a suite of masking and de-identification functions that perform subsetting, random generation, and dictionary replacement to de-identify strings, numerics, phone numbers, emails and more. These functions enable masking existing data using several methods such as obfuscation (removing identifying characteristics), generation of formatted random data, and data replacement or substitution. For more information, see Section 6.5.3, “Using MySQL Enterprise Data Masking and De-Identification”.
- MySQL Enterprise Edition includes a set of encryption functions based on the OpenSSL library that expose OpenSSL capabilities at the SQL level. For more information, see Section 30.4, “MySQL Enterprise Encryption Overview”.
- MySQL Enterprise Edition 5.7 and higher includes a keyring plugin that uses Oracle Key Vault as a backend for keyring storage. For more information, see Section 6.4.4, “The MySQL Keyring”.
- MySQL Transparent Data Encryption (TDE) provides at-rest encryption for MySQL Server for all files that might contain sensitive data. For more information, see Section 15.13, “InnoDB Data-at-Rest Encryption”, Section 17.3.2, “Encrypting Binary Log Files and Relay Log Files”, and Encrypting Audit Log Files.

For other related Enterprise security features, see Section 30.4, “MySQL Enterprise Encryption Overview”.

30.4 MySQL Enterprise Encryption Overview

MySQL Enterprise Edition includes a set of encryption functions based on the OpenSSL library that expose OpenSSL capabilities at the SQL level. These functions enable Enterprise applications to perform the following operations:

- Implement added data protection using public-key asymmetric cryptography
- Create public and private keys and digital signatures
- Perform asymmetric encryption and decryption

- Use cryptographic hashing for digital signing and data verification and validation

For more information, see Section 6.6, “MySQL Enterprise Encryption”.

For other related Enterprise security features, see Section 30.3, “MySQL Enterprise Security Overview”.

30.5 MySQL Enterprise Audit Overview

MySQL Enterprise Edition includes MySQL Enterprise Audit, implemented using a server plugin. MySQL Enterprise Audit uses the open MySQL Audit API to enable standard, policy-based monitoring and logging of connection and query activity executed on specific MySQL servers. Designed to meet the Oracle audit specification, MySQL Enterprise Audit provides an out of box, easy to use auditing and compliance solution for applications that are governed by both internal and external regulatory guidelines.

When installed, the audit plugin enables MySQL Server to produce a log file containing an audit record of server activity. The log contents include when clients connect and disconnect, and what actions they perform while connected, such as which databases and tables they access.

For more information, see Section 6.4.5, “MySQL Enterprise Audit”.

30.6 MySQL Enterprise Firewall Overview

MySQL Enterprise Edition includes MySQL Enterprise Firewall, an application-level firewall that enables database administrators to permit or deny SQL statement execution based on matching against allowlists of accepted statement patterns. This helps harden MySQL Server against attacks such as SQL injection or attempts to exploit applications by using them outside of their legitimate query workload characteristics.

Each MySQL account registered with the firewall has its own statement allowlist, enabling protection to be tailored per account. For a given account, the firewall can operate in recording or protecting mode, for training in the accepted statement patterns or protection against unacceptable statements.

For more information, see Section 6.4.7, “MySQL Enterprise Firewall”.

30.7 MySQL Enterprise Thread Pool Overview

MySQL Enterprise Edition includes MySQL Enterprise Thread Pool, implemented using a server plugin. The default thread-handling model in MySQL Server executes statements using one thread per client connection. As more clients connect to the server and execute statements, overall performance degrades. In MySQL Enterprise Edition, a thread pool plugin provides an alternative thread-handling model designed to reduce overhead and improve performance. The plugin implements a thread pool that increases server performance by efficiently managing statement execution threads for large numbers of client connections.

For more information, see Section 5.6.3, “MySQL Enterprise Thread Pool”.

30.8 MySQL Enterprise Data Masking and De-Identification Overview

MySQL Enterprise Edition 5.7 and higher includes MySQL Enterprise Data Masking and De-Identification, implemented as a plugin library containing a plugin and several loadable functions. Data masking hides sensitive information by replacing real values with substitutes. MySQL Enterprise Data Masking and De-Identification functions enable masking existing data using several methods such as obfuscation (removing identifying characteristics), generation of formatted random data, and data replacement or substitution.

For more information, see Section 6.5, “MySQL Enterprise Data Masking and De-Identification”.

Chapter 31 MySQL Workbench

MySQL Workbench provides a graphical tool for working with MySQL servers and databases. MySQL Workbench fully supports MySQL versions 5.5 and higher.

The following discussion briefly describes MySQL Workbench capabilities. For more information, see the MySQL Workbench manual, available at https://dev.mysql.com/doc/workbench/en/.

MySQL Workbench provides five main areas of functionality:

- **SQL Development**: Enables you to create and manage connections to database servers. As well as enabling you to configure connection parameters, MySQL Workbench provides the capability to execute SQL queries on the database connections using the built-in SQL Editor. This functionality replaces that previously provided by the Query Browser standalone application.
- **Data Modeling**: Enables you to create models of your database schema graphically, reverse and forward engineer between a schema and a live database, and edit all aspects of your database using the comprehensive Table Editor. The Table Editor provides easy-to-use facilities for editing Tables, Columns, Indexes, Triggers, Partitioning, Options, Inserts and Privileges, Routines and Views.
- **Server Administration**: Enables you to create and administer server instances.
- **Data Migration**: Allows you to migrate from Microsoft SQL Server, Sybase ASE, SQLite, SQL Anywhere, PostreSQL, and other RDBMS tables, objects and data to MySQL. Migration also supports migrating from earlier versions of MySQL to the latest releases.
- **MySQL Enterprise Support**: Support for Enterprise products such as MySQL Enterprise Backup and MySQL Audit.

MySQL Workbench is available in two editions, the Community Edition and the Commercial Edition. The Community Edition is available free of charge. The Commercial Edition provides additional Enterprise features, such as database documentation generation, at low cost.

Chapter 32 MySQL on the OCI Marketplace

Table of Contents

This chapter describes how to deploy MySQL Enterprise Edition as an Oracle Cloud Infrastructure (OCI) Marketplace Application. This is a BYOL product.

Note

For more information on OCI marketplace, see Overview of Marketplace.

The MySQL Enterprise Edition Marketplace Application is an OCI compute instance, running Oracle Linux 7.9, with MySQL EE 8.0. The MySQL EE installation on the deployed image is similar to the RPM installation, as described in Section 2.5.4, “Installing MySQL on Linux Using RPM Packages from Oracle”.

For more information on MySQL Enterprise Edition, see Chapter 30, *MySQL Enterprise Edition*.

For more information on MySQL advanced configuration, see Secure Deployment Guide.

For more information on Oracle Linux 7, see Oracle Linux Documentation

This product is user-managed, meaning you are responsible for upgrades and maintenance.

32.1 Prerequisites to Deploying MySQL EE on Oracle Cloud Infrastructure

The following assumptions are made:

- You are familiar with OCI terminology. If you are new to OCI, see Getting Started.
- You have access to a properly configured Virtual Cloud Network (VCN) and subnet. For more information, see Virtual Networking.
- You have the permissions required to deploy OCI Marketplace applications in a compartment of your tenancy. For more information, see How Policies Work.

32.2 Deploying MySQL EE on Oracle Cloud Infrastructure

To deploy MySQL EE on Oracle Cloud Infrastructure, do the following:

1. Open the OCI Marketplace and select **MySQL**.

   The **MySQL** listing is displayed.

2. Click **Launch Instance** to begin the application launch process.

   The **Create Compute Instance** dialog is displayed.

   See To create a Linux instance for information on how to complete the fields.

By default, the MySQL server listens on port 3306 and is configured with a single user, root.

Important

When the deployment is complete, and the MySQL server is started, you must connect to the compute instance and retrieve the default root password which was written to the MySQL log file.

See Connecting with SSH for more information.

The following MySQL software is installed:

- MySQL Server EE
- MySQL Enterprise Backup
- MySQL Shell
- MySQL Router

MySQL Configuration

For security, the following are enabled:

- SELinux: for more information, see Configuring and Using SELinux
- `firewalld`: for more information, see Controlling the firewalld Firewall Service

The following MySQL plugins are enabled:

- `thread_pool`
- `validate_password`

On startup, the following occurs:

- MySQL Server reads `/etc/my.cnf` and all files named `*.cnf` in `/etc/my.cnf.d/`.
- `/etc/my.cnf.d/perf-tuning.cnf` is created by `/usr/bin/mkcnf` based on the selected OCI shape.

Note

To disable this mechanism, remove `/etc/systemd/system/mysqld.service.d/perf-tuning.conf`.

Performance tuning is configured for the following shapes:

- VM.Standard2.1
- VM.Standard2.2
- VM.Standard2.4
- VM.Standard2.8
- VM.Standard2.16
- VM.Standard2.24
- VM.Standard.E2.1
- VM.Standard.E2.2
- VM.Standard.E2.4

- VM.Standard.E2.8
- VM.Standard.E3.Flex
- VM.Standard.E4.Flex
- BM.Standard2.52

For all other shapes, the tuning for VM.Standard2.1 is used.

32.3 Configuring Network Access

For information on OCI Security Rules, see Security Rules.

Important

You must enable ingress on the following ports:

- 22: SSH
- 3306: MySQL
- 33060: (optional) MySQL X Protocol. Used by MySQL Shell.

32.4 Connecting

This section describes the various connection methods for connecting to the deployed MySQL server on the OCI Compute Instance.

Connecting with SSH

This section gives some detail on connecting from a UNIX-like platform to the OCI Compute. For more information on connecting with SSH, see Accessing an Oracle Linux Instance Using SSH and Connecting to Your Instance.

To connect to the Oracle Linux running on the Compute Instance with SSH, run the following command:

```
ssh opc@computeIP
```

where `opc` is the compute user and *`computeIP`* is the IP address of your Compute Instance.

To find the temporary root password created for the root user, run the following command:

```
sudo grep 'temporary password' /var/log/mysqld.log
```

To change your default password, log in to the server using the generated, temporary password, using the following command: `mysql -uroot -p`. Then run the following:

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'MyNewPass4!';
```

Connecting with MySQL Client

Note

To connect from your local MySQL client, you must first create on the MySQL server a user which allows remote login.

To connect to the MySQL Server from your local MySQL client, run the following command from your shell session:

```
mysql -uroot -p -hcomputeIP
```

where *computeIP* is the IP address of your Compute Instance.

Connecting with MySQL Shell

To connect to the MySQL Server from your local MySQL Shell, run the following command to start your shell session:

```
mysqlsh \connect root@computeIP
```

where *computeIP* is the IP address of your Compute Instance.

For more information on MySQL Shell connections, see MySQL Shell Connections.

Connecting with Workbench

To connect to the MySQL Server from MySQL Workbench, see Connections in MySQL Workbench.

32.5 Maintenance

This product is user-managed, meaning you are responsible for upgrades and maintenance.

Upgrading MySQL

The existing installation is RPM-based, to upgrade the MySQL server, see Section 2.10.6, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”.

You can use `scp` to copy the required RPM to the OCI Compute Instance, or copy it from OCI Object Storage, if you have configured access to it. File Storage is also an option. For more information, see File Storage and NFS.

Backup and Restore

MySQL Enterprise Backup is the preferred backup and restore solution. For more information, see Backing Up to Cloud Storage.

For information on MySQL Enterprise Backup, see Getting Started with MySQL Enterprise Backup.

For information on the default MySQL backup and restore, see Chapter 7, *Backup and Recovery*.

Appendix A MySQL 8.0 Frequently Asked Questions

Table of Contents

A.1 MySQL 8.0 FAQ: General

A.1.1. Which version of MySQL is production-ready (GA)?

MySQL 8.0, 5.7, and MySQL 5.6 are supported for production use.

MySQL 8.0 achieved General Availability (GA) status with MySQL 8.0.11, which was released for production use on 19 April 2018.

MySQL 5.7 achieved General Availability (GA) status with MySQL 5.7.9, which was released for production use on 21 October 2015.

MySQL 5.6 achieved General Availability (GA) status with MySQL 5.6.10, which was released for production use on 5 February 2013.

MySQL 5.5 achieved General Availability (GA) status with MySQL 5.5.8, which was released for production use on 3 December 2010. Active development for MySQL 5.5 has ended.

MySQL 5.1 achieved General Availability (GA) status with MySQL 5.1.30, which was released for production use on 14 November 2008. Active development for MySQL 5.1 has ended.

MySQL 5.0 achieved General Availability (GA) status with MySQL 5.0.15, which was released for production use on 19 October 2005. Active development for MySQL 5.0 has ended.

A.1.2. Why did MySQL version numbering skip versions 6 and 7 and go straight to 8.0?

Due to the many new and important features we were introducing in this MySQL version, we decided to start a fresh new series. As the series numbers 6 and 7 had actually been used before by MySQL, we went to 8.0.

A.1.3. Can MySQL 8.0 do subqueries?

Yes. See Section 13.2.15, “Subqueries”.

A.1.4. Can MySQL 8.0 perform multiple-table inserts, updates, and deletes?

Yes. For the syntax required to perform multiple-table updates, see Section 13.2.17, “UPDATE Statement”; for that required to perform multiple-table deletes, see Section 13.2.2, “DELETE Statement”.

A multiple-table insert can be accomplished using a trigger whose `FOR EACH ROW` clause contains multiple `INSERT` statements within a `BEGIN ... END` block. See Section 25.3, “Using Triggers”.

A.1.5. Does MySQL 8.0 have Sequences?

No. However, MySQL has an `AUTO_INCREMENT` system, which in MySQL 8.0 can also handle inserts in a multi-source replication setup. With the `auto_increment_increment` and `auto_increment_offset` system variables, you can set each server to generate auto-increment values that don't conflict with other servers. The `auto_increment_increment` value should be greater than the number of servers, and each server should have a unique offset.

A.1.6. Does MySQL 8.0 have a `NOW()` function with fractions of seconds?

Yes, see Section 11.2.6, “Fractional Seconds in Time Values”.

A.1.7. Does MySQL 8.0 work with multi-core processors?

Yes. MySQL is fully multithreaded, and makes use of all CPUs made available to it. Not all CPUs may be available; modern operating systems should be able to utilize all underlying CPUs, but also make it possible to restrict a process to a specific CPU or sets of CPUs.

On Windows, there is currently a limit to the number of (logical) processors that `mysqld` can use: a single processor group, which is limited to a maximum of 64 logical processors.

Use of multiple cores may be seen in these ways:

- A single core is usually used to service the commands issued from one session.
- A few background threads make limited use of extra cores; for example, to keep background I/O tasks moving.
- If the database is I/O-bound (indicated by CPU consumption less than capacity), adding more CPUs is futile. If the database is partitioned into an I/O-bound part and a CPU-bond part, adding CPUs may still be useful.

A.1.8. Why do I see multiple processes for `mysqld`?

`mysqld` is a single-process program, not a multi-process program, and does not fork or launch other processes. However, `mysqld` is multithreaded and some process-reporting system utilities display separate entries for each thread of multithreaded processes, which may lead to the appearance of multiple `mysqld` processes when in fact there is only one.

A.1.9. Can MySQL 8.0 perform ACID transactions?

Yes. All current MySQL versions support transactions. The `InnoDB` storage engine offers full ACID transactions with row-level locking, multi-versioning, nonlocking repeatable reads, and all four SQL standard isolation levels.

The NDB storage engine supports the READ COMMITTED transaction isolation level only.

A.2 MySQL 8.0 FAQ: Storage Engines

A.2.1. Where can I obtain complete documentation for MySQL storage engines?

See Chapter 16, *Alternative Storage Engines*. That chapter contains information about all MySQL storage engines except for the InnoDB storage engine and the NDB storage engine (used for MySQL Cluster). InnoDB is covered in Chapter 15, *The InnoDB Storage Engine*. NDB is covered in Chapter 23, *MySQL NDB Cluster 8.0*.

A.2.2. Are there any new storage engines in MySQL 8.0?

No. InnoDB is the default storage engine for new tables. See Section 15.1, “Introduction to InnoDB” for details.

A.2.3. Have any storage engines been removed in MySQL 8.0?

The PARTITION storage engine plugin which provided partitioning support is replaced by a native partitioning handler. As part of this change, the server can no longer be built using -DWITH_PARTITION_STORAGE_ENGINE. partition is also no longer displayed in the output of SHOW PLUGINS, or shown in the INFORMATION_SCHEMA.PLUGINS table.

In order to support partitioning of a given table, the storage engine used for the table must now provide its own (“native”) partitioning handler. InnoDB is the only storage engine supported in MySQL 8.0 that includes a native partitioning handler. An attempt to create partitioned tables in MySQL 8.0 using any other storage engine fails. (The NDB storage engine used by MySQL Cluster also provides its own partitioning handler, but is currently not supported by MySQL 8.0.)

A.2.4. Can I prevent the use of a particular storage engine?

Yes. The disabled_storage_engines configuration option defines which storage engines cannot be used to create tables or tablespaces. By default, disabled_storage_engines is empty (no engines disabled), but it can be set to a comma-separated list of one or more engines.

A.2.5. Is there an advantage to using the InnoDB storage engine exclusively, as opposed to a combination of InnoDB and non-InnoDB storage engines?

Yes. Using InnoDB tables exclusively can simplify backup and recovery operations. MySQL Enterprise Backup does a hot backup of all tables that use the InnoDB storage engine. For tables using MyISAM or other non-InnoDB storage engines, it does a “warm” backup, where the database continues to run, but those tables cannot be modified while being backed up. See Section 30.2, “MySQL Enterprise Backup Overview”.

A.2.6. What are the unique benefits of the ARCHIVE storage engine?

The ARCHIVE storage engine stores large amounts of data without indexes; it has a small footprint, and performs selects using table scans. See Section 16.5, “The ARCHIVE Storage Engine”, for details.

A.3 MySQL 8.0 FAQ: Server SQL Mode

A.3.1. What are server SQL modes?

Server SQL modes define what SQL syntax MySQL should support and what kind of data validation checks it should perform. This makes it easier to use MySQL in different environments and to use MySQL together with other database servers. The MySQL Server apply these modes individually to different clients. For more information, see Section 5.1.11, “Server SQL Modes”.

A.3.2. How many server SQL modes are there?

Each mode can be independently switched on and off. See Section 5.1.11, “Server SQL Modes”, for a complete list of available modes.

A.3.3. How do you determine the server SQL mode?

You can set the default SQL mode (for `mysqld` startup) with the `--sql-mode` option. Using the statement `SET [GLOBAL|SESSION] sql_mode='modes'`, you can change the settings from within a connection, either locally to the connection, or to take effect globally. You can retrieve the current mode by issuing a `SELECT @@sql_mode` statement.

A.3.4. Is the mode dependent on the database or connection?

A mode is not linked to a particular database. Modes can be set locally to the session (connection), or globally for the server. you can change these settings using `SET [GLOBAL|SESSION] sql_mode='modes'`.

A.3.5. Can the rules for strict mode be extended?

When we refer to *strict mode*, we mean a mode where at least one of the modes `TRADITIONAL`, `STRICT_TRANS_TABLES`, or `STRICT_ALL_TABLES` is enabled. Options can be combined, so you can add restrictions to a mode. See Section 5.1.11, “Server SQL Modes”, for more information.

A.3.6. Does strict mode impact performance?

The intensive validation of input data that some settings requires more time than if the validation is not done. While the performance impact is not that great, if you do not require such validation (perhaps your application already handles all of this), then MySQL gives you the option of leaving strict mode disabled. However, if you do require it, strict mode can provide such validation.

A.3.7. What is the default server SQL mode when MySQL 8.0 is installed?

The default SQL mode in MySQL 8.0 includes these modes: `ONLY_FULL_GROUP_BY`, `STRICT_TRANS_TABLES`, `NO_ZERO_IN_DATE`, `NO_ZERO_DATE`, `ERROR_FOR_DIVISION_BY_ZERO`, and `NO_ENGINE_SUBSTITUTION`.

For information about all available modes and default MySQL behavior, see Section 5.1.11, “Server SQL Modes”.

A.4 MySQL 8.0 FAQ: Stored Procedures and Functions

A.4.1. Does MySQL 8.0 support stored procedures and functions?

Yes. MySQL 8.0 supports two types of stored routines, stored procedures and stored functions.

A.4.2. Where can I find documentation for MySQL stored procedures and stored functions?

See Section 25.2, “Using Stored Routines”.

A.4.3. Is there a discussion forum for MySQL stored procedures?

Yes. See https://forums.mysql.com/list.php?98.

A.4.4. Where can I find the ANSI SQL 2003 specification for stored procedures?

Unfortunately, the official specifications are not freely available (ANSI makes them available for purchase). However, there are books, such as *SQL-99 Complete, Really* by Peter Gulutzan and Trudy Pelzer, that provide a comprehensive overview of the standard, including coverage of stored procedures.

A.4.5. How do you manage stored routines?

It is always good practice to use a clear naming scheme for your stored routines. You can manage stored procedures with `CREATE [FUNCTION|PROCEDURE]`, `ALTER [FUNCTION|PROCEDURE]`, `DROP [FUNCTION|PROCEDURE]`, and `SHOW CREATE [FUNCTION|PROCEDURE]`. You can obtain information about existing stored procedures using the `ROUTINES` table in the `INFORMATION_SCHEMA` database (see Section 26.3.30, “The INFORMATION_SCHEMA ROUTINES Table”).

A.4.6. Is there a way to view all stored procedures and stored functions in a given database?

Yes. For a database named *`dbname`*, use this query on the `INFORMATION_SCHEMA.ROUTINES` table:

```
SELECT ROUTINE_TYPE, ROUTINE_NAME
```

```
    FROM INFORMATION_SCHEMA.ROUTINES
    WHERE ROUTINE_SCHEMA='dbname';
```

For more information, see Section 26.3.30, “The INFORMATION_SCHEMA ROUTINES Table”.

The body of a stored routine can be viewed using `SHOW CREATE FUNCTION` (for a stored function) or `SHOW CREATE PROCEDURE` (for a stored procedure). See Section 13.7.7.9, “SHOW CREATE PROCEDURE Statement”, for more information.

A.4.7. Where are stored procedures stored?

Stored procedures are stored in the `mysql.routines` and `mysql.parameters` tables, which are part of the data dictionary. You cannot access these tables directly. Instead, query the `INFORMATION_SCHEMA ROUTINES` and `PARAMETERS` tables. See Section 26.3.30, “The INFORMATION_SCHEMA ROUTINES Table”, and Section 26.3.20, “The INFORMATION_SCHEMA PARAMETERS Table”.

You can also use `SHOW CREATE FUNCTION` to obtain information about stored functions, and `SHOW CREATE PROCEDURE` to obtain information about stored procedures. See Section 13.7.7.9, “SHOW CREATE PROCEDURE Statement”.

A.4.8. Is it possible to group stored procedures or stored functions into packages?

No. This is not supported in MySQL 8.0.

A.4.9. Can a stored procedure call another stored procedure?

Yes.

A.4.10. Can a stored procedure call a trigger?

A stored procedure can execute an SQL statement, such as an `UPDATE`, that causes a trigger to activate.

A.4.11. Can a stored procedure access tables?

Yes. A stored procedure can access one or more tables as required.

A.4.12. Do stored procedures have a statement for raising application errors?

Yes. MySQL 8.0 implements the SQL standard `SIGNAL` and `RESIGNAL` statements. See Section 13.6.7, “Condition Handling”.

A.4.13. Do stored procedures provide exception handling?

MySQL implements `HANDLER` definitions according to the SQL standard. See Section 13.6.7.2, “DECLARE ... HANDLER Statement”, for details.

A.4.14. Can MySQL 8.0 stored routines return result sets?

Stored procedures can, but stored functions cannot. If you perform an ordinary `SELECT` inside a stored procedure, the result set is returned directly to the client. You need to use the MySQL 4.1 (or higher) client/server protocol for this to work. This means that, for example, in PHP, you need to use the `mysqli` extension rather than the old `mysql` extension.

A.4.15. Is `WITH RECOMPILE` supported for stored procedures?

Not in MySQL 8.0.

A.4.16. Is there a MySQL equivalent to using `mod_plsql` as a gateway on Apache to talk directly to a stored procedure in the database?

There is no equivalent in MySQL 8.0.

A.4.17. Can I pass an array as input to a stored procedure?

Not in MySQL 8.0.

A.4.18 Can I pass a cursor as an `IN` parameter to a stored procedure?

In MySQL 8.0, cursors are available inside stored procedures only.

A.4.19 Can I return a cursor as an `OUT` parameter from a stored procedure?

In MySQL 8.0, cursors are available inside stored procedures only. However, if you do not open a cursor on a `SELECT`, the result is sent directly to the client. You can also `SELECT INTO` variables. See Section 13.2.13, “SELECT Statement”.

A.4.20 Can I print out a variable's value within a stored routine for debugging purposes?

Yes, you can do this in a *stored procedure*, but not in a stored function. If you perform an ordinary `SELECT` inside a stored procedure, the result set is returned directly to the client. You must use the MySQL 4.1 (or above) client/server protocol for this to work. This means that, for example, in PHP, you need to use the `mysqli` extension rather than the old `mysql` extension.

A.4.21 Can I commit or roll back transactions inside a stored procedure?

Yes. However, you cannot perform transactional operations within a stored function.

A.4.22 Do MySQL 8.0 stored procedures and functions work with replication?

Yes, standard actions carried out in stored procedures and functions are replicated from a replication source server to a replica. There are a few limitations that are described in detail in Section 25.7, “Stored Program Binary Logging”.

A.4.23 Are stored procedures and functions created on a replication source server replicated to a replica?

Yes, creation of stored procedures and functions carried out through normal DDL statements on a replication source server are replicated to a replica, so that the objects exist on both servers. `ALTER` and `DROP` statements for stored procedures and functions are also replicated.

A.4.24 How are actions that take place inside stored procedures and functions replicated?

MySQL records each DML event that occurs in a stored procedure and replicates those individual actions to a replica. The actual calls made to execute stored procedures are not replicated.

Stored functions that change data are logged as function invocations, not as the DML events that occur inside each function.

A.4.25 Are there special security requirements for using stored procedures and functions together with replication?

Yes. Because a replica has authority to execute any statement read from a source's binary log, special security constraints exist for using stored functions with replication. If replication or binary logging in general (for the purpose of point-in-time recovery) is active, then MySQL DBAs have two security options open to them:

1. Any user wishing to create stored functions must be granted the `SUPER` privilege.
2. Alternatively, a DBA can set the `log_bin_trust_function_creators` system variable to 1, which enables anyone with the standard `CREATE ROUTINE` privilege to create stored functions.

A.4.26 What limitations exist for replicating stored procedure and function actions?

Nondeterministic (random) or time-based actions embedded in stored procedures may not replicate properly. By their very nature, randomly produced results are not predictable and cannot be exactly reproduced; therefore, random actions replicated to a replica do not mirror

those performed on a source. Declaring stored functions to be `DETERMINISTIC` or setting the `log_bin_trust_function_creators` system variable to 0 keeps random operations producing random values from being invoked.

In addition, time-based actions cannot be reproduced on a replica because the timing of such actions in a stored procedure is not reproducible through the binary log used for replication. It records only DML events and does not factor in timing constraints.

Finally, nontransactional tables for which errors occur during large DML actions (such as bulk inserts) may experience replication issues in that a source may be partially updated from DML activity, but no updates are done to the replica because of the errors that occurred. A workaround is for a function's DML actions to be carried out with the `IGNORE` keyword so that updates on the source that cause errors are ignored and updates that do not cause errors are replicated to the replica.

A.4.27 Do the preceding limitations affect the ability of MySQL to do point-in-time recovery?

The same limitations that affect replication do affect point-in-time recovery.

A.4.28 What is being done to correct the aforementioned limitations?

You can choose either statement-based replication or row-based replication. The original replication implementation is based on statement-based binary logging. Row-based binary logging resolves the limitations mentioned earlier.

Mixed replication is also available (by starting the server with `--binlog-format=mixed`). This hybrid form of replication “knows” whether statement-level replication can safely be used, or row-level replication is required.

For additional information, see Section 17.2.1, “Replication Formats”.

A.5 MySQL 8.0 FAQ: Triggers

A.5.1. Where can I find the documentation for MySQL 8.0 triggers?

See Section 25.3, “Using Triggers”.

A.5.2. Is there a discussion forum for MySQL Triggers?

Yes. It is available at https://forums.mysql.com/list.php?99.

A.5.3. Does MySQL 8.0 have statement-level or row-level triggers?

In MySQL 8.0, all triggers are `FOR EACH ROW`; that is, the trigger is activated for each row that is inserted, updated, or deleted. MySQL 8.0 does not support triggers using `FOR EACH STATEMENT`.

A.5.4. Are there any default triggers?

Not explicitly. MySQL does have specific special behavior for some `TIMESTAMP` columns, as well as for columns which are defined using `AUTO_INCREMENT`.

A.5.5. How are triggers managed in MySQL?

In MySQL 8.0, triggers can be created using the `CREATE TRIGGER` statement, and dropped using `DROP TRIGGER`. See Section 13.1.22, “CREATE TRIGGER Statement”, and Section 13.1.34, “DROP TRIGGER Statement”, for more about these statements.

Information about triggers can be obtained by querying the `INFORMATION_SCHEMA.TRIGGERS` table. See Section 26.3.45, “The INFORMATION_SCHEMA TRIGGERS Table”.

A.5.6. Is there a way to view all triggers in a given database?

Yes. You can obtain a listing of all triggers defined on database `dbname` using a query on the `INFORMATION_SCHEMA.TRIGGERS` table such as the one shown here:

```
SELECT TRIGGER_NAME, EVENT_MANIPULATION, EVENT_OBJECT_TABLE, ACTION_STATEMENT
    FROM INFORMATION_SCHEMA.TRIGGERS
    WHERE TRIGGER_SCHEMA='dbname';
```

For more information about this table, see Section 26.3.45, “The INFORMATION_SCHEMA TRIGGERS Table”.

You can also use the `SHOW TRIGGERS` statement, which is specific to MySQL. See Section 13.7.7.40, “SHOW TRIGGERS Statement”.

A.5.7. Where are triggers stored?

Triggers are stored in the `mysql.triggers` system table, which is part of the data dictionary.

A.5.8. Can a trigger call a stored procedure?

Yes.

A.5.9. Can triggers access tables?

A trigger can access both old and new data in its own table. A trigger can also affect other tables, but it is not permitted to modify a table that is already being used (for reading or writing) by the statement that invoked the function or trigger.

A.5.10. Can a table have multiple triggers with the same trigger event and action time?

In MySQL 8.0, it is possible to define multiple triggers for a given table that have the same trigger event and action time. For example, you can have two `BEFORE UPDATE` triggers for a table. By default, triggers that have the same trigger event and action time activate in the order they were created. To affect trigger order, specify a clause after `FOR EACH ROW` that indicates `FOLLOWS` or `PRECEDES` and the name of an existing trigger that also has the same trigger event and action time. With `FOLLOWS`, the new trigger activates after the existing trigger. With `PRECEDES`, the new trigger activates before the existing trigger.

A.5.11. Is it possible for a trigger to update tables on a remote server?

Yes. A table on a remote server could be updated using the `FEDERATED` storage engine. (See Section 16.8, “The FEDERATED Storage Engine”).

A.5.12. Do triggers work with replication?

Yes. However, the way in which they work depends whether you are using MySQL's “classic” statement-based or row-based replication format.

When using statement-based replication, triggers on the replica are executed by statements that are executed on the source (and replicated to the replica).

When using row-based replication, triggers are not executed on the replica due to statements that were run on the source and then replicated to the replica. Instead, when using row-based replication, the changes caused by executing the trigger on the source are applied on the replica.

For more information, see Section 17.5.1.36, “Replication and Triggers”.

A.5.13. How are actions carried out through triggers on a source replicated to a replica?

Again, this depends on whether you are using statement-based or row-based replication.

Statement-based replication. First, the triggers that exist on a source must be re-created on the replica server. Once this is done, the replication flow works as any other standard DML statement that participates in replication. For example, consider a table `EMP` that has an `AFTER` insert trigger, which exists on a replication source server. The same `EMP` table and `AFTER` insert trigger exist on the replica server as well. The replication flow would be:

1. An `INSERT` statement is made to `EMP`.
2. The `AFTER` trigger on `EMP` activates.
3. The `INSERT` statement is written to the binary log.
4. The replica picks up the `INSERT` statement to `EMP` and executes it.
5. The `AFTER` trigger on `EMP` that exists on the replica activates.

Row-based replication. When you use row-based replication, the changes caused by executing the trigger on the source are applied on the replica. However, the triggers themselves are not actually executed on the replica under row-based replication. This is because, if both the source and the replica applied the changes from the source and, in addition, the trigger causing these changes were applied on the replica, the changes would in effect be applied twice on the replica, leading to different data on the source and the replica.

In most cases, the outcome is the same for both row-based and statement-based replication. However, if you use different triggers on the source and replica, you cannot use row-based replication. (This is because the row-based format replicates the changes made by triggers executing on the source to the replicas, rather than the statements that caused the triggers to execute, and the corresponding triggers on the replica are not executed.) Instead, any statements causing such triggers to be executed must be replicated using statement-based replication.

For more information, see Section 17.5.1.36, “Replication and Triggers”.

A.6 MySQL 8.0 FAQ: Views

A.6.1. Where can I find documentation covering MySQL Views?

See Section 25.5, “Using Views”.

You may also find the MySQL User Forums to be helpful.

A.6.2. Is there a discussion forum for MySQL Views?

See the MySQL User Forums.

A.6.3. What happens to a view if an underlying table is dropped or renamed?

After a view has been created, it is possible to drop or alter a table or view to which the definition refers. To check a view definition for problems of this kind, use the `CHECK TABLE` statement. (See Section 13.7.3.2, “CHECK TABLE Statement”.)

A.6.4. Does MySQL 8.0 have table snapshots?

No.

A.6.5. Does MySQL 8.0 have materialized views?

No.

A.6.6. Can you insert into views that are based on joins?

It is possible, provided that your `INSERT` statement has a column list that makes it clear there is only one table involved.

You *cannot* insert into multiple tables with a single insert on a view.

A.7 MySQL 8.0 FAQ: INFORMATION_SCHEMA

A.7.1. Where can I find documentation for the MySQL `INFORMATION_SCHEMA` database?

See Chapter 26, *INFORMATION_SCHEMA Tables*.

You may also find the MySQL User Forums to be helpful.

A.7.2. Is there a discussion forum for `INFORMATION_SCHEMA`?

See the MySQL User Forums.

A.7.3. Where can I find the ANSI SQL 2003 specification for `INFORMATION_SCHEMA`?

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

A.7.4. What is the difference between the Oracle Data Dictionary and MySQL `INFORMATION_SCHEMA`?

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

A.7.5. Can I add to or otherwise modify the tables found in the `INFORMATION_SCHEMA` database?

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

A.8 MySQL 8.0 FAQ: Migration

A.8.1. Where can I find information on how to migrate from MySQL 5.7 to MySQL 8.0?

For detailed upgrade information, see Section 2.10, “Upgrading MySQL”. Do not skip a major version when upgrading, but rather complete the process in steps, upgrading from one major version to the next in each step. This may seem more complicated, but ultimately saves time and trouble. If you encounter problems during the upgrade, their origin is easier to identify, either by you or, if you have a MySQL Enterprise subscription, by MySQL support.

A.8.2. How has storage engine (table type) support changed in MySQL 8.0 from previous versions?

Storage engine support has changed as follows:

- Support for `ISAM` tables was removed in MySQL 5.0 and you should now use the `MyISAM` storage engine in place of `ISAM`. To convert a table *`tblname`* from `ISAM` to `MyISAM`, simply issue a statement such as this one:

  ```
  ALTER TABLE tblname ENGINE=MYISAM;
  ```

- Internal `RAID` for `MyISAM` tables was also removed in MySQL 5.0. This was formerly used to allow large tables in file systems that did not support file sizes greater than 2GB. All modern file systems allow for larger tables; in addition, there are now other solutions such as `MERGE` tables and views.

- The `VARCHAR` column type now retains trailing spaces in all storage engines.

- `MEMORY` tables (formerly known as `HEAP` tables) can also contain `VARCHAR` columns.

A.9 MySQL 8.0 FAQ: Security

A.9.1. Where can I find documentation that addresses security issues for MySQL?

The best place to start is Chapter 6, *Security*.

Other portions of the MySQL Documentation which you may find useful with regard to specific security concerns include the following:

- Section 6.1.1, “Security Guidelines”.
- Section 6.1.3, “Making MySQL Secure Against Attackers”.
- Section B.3.3.2, “How to Reset the Root Password”.
- Section 6.1.5, “How to Run MySQL as a Normal User”.
- Section 6.1.4, “Security-Related mysqld Options and Variables”.

- Section 6.1.6, “Security Considerations for LOAD DATA LOCAL”.
- Section 2.9, “Postinstallation Setup and Testing”.
- Section 6.3, “Using Encrypted Connections”.
- Loadable Function Security Precautions.

There is also the Secure Deployment Guide, which provides procedures for deploying a generic binary distribution of MySQL Enterprise Edition Server with features for managing the security of your MySQL installation.

A.9.2. What is the default authentication plugin in MySQL 8.0?

The default authentication plugin in MySQL 8.0 is `caching_sha2_password`. For information about this plugin, see Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

The `caching_sha2_password` plugin provides more secure password encryption than the `mysql_native_password` plugin (the default plugin in previous MySQL series). For information about the implications of this change of default plugin for server operation and compatibility of the server with clients and connectors, see caching_sha2_password as the Preferred Authentication Plugin.

For general information about pluggable authentication and other available authentication plugins, see Section 6.2.17, “Pluggable Authentication”, and Section 6.4.1, “Authentication Plugins”.

A.9.3. Does MySQL 8.0 have native support for SSL?

Most 8.0 binaries have support for SSL connections between the client and server. See Section 6.3, “Using Encrypted Connections”.

You can also tunnel a connection using SSH, if (for example) the client application does not support SSL connections. For an example, see Section 6.3.4, “Connecting to MySQL Remotely from Windows with SSH”.

A.9.4. Is SSL support built into MySQL binaries, or must I recompile the binary myself to enable it?

Most 8.0 binaries have SSL enabled for client/server connections that are secured, authenticated, or both. See Section 6.3, “Using Encrypted Connections”.

A.9.5. Does MySQL 8.0 have built-in authentication against LDAP directories?

The Enterprise edition includes a PAM Authentication Plugin that supports authentication against an LDAP directory.

A.9.6. Does MySQL 8.0 include support for Roles Based Access Control (RBAC)?

Not at this time.

A.9.7. Does MySQL 8.0 support TLS 1.0 and 1.1?

Support for the TLSv1 and TLSv1.1 connection protocols is removed as of MySQL 8.0.28. The protocols were deprecated from MySQL 8.0.26. For the consequences of that removal, see Removal of Support for the TLSv1 and TLSv1.1 Protocols.

Support for TLS versions 1.0 and 1.1 is removed because those protocol versions are old, released in 1996 and 2006, respectively. The algorithms used are weak and outdated.

Unless you are using very old versions of MySQL Server or connectors, you are unlikely to have connections using TLS 1.0 or 1.1. MySQL connectors and clients select the highest TLS version available by default.

When was support for TLS 1.2 added to MySQL Server? MySQL Community Server added TLS 1.2 support when the community server switched to OpenSSL for MySQL 5.6, 5.7, and 8.0 in 2019. For MySQL Enterprise Edition, OpenSSL added TLS 1.2 support in 2015, in MySQL Server 5.7.10.

How can one view which TLS versions are in active use? For MySQL 5.7 or 8.0, review whether TLS 1.0 or 1.1 is in use by running this query:

```
SELECT
  `session_ssl_status`.`thread_id`, `session_ssl_status`.`ssl_version`,
  `session_ssl_status`.`ssl_cipher`, `session_ssl_status`.`ssl_sessions_reused`
FROM `sys`.`session_ssl_status`
WHERE ssl_version NOT IN ('TLSv1.3','TLSv1.2');
```

If a thread using TLSv1.0 or TLSv1.1 is listed, you can determine where this connection is coming from by running this query:

```
SELECT thd_id,conn_id, user, db, current_statement, program_name
FROM sys.processlist
WHERE thd_id IN (
                  SELECT `session_ssl_status`.`thread_id`
                  FROM `sys`.`session_ssl_status`
                  WHERE ssl_version NOT IN ('TLSv1.3','TLSv1.2')
                );
```

Alternatively, you can run this query:

```
SELECT *
FROM sys.session
WHERE thd_id IN (
                  SELECT `session_ssl_status`.`thread_id`
                  FROM `sys`.`session_ssl_status`
                  WHERE ssl_version NOT IN ('TLSv1.3','TLSv1.2')
                );
```

These queries provide details needed to determine which application is not supporting TLS 1.2 or 1.3, and target upgrades for those.

Are there other options for testing for TLS 1.0 or 1.1? Yes, you can disable those versions prior to upgrading your server to a newer version. Explicitly specify which version to use, either in `mysql.cnf` (or `mysql.ini`) or by using `SET PERSIST`, for example: `--tls-version=TLSv12`.

Do all MySQL Connectors (5.7 and 8.0) support TLS 1.2 and higher? What about C and C++ applications using `libmysql`? For C and C++ applications using the community `libmysqlclient` library, use an OpenSSL-based library (that is, do *not* use YaSSL). Usage of OpenSSL was unified in 2018 (in MySQL 8.0.4 and 5.7.28, respectively). The same applies for Connector/ODBC and Connector/C++. To determine what library dependencies are used, run the following commands to see if OpenSSL is listed. On Linux, use this command:

```
$> sudo ldd usr/local/mysql/lib/libmysqlclient.a | grep -i openssl
```

On MacOS, use this command:

```
$> sudo otool -l /usr/local/mysql/lib/libmysqlclient.a | grep -i openssl
```

What about Connector/J? Java 8 moved to TLS 1.2 as the default in January 2014; TLS 1.2 was supported prior to that, so unless you are running a very old version of Connector/J, you have TLS 1.2 support.

What about Connector/NET? For .NET applications, Microsoft stopped support of TLS 1.0 and 1.1 at the end of 2020. Support for TLS 1.2 was added in 2012. You would need to have a very old version of Connector/NET not to have support for TLS 1.2.

What about Connector/Python? It depends on what version of Python you are running. The SSL module in Python 2.6 supports TLS up to version 1.0 only. In that case, you will need to upgrade to Python 2.7.9 or higher, or Python 3.x, both of which support newer versions of TLS. For details, see Connector/Python Versions and https://www.calazan.com/how-to-check-if-your-python-app-supports-tls-12/.

What about Connector/Node.js or Node MySQL2? TLS comes with `nodejs`, and all supported versions of Node.js use OpenSSL v1.1.1 (as of April 2020), which again supports TLS 1.2 and higher.

What about PHP? These versions of PHP support TLS 1.2 and higher.

A.10 MySQL 8.0 FAQ: NDB Cluster

In the following section, we answer questions that are frequently asked about MySQL NDB Cluster and the `NDB` storage engine.

A.10.1 Which versions of the MySQL software support NDB Cluster? Do I have to compile from source?

NDB Cluster is not supported in standard MySQL Server 8.0 releases. Instead, MySQL NDB Cluster is provided as a separate product. Available NDB Cluster release series include the following:

- **NDB Cluster 7.2.** This series is no longer supported for new deployments or maintained. Users of NDB Cluster 7.2 should upgrade to a newer release series as soon as possible. We recommend that new deployments use the latest NDB Cluster 8.0 release.
- **NDB Cluster 7.3.** This series is a previous General Availability (GA) version of NDB Cluster, still available for production use, although we recommend that new deployments use the latest NDB Cluster 8.0 release. The most recent NDB Cluster 7.3 release can be obtained from https://dev.mysql.com/downloads/cluster/.
- **NDB Cluster 7.4.** This series is a previous General Availability (GA) version of NDB Cluster, still available for production use, although we recommend that new deployments use the latest NDB Cluster 8.0 release. The most recent NDB Cluster 7.4 release can be obtained from https://dev.mysql.com/downloads/cluster/.
- **NDB Cluster 7.5.** This series is a previous General Availability (GA) version of NDB Cluster, still available for production use, although we recommend that new deployments use the latest NDB Cluster 7.6 release. The latest NDB Cluster 7.5 releases can be obtained from https://dev.mysql.com/downloads/cluster/.
- **NDB Cluster 7.6.** This series is a previous General Availability (GA) version of NDB Cluster, still available for production use, although we recommend that new deployments use the latest NDB Cluster 8.0 release. The latest NDB Cluster 7.6 releases can be obtained from https://dev.mysql.com/downloads/cluster/.
- **NDB Cluster 8.0.** This series is the most recent General Availability (GA) version of NDB Cluster, based on version 8.0 of the `NDB` storage engine and MySQL Server 8.0. NDB Cluster 8.0 is available for production use; new deployments intended for production should use the latest GA release in this series, which is currently NDB Cluster 8.0.34. You can obtain the most recent NDB Cluster 8.0 release from https://dev.mysql.com/downloads/cluster/. For information about new features and other important changes in this series, see Section 23.2.4, “What is New in MySQL NDB Cluster”.

You can obtain and compile NDB Cluster from source (see Section 23.3.1.4, “Building NDB Cluster from Source on Linux”, and Section 23.3.2.2, “Compiling and Installing NDB Cluster from Source on Windows”), but for all but the most specialized cases, we recommend using one of the following installers provided by Oracle that is appropriate to your operating platform and circumstances:

- Linux binary release (`tar.gz` file)

- Linux RPM package
- Linux `.deb` file
- Windows binary “no-install” release
- Windows MSI Installer

Installation packages may also be available from your platform's package management system.

You can determine whether your MySQL Server has `NDB` support using one of the statements `SHOW VARIABLES LIKE 'have_%'`, `SHOW ENGINES`, or `SHOW PLUGINS`.

A.10.2. What do “NDB” and “NDBCLUSTER” mean?

“NDB” stands for “**N**etwork **D**ata**b**ase”. `NDB` and `NDBCLUSTER` are both names for the storage engine that enables clustering support with MySQL. `NDB` is preferred, but either name is correct.

A.10.3. What is the difference between using NDB Cluster versus using MySQL Replication?

In traditional MySQL replication, a source MySQL server updates one or more replicas. Transactions are committed sequentially, and a slow transaction can cause the replica to lag behind the source. This means that if the source fails, it is possible that the replica might not have recorded the last few transactions. If a transaction-safe engine such as `InnoDB` is being used, a transaction is either completed on the replica or not applied at all, but replication does not guarantee that all data on the source and the replica remains consistent at all times. In NDB Cluster, all data nodes are kept in synchrony, and a transaction committed by any one data node is committed for all data nodes. In the event of a data node failure, all remaining data nodes remain in a consistent state.

In short, whereas standard MySQL replication is *asynchronous*, NDB Cluster is *synchronous*.

Asynchronous replication is also available in NDB Cluster. *NDB Cluster Replication* (also sometimes known as “geo-replication”) includes the capability to replicate both between two NDB Clusters, and from an NDB Cluster to a non-Cluster MySQL server. See Section 23.7, “NDB Cluster Replication”.

A.10.4. Do I need any special networking to run NDB Cluster? How do computers in a cluster communicate?

NDB Cluster is intended to be used in a high-bandwidth environment, with computers connecting using TCP/IP. Its performance depends directly upon the connection speed between the cluster's computers. The minimum connectivity requirements for NDB Cluster include a typical 100-megabit Ethernet network or the equivalent. We recommend you use gigabit Ethernet whenever available.

A.10.5. How many computers do I need to run an NDB Cluster, and why?

A minimum of three computers is required to run a viable cluster. However, the minimum *recommended* number of computers in an NDB Cluster is four: one each to run the management and SQL nodes, and two computers to serve as data nodes. The purpose of the two data nodes is to provide redundancy; the management node must run on a separate machine to guarantee continued arbitration services in the event that one of the data nodes fails.

To provide increased throughput and high availability, you should use multiple SQL nodes (MySQL Servers connected to the cluster). It is also possible (although not strictly necessary) to run multiple management servers.

A.10.6. What do the different computers do in an NDB Cluster?

An NDB Cluster has both a physical and logical organization, with computers being the physical elements. The logical or functional elements of a cluster are referred to as *nodes*, and a computer housing a cluster node is sometimes referred to as a *cluster host*. There are three types of nodes, each corresponding to a specific role within the cluster. These are:

- **Management node.** This node provides management services for the cluster as a whole, including startup, shutdown, backups, and configuration data for the other nodes. The management node server is implemented as the application `ndb_mgmd`; the management client used to control NDB Cluster is `ndb_mgm`. See Section 23.5.4, “ndb_mgmd — The NDB Cluster Management Server Daemon”, and Section 23.5.5, “ndb_mgm — The NDB Cluster Management Client”, for information about these programs.

- **Data node.** This type of node stores and replicates data. Data node functionality is handled by instances of the `NDB` data node process `ndbd`. For more information, see Section 23.5.1, “ndbd — The NDB Cluster Data Node Daemon”.

- **SQL node.** This is simply an instance of MySQL Server (`mysqld`) that is built with support for the `NDBCLUSTER` storage engine and started with the `--ndb-cluster` option to enable the engine and the `--ndb-connectstring` option to enable it to connect to an NDB Cluster management server. For more about these options, see MySQL Server Options for NDB Cluster.

Note

An *API node* is any application that makes direct use of Cluster data nodes for data storage and retrieval. An SQL node can thus be considered a type of API node that uses a MySQL Server to provide an SQL interface to the Cluster. You can write such applications (that do not depend on a MySQL Server) using the NDB API, which supplies a direct, object-oriented transaction and scanning interface to NDB Cluster data; see NDB Cluster API Overview: The NDB API, for more information.

A.10.7 When I run the `SHOW` command in the NDB Cluster management client, I see a line of output that looks like this:

```
id=2    @10.100.10.32  (Version: 8.0.34-ndb-8.0.34 Nodegroup: 0, *)
```

What does the `*` mean? How is this node different from the others?

The simplest answer is, “It's not something you can control, and it's nothing that you need to worry about in any case, unless you're a software engineer writing or analyzing the NDB Cluster source code”.

If you don't find that answer satisfactory, here's a longer and more technical version:

A number of mechanisms in NDB Cluster require distributed coordination among the data nodes. These distributed algorithms and protocols include global checkpointing, DDL (schema) changes, and node restart handling. To make this coordination simpler, the data nodes “elect” one of their number to act as leader. There is no user-facing mechanism for influencing this selection, which is completely automatic; the fact that it *is* automatic is a key part of NDB Cluster's internal architecture.

When a node acts as the “leader” for any of these mechanisms, it is usually the point of coordination for the activity, and the other nodes act as “followers”, carrying out their parts of the activity as directed by the leader. If the node acting as leader fails, then the remaining nodes elect a new leader. Tasks in progress that were being coordinated by the old leader may either fail or be continued by the new leader, depending on the actual mechanism involved.

It is possible for some of these different mechanisms and protocols to have different leader nodes, but in general the same leader is chosen for all of them. The node indicated as the leader in the output of `SHOW` in the management client is known internally as the `DICT` manager, responsible for coordinating DDL and metadata activity.

NDB Cluster is designed in such a way that the choice of leader has no discernible effect outside the cluster itself. For example, the current leader does not have significantly higher CPU or resource usage than the other data nodes, and failure of the leader should not have a significantly different impact on the cluster than the failure of any other data node.

A.10.8. With which operating systems can I use NDB Cluster?

NDB Cluster is supported on most Unix-like operating systems. NDB Cluster is also supported in production settings on Microsoft Windows operating systems.

For more detailed information concerning the level of support which is offered for NDB Cluster on various operating system versions, operating system distributions, and hardware platforms, please refer to https://www.mysql.com/support/supportedplatforms/cluster.html.

A.10.9. What are the hardware requirements for running NDB Cluster?

NDB Cluster should run on any platform for which `NDB`-enabled binaries are available. For data nodes and API nodes, faster CPUs and more memory are likely to improve performance, and 64-bit CPUs are likely to be more effective than 32-bit processors. There must be sufficient memory on machines used for data nodes to hold each node's share of the database (see *How much RAM do I Need?* for more information). For a computer which is used only for running the NDB Cluster management server, the requirements are minimal; a common desktop PC (or the equivalent) is generally sufficient for this task. Nodes can communicate through the standard TCP/IP network and hardware. They can also use the high-speed SCI protocol; however, special networking hardware and software are required to use SCI (see Section 23.4.4, “Using High-Speed Interconnects with NDB Cluster”).

A.10.10. How much RAM do I need to use NDB Cluster? Is it possible to use disk memory at all?

NDB Cluster was originally implemented as in-memory only, but all versions currently available also provide the ability to store NDB Cluster on disk. See Section 23.6.11, “NDB Cluster Disk Data Tables”, for more information.

For in-memory `NDB` tables, you can use the following formula for obtaining a rough estimate of how much RAM is needed for each data node in the cluster:

```
(SizeofDatabase × NumberOfReplicas × 1.1 ) / NumberOfDataNodes
```

To calculate the memory requirements more exactly requires determining, for each table in the cluster database, the storage space required per row (see Section 11.7, “Data Type Storage Requirements”, for details), and multiplying this by the number of rows. You must also remember to account for any column indexes as follows:

- Each primary key or hash index created for an `NDBCLUSTER` table requires 21−25 bytes per record. These indexes use `IndexMemory`.
- Each ordered index requires 10 bytes storage per record, using `DataMemory`.
- Creating a primary key or unique index also creates an ordered index, unless this index is created with `USING HASH`. In other words:
  - A primary key or unique index on a Cluster table normally takes up 31 to 35 bytes per record.
  - However, if the primary key or unique index is created with `USING HASH`, then it requires only 21 to 25 bytes per record.

Creating NDB Cluster tables with `USING HASH` for all primary keys and unique indexes generally causes table updates to run more quickly—in some cases by a much as 20 to 30 percent faster than updates on tables where `USING HASH` was not used in creating primary and unique keys. This is due to the fact that less memory is required (because no ordered indexes are created), and that less CPU must be utilized (because fewer indexes must be read and possibly updated). However, it also means that queries that could otherwise use range scans must be satisfied by other means, which can result in slower selects.

When calculating Cluster memory requirements, you may find useful the `ndb_size.pl` utility which is available in recent MySQL 8.0 releases. This Perl script connects to a current (non-Cluster) MySQL database and creates a report on how much space that database would require if it used the `NDBCLUSTER` storage engine. For more information, see Section 23.5.28, “ndb_size.pl — NDBCLUSTER Size Requirement Estimator”.

It is especially important to keep in mind that *every NDB Cluster table must have a primary key*. The `NDB` storage engine creates a primary key automatically if none is defined; this primary key is created without `USING HASH`.

You can determine how much memory is being used for storage of NDB Cluster data and indexes at any given time using the `REPORT MEMORYUSAGE` command in the `ndb_mgm` client; see Section 23.6.1, “Commands in the NDB Cluster Management Client”, for more information. In addition, warnings are written to the cluster log when 80% of available `DataMemory` or (prior to NDB 7.6) `IndexMemory` is in use, and again when usage reaches 90%, 99%, and 100%.

A.10.11 What file systems can I use with NDB Cluster? What about network file systems or network shares?

Generally, any file system that is native to the host operating system should work well with NDB Cluster. If you find that a given file system works particularly well (or not so especially well) with NDB Cluster, we invite you to discuss your findings in the NDB Cluster Forums.

For Windows, we recommend that you use `NTFS` file systems for NDB Cluster, just as we do for standard MySQL. We do not test NDB Cluster with `FAT` or `VFAT` file systems. Because of this, we do not recommend their use with MySQL or NDB Cluster.

NDB Cluster is implemented as a shared-nothing solution; the idea behind this is that the failure of a single piece of hardware should not cause the failure of multiple cluster nodes, or possibly even the failure of the cluster as a whole. For this reason, the use of network shares or network file systems is not supported for NDB Cluster. This also applies to shared storage devices such as SANs.

A.10.12 Can I run NDB Cluster nodes inside virtual machines (such as those created by VMWare, VirtualBox, Parallels, or Xen)?

NDB Cluster is supported for use in virtual machines. We currently support and test using Oracle VM.

Some NDB Cluster users have successfully deployed NDB Cluster using other virtualization products; in such cases, Oracle can provide NDB Cluster support, but issues specific to the virtual environment must be referred to that product's vendor.

A.10.13 I am trying to populate an NDB Cluster database. The loading process terminates prematurely and I get an error message like this one:

```
ERROR 1114: The table 'my_cluster_table' is full
```

Why is this happening?

The cause is very likely to be that your setup does not provide sufficient RAM for all table data and all indexes, *including the primary key required by the `NDB` storage engine and automatically created in the event that the table definition does not include the definition of a primary key*.

It is also worth noting that all data nodes should have the same amount of RAM, since no data node in a cluster can use more memory than the least amount available to any individual data node. For example, if there are four computers hosting Cluster data nodes, and three of these have 3GB of RAM available to store Cluster data while the remaining data node has only 1GB RAM, then each data node can devote at most 1GB to NDB Cluster data and indexes.

In some cases it is possible to get `Table is full` errors in MySQL client applications even when `ndb_mgm -e "ALL REPORT MEMORYUSAGE"` shows significant free `DataMemory`. You can force `NDB` to create extra partitions for NDB Cluster tables and thus have more memory available for hash indexes by using the `MAX_ROWS` option for `CREATE TABLE`. In general, setting `MAX_ROWS` to twice the number of rows that you expect to store in the table should be sufficient.

For similar reasons, you can also sometimes encounter problems with data node restarts on nodes that are heavily loaded with data. The `MinFreePct` parameter can help with this issue by reserving a portion (5% by default) of `DataMemory` and (prior to NDB 7.6) `IndexMemory` for use in restarts. This reserved memory is not available for storing `NDB` tables or data.

A.10.14. NDB Cluster uses TCP/IP. Does this mean that I can run it over the Internet, with one or more nodes in remote locations?

It is *very* unlikely that a cluster would perform reliably under such conditions, as NDB Cluster was designed and implemented with the assumption that it would be run under conditions guaranteeing dedicated high-speed connectivity such as that found in a LAN setting using 100 Mbps or gigabit Ethernet—preferably the latter. We neither test nor warrant its performance using anything slower than this.

Also, it is extremely important to keep in mind that communications between the nodes in an NDB Cluster are not secure; they are neither encrypted nor safeguarded by any other protective mechanism. The most secure configuration for a cluster is in a private network behind a firewall, with no direct access to any Cluster data or management nodes from outside. (For SQL nodes, you should take the same precautions as you would with any other instance of the MySQL server.) For more information, see Section 23.6.20, “NDB Cluster Security Issues”.

A.10.15. Do I have to learn a new programming or query language to use NDB Cluster?

No. Although some specialized commands are used to manage and configure the cluster itself, only standard (My)SQL statements are required for the following operations:

- Creating, altering, and dropping tables
- Inserting, updating, and deleting table data
- Creating, changing, and dropping primary and unique indexes

Some specialized configuration parameters and files are required to set up an NDB Cluster—see Section 23.4.3, “NDB Cluster Configuration Files”, for information about these.

A few simple commands are used in the NDB Cluster management client (`ndb_mgm`) for tasks such as starting and stopping cluster nodes. See Section 23.6.1, “Commands in the NDB Cluster Management Client”.

A.10.16. What programming languages and APIs are supported by NDB Cluster?

NDB Cluster supports the same programming APIs and languages as the standard MySQL Server, including ODBC, .Net, the MySQL C API, and numerous drivers for popular scripting languages such as PHP, Perl, and Python. NDB Cluster applications written using these APIs

behave similarly to other MySQL applications; they transmit SQL statements to a MySQL Server (in the case of NDB Cluster, an SQL node), and receive responses containing rows of data. For more information about these APIs, see Chapter 29, *Connectors and APIs*.

NDB Cluster also supports application programming using the NDB API, which provides a low-level C++ interface to NDB Cluster data without needing to go through a MySQL Server. See The NDB API. In addition, many `NDBCLUSTER` management functions are exposed by the C-language MGM API; see The MGM API, for more information.

NDB Cluster also supports Java application programming using ClusterJ, which supports a domain object model of data using sessions and transactions. See Java and NDB Cluster, for more information.

In addition, NDB Cluster provides support for `memcached`, allowing developers to access data stored in NDB Cluster using the `memcached` interface; for more information, see ndbmemcache—Memcache API for NDB Cluster (NO LONGER SUPPORTED).

NDB Cluster also includes adapters supporting NoSQL applications written against `Node.js`, with NDB Cluster as the data store. See MySQL NoSQL Connector for JavaScript, for more information.

A.10.17 Does NDB Cluster include any management tools?

NDB Cluster includes a command line client for performing basic management functions. See Section 23.5.5, “ndb_mgm — The NDB Cluster Management Client”, and Section 23.6.1, “Commands in the NDB Cluster Management Client”.

NDB Cluster 7.6 and earlier are also supported by MySQL Cluster Manager, a separate product providing an advanced command line interface that can automate many NDB Cluster management tasks such as rolling restarts and configuration changes. Beginning with version 1.4.8, MySQL Cluster Manager also provides experimental support for NDB Cluster 8.0. For more information about MySQL Cluster Manager, see MySQL Cluster Manager 1.4.8 User Manual.

A.10.18 How do I find out what an error or warning message means when using NDB Cluster?

There are two ways in which this can be done:

- From within the `mysql` client, use `SHOW ERRORS` or `SHOW WARNINGS` immediately upon being notified of the error or warning condition.
- From a system shell prompt, use `perror --ndb error_code`.

A.10.19 Is NDB Cluster transaction-safe? What isolation levels are supported?

Yes. For tables created with the `NDB` storage engine, transactions are supported. Currently, NDB Cluster supports only the `READ COMMITTED` transaction isolation level.

A.10.20 What storage engines are supported by NDB Cluster?

NDB Cluster requires the `NDB` storage engine. That is, in order for a table to be shared between nodes in an NDB Cluster, the table must be created using `ENGINE=NDB` (or the equivalent option `ENGINE=NDBCLUSTER`).

It is possible to create tables using other storage engines (such as `InnoDB` or `MyISAM`) on a MySQL server being used with NDB Cluster, but since these tables do not use `NDB`, they do not participate in clustering; each such table is strictly local to the individual MySQL server instance on which it is created.

NDB Cluster is quite different from `InnoDB` clustering with regard to architecture, requirements, and implementation; despite any similarity in their names, the two are not compatible. For more

information about `InnoDB` clustering, see MySQL AdminAPI. See also Section 23.2.6, “MySQL Server Using InnoDB Compared with NDB Cluster”, for information about the differences between the `NDB` and `InnoDB` storage engines.

A.10.21. In the event of a catastrophic failure— for example, the whole city loses power *and* my UPS fails —would I lose all my data?

All committed transactions are logged. Therefore, although it is possible that some data could be lost in the event of a catastrophe, this should be quite limited. Data loss can be further reduced by minimizing the number of operations per transaction. (It is not a good idea to perform large numbers of operations per transaction in any case.)

A.10.22. Is it possible to use `FULLTEXT` indexes with NDB Cluster?

`FULLTEXT` indexing is currently supported only by the `InnoDB` and `MyISAM` storage engines. See Section 12.10, “Full-Text Search Functions”, for more information.

A.10.23. Can I run multiple nodes on a single computer?

It is possible but not always advisable. One of the chief reasons to run a cluster is to provide redundancy. To obtain the full benefits of this redundancy, each node should reside on a separate machine. If you place multiple nodes on a single machine and that machine fails, you lose all of those nodes. For this reason, if you do run multiple data nodes on a single machine, it is *extremely* important that they be set up in such a way that the failure of this machine does not cause the loss of all the data nodes in a given node group.

Given that NDB Cluster can be run on commodity hardware loaded with a low-cost (or even no-cost) operating system, the expense of an extra machine or two is well worth it to safeguard mission-critical data. It also worth noting that the requirements for a cluster host running a management node are minimal. This task can be accomplished with a 300 MHz Pentium or equivalent CPU and sufficient RAM for the operating system, plus a small amount of overhead for the `ndb_mgmd` and `ndb_mgm` processes.

It is acceptable to run multiple cluster data nodes on a single host that has multiple CPUs, cores, or both. The NDB Cluster distribution also provides a multithreaded version of the data node binary intended for use on such systems. For more information, see Section 23.5.3, “ndbmtd — The NDB Cluster Data Node Daemon (Multi-Threaded)”.

It is also possible in some cases to run data nodes and SQL nodes concurrently on the same machine; how well such an arrangement performs is dependent on a number of factors such as number of cores and CPUs as well as the amount of disk and memory available to the data node and SQL node processes, and you must take these factors into account when planning such a configuration.

A.10.24. Can I add data nodes to an NDB Cluster without restarting it?

It is possible to add new data nodes to a running NDB Cluster without taking the cluster offline. For more information, see Section 23.6.7, “Adding NDB Cluster Data Nodes Online”.

For other types of NDB Cluster nodes, a rolling restart is all that is required (see Section 23.6.5, “Performing a Rolling Restart of an NDB Cluster”).

A.10.25. Are there any limitations that I should be aware of when using NDB Cluster?

Limitations on `NDB` tables in MySQL NDB Cluster include the following:

- Temporary tables are not supported; a `CREATE TEMPORARY TABLE` statement using `ENGINE=NDB` or `ENGINE=NDBCLUSTER` fails with an error.
- The only types of user-defined partitioning supported for `NDBCLUSTER` tables are `KEY` and `LINEAR KEY`. Trying to create an `NDB` table using any other partitioning type fails with an error.

- `FULLTEXT` indexes are not supported.
- Index prefixes are not supported. Only complete columns may be indexed.
- Spatial indexes are not supported (although spatial columns can be used). See Section 11.4, “Spatial Data Types”.
- Support for partial transactions and partial rollbacks is comparable to that of other transactional storage engines such as `InnoDB` that can roll back individual statements.
- The maximum number of attributes allowed per table is 512. Attribute names cannot be any longer than 31 characters. For each table, the maximum combined length of the table and database names is 122 characters.
- Priot to NDB 8.0, the maximum size for a table row is 14 kilobytes, not counting `BLOB` values. In NDB 8.0, this maximum is increased to 30000 bytes. See Section 23.2.7.5, “Limits Associated with Database Objects in NDB Cluster”, for more information.

  There is no set limit for the number of rows per `NDB` table. Limits on table size depend on a number of factors, in particular on the amount of RAM available to each data node.

For a complete listing of limitations in NDB Cluster, see Section 23.2.7, “Known Limitations of NDB Cluster”. See also Section 23.2.7.11, “Previous NDB Cluster Issues Resolved in NDB Cluster 8.0”.

A.10.26 Does NDB Cluster support foreign keys?

NDB Cluster provides support for foreign key constraints which is comparable to that found in the `InnoDB` storage engine; see Section 1.6.3.2, “FOREIGN KEY Constraints”, for more detailed information, as well as Section 13.1.20.5, “FOREIGN KEY Constraints”. Applications requiring foreign key support should use NDB Cluster 7.3, 7.4, 7.5, or later.

A.10.27 How do I import an existing MySQL database into an NDB Cluster?

You can import databases into NDB Cluster much as you would with any other version of MySQL. Other than the limitations mentioned elsewhere in this FAQ, the only other special requirement is that any tables to be included in the cluster must use the `NDB` storage engine. This means that the tables must be created with `ENGINE=NDB` or `ENGINE=NDBCLUSTER`.

It is also possible to convert existing tables that use other storage engines to `NDBCLUSTER` using one or more `ALTER TABLE` statement. However, the definition of the table must be compatible with the `NDBCLUSTER` storage engine prior to making the conversion. In MySQL 8.0, an additional workaround is also required; see Section 23.2.7, “Known Limitations of NDB Cluster”, for details.

A.10.28 How do NDB Cluster nodes communicate with one another?

Cluster nodes can communicate through any of three different transport mechanisms: TCP/IP, SHM (shared memory), and SCI (Scalable Coherent Interface). Where available, SHM is used by default between nodes residing on the same cluster host; however, this is considered experimental. SCI is a high-speed (1 gigabit per second and higher), high-availability protocol used in building scalable multi-processor systems; it requires special hardware and drivers. See Section 23.4.4, “Using High-Speed Interconnects with NDB Cluster”, for more about using SCI as a transport mechanism for NDB Cluster.

A.10.29 What is an *arbitrator*?

If one or more data nodes in a cluster fail, it is possible that not all cluster data nodes are able to “see” one another. In fact, it is possible that two sets of data nodes might become isolated from one another in a network partitioning, also known as a “split-brain” scenario. This type of

situation is undesirable because each set of data nodes tries to behave as though it is the entire cluster. An arbitrator is required to decide between the competing sets of data nodes.

When all data nodes in at least one node group are alive, network partitioning is not an issue, because no single subset of the cluster can form a functional cluster on its own. The real problem arises when no single node group has all its nodes alive, in which case network partitioning (the “split-brain” scenario) becomes possible. Then an arbitrator is required. All cluster nodes recognize the same node as the arbitrator, which is normally the management server; however, it is possible to configure any of the MySQL Servers in the cluster to act as the arbitrator instead. The arbitrator accepts the first set of cluster nodes to contact it, and tells the remaining set to shut down. Arbitrator selection is controlled by the `ArbitrationRank` configuration parameter for MySQL Server and management server nodes. You can also use the `ArbitrationRank` configuration parameter to control the arbitrator selection process. For more information about these parameters, see Section 23.4.3.5, “Defining an NDB Cluster Management Server”.

The role of arbitrator does not in and of itself impose any heavy demands upon the host so designated, and thus the arbitrator host does not need to be particularly fast or to have extra memory especially for this purpose.

A.10.30 What data types are supported by NDB Cluster?

NDB Cluster supports all of the usual MySQL data types, including those associated with MySQL's spatial extensions; however, the `NDB` storage engine does not support spatial indexes. (Spatial indexes are supported only by `MyISAM`; see Section 11.4, “Spatial Data Types”, for more information.) In addition, there are some differences with regard to indexes when used with `NDB` tables.

Note

NDB Cluster Disk Data tables (that is, tables created with `TABLESPACE ... STORAGE DISK ENGINE=NDB` or `TABLESPACE ... STORAGE DISK ENGINE=NDBCLUSTER`) have only fixed-width rows. This means that (for example) each Disk Data table record containing a `VARCHAR(255)` column requires space for 255 characters (as required for the character set and collation being used for the table), regardless of the actual number of characters stored therein.

See Section 23.2.7, “Known Limitations of NDB Cluster”, for more information about these issues.

A.10.31 How do I start and stop NDB Cluster?

It is necessary to start each node in the cluster separately, in the following order:

1. Start the management node, using the `ndb_mgmd` command.

   You must include the `-f` or `--config-file` option to tell the management node where its configuration file can be found.

2. Start each data node with the `ndbd` command.

   Each data node must be started with the `-c` or `--ndb-connectstring` option so that the data node knows how to connect to the management server.

3. Start each MySQL Server (SQL node) using your preferred startup script, such as `mysqld_safe`.

   Each MySQL Server must be started with the `--ndbcluster` and `--ndb-connectstring` options. These options cause `mysqld` to enable `NDBCLUSTER` storage engine support and how to connect to the management server.

Each of these commands must be run from a system shell on the machine housing the affected node. (You do not have to be physically present at the machine—a remote login shell can be used for this purpose.) You can verify that the cluster is running by starting the NDB management client `ndb_mgm` on the machine housing the management node and issuing the `SHOW` or `ALL STATUS` command.

To shut down a running cluster, issue the command `SHUTDOWN` in the management client. Alternatively, you may enter the following command in a system shell:

```
$> ndb_mgm -e "SHUTDOWN"
```

(The quotation marks in this example are optional, since there are no spaces in the command string following the `-e` option; in addition, the `SHUTDOWN` command, like other management client commands, is not case-sensitive.)

Either of these commands causes the `ndb_mgm`, `ndb_mgm`, and any `ndbd` processes to terminate gracefully. MySQL servers running as SQL nodes can be stopped using `mysqladmin shutdown`.

For more information, see Section 23.6.1, “Commands in the NDB Cluster Management Client”, and Section 23.3.6, “Safe Shutdown and Restart of NDB Cluster”.

MySQL Cluster Manager provides additional ways to handle starting ansd stopping of NDB Cluster nodes. See MySQL Cluster Manager 1.4.8 User Manual, for more information about this tool.

A.10.32. What happens to NDB Cluster data when the NDB Cluster is shut down?

The data that was held in memory by the cluster's data nodes is written to disk, and is reloaded into memory the next time that the cluster is started.

A.10.33. Is it a good idea to have more than one management node for an NDB Cluster?

It can be helpful as a fail-safe. Only one management node controls the cluster at any given time, but it is possible to configure one management node as primary, and one or more additional management nodes to take over in the event that the primary management node fails.

See Section 23.4.3, “NDB Cluster Configuration Files”, for information on how to configure NDB Cluster management nodes.

A.10.34. Can I mix different kinds of hardware and operating systems in one NDB Cluster?

Yes, as long as all machines and operating systems have the same “endianness” (all big-endian or all little-endian).

It is also possible to use software from different NDB Cluster releases on different nodes. However, we support such use only as part of a rolling upgrade procedure (see Section 23.6.5, “Performing a Rolling Restart of an NDB Cluster”).

A.10.35. Can I run two data nodes on a single host? Two SQL nodes?

Yes, it is possible to do this. In the case of multiple data nodes, it is advisable (but not required) for each node to use a different data directory. If you want to run multiple SQL nodes on one machine, each instance of `mysqld` must use a different TCP/IP port.

Running data nodes and SQL nodes together on the same host is possible, but you should be aware that the `ndbd` or `ndbmtd` processes may compete for memory with `mysqld`.

A.10.36. Can I use host names with NDB Cluster?

Yes, it is possible to use DNS and DHCP for cluster hosts. However, if your application requires “five nines” availability, you should use fixed (numeric) IP addresses, since making

communication between Cluster hosts dependent on services such as DNS and DHCP introduces additional potential points of failure.

A.10.37 Does NDB Cluster support IPv6?

IPv6 is supported for connections between SQL nodes (MySQL servers), but connections between all other types of NDB Cluster nodes must use IPv4.

In practical terms, this means that you can use IPv6 for replication between NDB Clusters, but connections between nodes in the same NDB Cluster must use IPv4. For more information, see Section 23.7.3, “Known Issues in NDB Cluster Replication”.

A.10.38 How do I handle MySQL users in an NDB Cluster having multiple MySQL servers?

MySQL user accounts and privileges are normally not automatically propagated between different MySQL servers accessing the same NDB Cluster. MySQL NDB Cluster provides support for shared and synchronized users and privileges using the `NDB_STORED_USER` privilege; see Distributed Privileges Using Shared Grant Tables, for more information. You should be aware that this implementation is new to NDB 8.0 and is not compatible with the shared privileges mechanism employed in earlier versions of NDB Cluster, which is no longer supported in NDB 8.0.

A.10.39 How do I continue to send queries in the event that one of the SQL nodes fails?

MySQL NDB Cluster does not provide any sort of automatic failover between SQL nodes. Your application must be prepared to handle the loss of SQL nodes and to fail over between them.

A.10.40 How do I back up and restore an NDB Cluster?

You can use the NDB Cluster native backup and restore functionality in the NDB management client and the `ndb_restore` program. See Section 23.6.8, “Online Backup of NDB Cluster”, and Section 23.5.23, “ndb_restore — Restore an NDB Cluster Backup”.

You can also use the traditional functionality provided for this purpose in `mysqldump` and the MySQL server. See Section 4.5.4, “mysqldump — A Database Backup Program”, for more information.

A.10.41 What is an “angel process”?

This process monitors and, if necessary, attempts to restart the data node process. If you check the list of active processes on your system after starting `ndbd`, you can see that there are actually 2 processes running by that name, as shown here (we omit the output from `ndb_mgmd` and `ndbd` for brevity):

```
$> ./ndb_mgmd

$> ps aux | grep ndb
me      23002  0.0  0.0 122948  3104 ?        Ssl  14:14   0:00 ./ndb_mgmd
me      23025  0.0  0.0   5284   820 pts/2    S+   14:14   0:00 grep ndb

$> ./ndbd -c 127.0.0.1 --initial

$> ps aux | grep ndb
me      23002  0.0  0.0 123080  3356 ?        Ssl  14:14   0:00 ./ndb_mgmd
me      23096  0.0  0.0  35876  2036 ?        Ss   14:14   0:00 ./ndbmtd -c 127.0.0.1 --initial
me      23097  1.0  2.4 524116 91096 ?        Sl   14:14   0:00 ./ndbmtd -c 127.0.0.1 --initial
me      23168  0.0  0.0   5284   812 pts/2    R+   14:15   0:00 grep ndb
```

The `ndbd` process showing `0.0` for both memory and CPU usage is the angel process (although it actually does use a very small amount of each). This process merely checks to see if the main `ndbd` or `ndbmtd` process (the primary data node process which actually handles the data) is running. If permitted to do so (for example, if the `StopOnError` configuration parameter is set to `false`), the angel process tries to restart the primary data node process.

A.11 MySQL 8.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets

This set of Frequently Asked Questions derives from the experience of MySQL's Support and Development groups in handling many inquiries about CJK (Chinese-Japanese-Korean) issues.

A.11.1 What CJK character sets are available in MySQL?

The list of CJK character sets may vary depending on your MySQL version. For example, the gb18030 character set is not supported prior to MySQL 5.7.4. However, since the name of the applicable language appears in the DESCRIPTION column for every entry in the INFORMATION_SCHEMA.CHARACTER_SETS table, you can obtain a current list of all the non-Unicode CJK character sets using this query:

```
mysql> SELECT CHARACTER_SET_NAME, DESCRIPTION
       FROM INFORMATION_SCHEMA.CHARACTER_SETS
       WHERE DESCRIPTION LIKE '%Chin%'
       OR DESCRIPTION LIKE '%Japanese%'
       OR DESCRIPTION LIKE '%Korean%'
       ORDER BY CHARACTER_SET_NAME;
+--------------------+---------------------------------+
| CHARACTER_SET_NAME | DESCRIPTION                     |
+--------------------+---------------------------------+
| big5               | Big5 Traditional Chinese        |
| cp932              | SJIS for Windows Japanese       |
| eucjpms            | UJIS for Windows Japanese       |
| euckr              | EUC-KR Korean                   |
| gb18030            | China National Standard GB18030 |
| gb2312             | GB2312 Simplified Chinese       |
| gbk                | GBK Simplified Chinese          |
| sjis               | Shift-JIS Japanese              |
| ujis               | EUC-JP Japanese                 |
+--------------------+---------------------------------+
```

(For more information, see Section 26.3.4, “The INFORMATION_SCHEMA CHARACTER_SETS Table”.)

MySQL supports three variants of the *GB* (*Guojia Biaozhun*, or *National Standard*, or *Simplified Chinese*) character sets which are official in the People's Republic of China: `gb2312`, `gbk`, and (as of MySQL 5.7.4) `gb18030`.

Sometimes people try to insert `gbk` characters into `gb2312`, and it works most of the time because `gbk` is a superset of `gb2312`. But eventually they try to insert a rarer Chinese character and it does not work. (For an example, see Bug #16072).

Here, we try to clarify exactly what characters are legitimate in `gb2312` or `gbk`, with reference to the official documents. Please check these references before reporting `gb2312` or `gbk` bugs:

- The MySQL `gbk` character set is in reality “Microsoft code page 936”. This differs from the official `gbk` for characters `A1A4` (middle dot), `A1AA` (em dash), `A6E0-A6F5`, and `A8BB-A8C0`.
- For a listing of `gbk`/Unicode mappings, see http://www.unicode.org/Public/MAPPINGS/VENDORS/MICSFT/WINDOWS/CP936.TXT.

It is also possible to store CJK characters in Unicode character sets, although the available collations may not sort characters quite as you expect:

- The `utf8` and `ucs2` character sets support the characters from Unicode Basic Multilingual Plane (BMP). These characters have code point values between `U+0000` and `U+FFFF`.
- The `utf8mb4`, `utf16`, `utf16le`, and `utf32` character sets support BMP characters, as well as supplementary characters that lie outside the BMP. Supplementary characters have code point values between `U+10000` and `U+10FFFF`.

The collation used for a Unicode character set determines the ability to sort (that is, distinguish) characters in the set:

- Collations based on Unicode Collation Algorithm (UCA) 4.0.0 distinguish only BMP characters.
- Collations based on UCA 5.2.0 or 9.0.0 distinguish BMP and supplementary characters.
- Non-UCA collations may not distinguish all Unicode characters. For example, the `utf8mb4` default collation is `utf8mb4_general_ci`, which distinguishes only BMP characters.

Moreover, distinguishing characters is not the same as ordering them per the conventions of a given CJK language. Currently, MySQL has only one CJK-specific UCA collation, `gb18030_unicode_520_ci` (which requires use of the non-Unicode `gb18030` character set).

For information about Unicode collations and their differentiating properties, including collation properties for supplementary characters, see Section 10.10.1, “Unicode Character Sets”.

A.11.2. I have inserted CJK characters into my table. Why does `SELECT` display them as “?” characters?

This problem is usually due to a setting in MySQL that does not match the settings for the application program or the operating system. Here are some common steps for correcting these types of issues:

- *Be certain of what MySQL version you are using*.

  Use the statement `SELECT VERSION();` to determine this.

- *Make sure that the database is actually using the desired character set*.

  People often think that the client character set is always the same as either the server character set or the character set used for display purposes. However, both of these are

false assumptions. You can make sure by checking the result of `SHOW CREATE TABLE tablename` or, better yet, by using this statement:

```
SELECT character_set_name, collation_name
    FROM information_schema.columns
    WHERE table_schema = your_database_name
        AND table_name = your_table_name
        AND column_name = your_column_name;
```

- *Determine the hexadecimal value of the character or characters that are not being displayed correctly*.

  You can obtain this information for a column `column_name` in the table `table_name` using the following query:

  ```
  SELECT HEX(column_name)
  FROM table_name;
  ```

  `3F` is the encoding for the `?` character; this means that `?` is the character actually stored in the column. This most often happens because of a problem converting a particular character from your client character set to the target character set.

- *Make sure that a round trip is possible. When you select `literal` (or `_introducer hexadecimal-value`), do you obtain `literal` as a result*?

  For example, the Japanese Katakana character *Pe* (`ペ`) exists in all CJK character sets, and has the code point value (hexadecimal coding) `0x30da`. To test a round trip for this character, use this query:

  ```
  SELECT 'ペ' AS `ペ`;         /* or SELECT _ucs2 0x30da; */
  ```

  If the result is not also ペ, the round trip failed.

  For bug reports regarding such failures, we might ask you to follow up with `SELECT HEX('ペ');`. Then we can determine whether the client encoding is correct.

- *Make sure that the problem is not with the browser or other application, rather than with MySQL*.

  Use the `mysql` client program to accomplish this task. If `mysql` displays characters correctly but your application does not, your problem is probably due to system settings.

  To determine your settings, use the `SHOW VARIABLES` statement, whose output should resemble what is shown here:

  ```
  mysql> SHOW VARIABLES LIKE 'char%';
  +--------------------------+----------------------------------------+
  | Variable_name            | Value                                  |
  +--------------------------+----------------------------------------+
  | character_set_client     | utf8                                   |
  | character_set_connection | utf8                                   |
  | character_set_database   | latin1                                 |
  | character_set_filesystem | binary                                 |
  | character_set_results    | utf8                                   |
  | character_set_server     | latin1                                 |
  | character_set_system     | utf8                                   |
  | character_sets_dir       | /usr/local/mysql/share/mysql/charsets/ |
  +--------------------------+----------------------------------------+
  ```

  These are typical character-set settings for an international-oriented client (notice the use of `utf8` Unicode) connected to a server in the West (`latin1` is a West Europe character set).

  Although Unicode (usually the `utf8` variant on Unix, and the `ucs2` variant on Windows) is preferable to Latin, it is often not what your operating system utilities support best. Many

Windows users find that a Microsoft character set, such as `cp932` for Japanese Windows, is suitable.

If you cannot control the server settings, and you have no idea what setting your underlying computer uses, try changing to a common character set for the country that you're in (`euckr` = Korea; `gb18030`, `gb2312` or `gbk` = People's Republic of China; `big5` = Taiwan; `sjis`, `ujis`, `cp932`, or `eucjpms` = Japan; `ucs2` or `utf8` = anywhere). Usually it is necessary to change only the client and connection and results settings. The `SET NAMES`. statement changes all three at once. For example:

```
SET NAMES 'big5';
```

Once the setting is correct, you can make it permanent by editing `my.cnf` or `my.ini`. For example you might add lines looking like these:

```
[mysqld]
character-set-server=big5
[client]
default-character-set=big5
```

It is also possible that there are issues with the API configuration setting being used in your application; see *Why does my GUI front end or browser not display CJK characters correctly...?* for more information.

A.11.3. What problems should I be aware of when working with the Big5 Chinese character set?

MySQL supports the Big5 character set which is common in Hong Kong and Taiwan (Republic of China). The MySQL `big5` character set is in reality Microsoft code page 950, which is very similar to the original `big5` character set.

A feature request for adding `HKSCS` extensions has been filed. People who need this extension may find the suggested patch for Bug #13577 to be of interest.

A.11.4. Why do Japanese character set conversions fail?

MySQL supports the `sjis`, `ujis`, `cp932`, and `eucjpms` character sets, as well as Unicode. A common need is to convert between character sets. For example, there might be a Unix server (typically with `sjis` or `ujis`) and a Windows client (typically with `cp932`).

In the following conversion table, the `ucs2` column represents the source, and the `sjis`, `cp932`, `ujis`, and `eucjpms` columns represent the destinations; that is, the last 4 columns provide the hexadecimal result when we use `CONVERT(ucs2)` or we assign a `ucs2` column containing the value to an `sjis`, `cp932`, `ujis`, or `eucjpms` column.

Character Name	ucs2	sjis	cp932	ujis	eucjpms
BROKEN BAR	00A6	3F	3F	8FA2C3	3F
FULLWIDTH BROKEN BAR	FFE4	3F	FA55	3F	8FA2
YEN SIGN	00A5	3F	3F	20	3F
FULLWIDTH YEN SIGN	FFE5	818F	818F	A1EF	3F
TILDE	007E	7E	7E	7E	7E
OVERLINE	203E	3F	3F	20	3F
HORIZONTAL BAR	2015	815C	815C	A1BD	A1BD
EM DASH	2014	3F	3F	3F	3F

Character Name	ucs2	sjis	cp932	ujis	eucjpms
REVERSE SOLIDUS	005C	815F	5C	5C	5C
FULLWIDTH REVERSE SOLIDUS	FF3C	3F	815F	3F	A1C0
WAVE DASH	301C	8160	3F	A1C1	3F
FULLWIDTH TILDE	FF5E	3F	8160	3F	A1C1
DOUBLE VERTICAL LINE	2016	8161	3F	A1C2	3F
PARALLEL TO	2225	3F	8161	3F	A1C2
MINUS SIGN	2212	817C	3F	A1DD	3F
FULLWIDTH HYPHEN-MINUS	FF0D	3F	817C	3F	A1DD
CENT SIGN	00A2	8191	3F	A1F1	3F
FULLWIDTH CENT SIGN	FFE0	3F	8191	3F	A1F1
POUND SIGN	00A3	8192	3F	A1F2	3F
FULLWIDTH POUND SIGN	FFE1	3F	8192	3F	A1F2
NOT SIGN	00AC	81CA	3F	A2CC	3F
FULLWIDTH NOT SIGN	FFE2	3F	81CA	3F	A2CC

Now consider the following portion of the table.

	ucs2	sjis	cp932
NOT SIGN	00AC	81CA	3F
FULLWIDTH NOT SIGN	FFE2	3F	81CA

This means that MySQL converts the `NOT SIGN` (Unicode `U+00AC`) to `sjis` code point `0x81CA` and to `cp932` code point `3F`. (`3F` is the question mark (“?”. This is what is always used when the conversion cannot be performed.)

A.11.5 What should I do if I want to convert SJIS `81CA` to `cp932`?

Our answer is: “?”. There are disadvantages to this, and many people would prefer a “loose” conversion, so that `81CA (NOT SIGN)` in `sjis` becomes `81CA (FULLWIDTH NOT SIGN)` in `cp932`.

A.11.6 How does MySQL represent the Yen (`¥`) sign?

A problem arises because some versions of Japanese character sets (both `sjis` and `euc`) treat `5C` as a *reverse solidus* (`\`, also known as a backslash), whereas others treat it as a yen sign (`¥`).

MySQL follows only one version of the JIS (Japanese Industrial Standards) standard description. In MySQL, *5C is always the reverse solidus (\)*.

A.11.7. Of what issues should I be aware when working with Korean character sets in MySQL?

In theory, while there have been several versions of the `euckr` (*Extended Unix Code Korea*) character set, only one problem has been noted. We use the "ASCII" variant of EUC-KR, in which the code point `0x5c` is REVERSE SOLIDUS, that is `\`, instead of the "KS-Roman" variant of EUC-KR, in which the code point `0x5c` is `WON SIGN` (₩). This means that you cannot convert Unicode `U+20A9` to `euckr`:

```
mysql> SELECT
           CONVERT('₩' USING euckr) AS euckr,
           HEX(CONVERT('₩' USING euckr)) AS hexeuckr;
+-------+----------+
| euckr | hexeuckr |
+-------+----------+
| ?     | 3F       |
+-------+----------+
```

A.11.8. Why do I get `Incorrect string value` error messages?

To see the problem, create a table with one Unicode (`ucs2`) column and one Chinese (`gb2312`) column.

```
mysql> CREATE TABLE ch
       (ucs2 CHAR(3) CHARACTER SET ucs2,
       gb2312 CHAR(3) CHARACTER SET gb2312);
```

In nonstrict SQL mode, try to place the rare character 汃 in both columns.

```
mysql> SET sql_mode = '';
mysql> INSERT INTO ch VALUES ('A汃B','A汃B');
Query OK, 1 row affected, 1 warning (0.00 sec)
```

The `INSERT` produces a warning. Use the following statement to see what it is:

```
mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1366
Message: Incorrect string value: '\xE6\xB1\x8CB' for column 'gb2312' at row 1
```

So it is a warning about the `gb2312` column only.

```
mysql> SELECT ucs2,HEX(ucs2),gb2312,HEX(gb2312) FROM ch;
+-------+--------------+--------+-------------+
| ucs2  | HEX(ucs2)    | gb2312 | HEX(gb2312) |
+-------+--------------+--------+-------------+
| A汃B | 00416C4C0042 | A?B    | 413F42      |
+-------+--------------+--------+-------------+
```

Several things need explanation here:

1. The 汃 character is not in the `gb2312` character set, as described earlier.
2. If you are using an old version of MySQL, you may see a different message.
3. A warning occurs rather than an error because MySQL is not set to use strict SQL mode. In nonstrict mode, MySQL tries to do what it can, to get the best fit, rather than give up. With strict SQL mode, the `Incorrect string value` message occurs as an error rather than a warning, and the `INSERT` fails.

A.11.9. Why does my GUI front end or browser display CJK characters incorrectly in my application using Access, PHP, or another API?

Obtain a direct connection to the server using the `mysql` client, and try the same query there. If `mysql` responds correctly, the trouble may be that your application interface requires initialization. Use `mysql` to tell you what character set or sets it uses with the statement `SHOW VARIABLES LIKE 'char%';`. If you are using Access, you are most likely connecting with Connector/ODBC. In this case, you should check Configuring Connector/ODBC. If, for example, you use `big5`, you would enter `SET NAMES 'big5'`. (In this case, no `;` character is required.) If you are using ASP, you might need to add `SET NAMES` in the code. Here is an example that has worked in the past:

```
<%
Session.CodePage=0
Dim strConnection
Dim Conn
strConnection="driver={MySQL ODBC 3.51 Driver};server=server;uid=username;" \
               & "pwd=password;database=database;stmt=SET NAMES 'big5';"
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open strConnection
%>
```

In much the same way, if you are using any character set other than `latin1` with Connector/NET, you must specify the character set in the connection string. See Connector/NET Connections, for more information.

If you are using PHP, try this:

```
<?php
  $link = new mysqli($host, $usr, $pwd, $db);

  if( mysqli_connect_errno() )
  {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
  }

  $link->query("SET NAMES 'utf8'");
?>
```

In this case, we used `SET NAMES` to change `character_set_client`, `character_set_connection`, and `character_set_results`.

Another issue often encountered in PHP applications has to do with assumptions made by the browser. Sometimes adding or changing a `<meta>` tag suffices to correct the problem: for example, to insure that the user agent interprets page content as `UTF-8`, include `<meta http-equiv="Content-Type" content="text/html; charset=utf-8">` in the `<head>` section of the HTML page.

If you are using Connector/J, see Using Character Sets and Unicode.

A.11.10 ve upgraded to MySQL 8.0. How can I revert to behavior like that in MySQL 4.0 with regard to character sets?

In MySQL Version 4.0, there was a single “global” character set for both server and client, and the decision as to which character to use was made by the server administrator. This changed starting with MySQL Version 4.1. What happens now is a “handshake”, as described in Section 10.4, “Connection Character Sets and Collations”:

> When a client connects, it sends to the server the name of the character set that it wants to use. The server uses the name to set the `character_set_client`, `character_set_results`, and `character_set_connection` system variables. In effect, the server performs a `SET NAMES` operation using the character set name.

The effect of this is that you cannot control the client character set by starting `mysqld` with `--character-set-server=utf8`. However, some Asian customers prefer the MySQL

4.0 behavior. To make it possible to retain this behavior, we added a `mysqld` switch, `--character-set-client-handshake`, which can be turned off with `--skip-character-set-client-handshake`. If you start `mysqld` with `--skip-character-set-client-handshake`, then, when a client connects, it sends to the server the name of the character set that it wants to use. However, *the server ignores this request from the client*.

By way of example, suppose that your favorite server character set is `latin1`. Suppose further that the client uses `utf8` because this is what the client's operating system supports. Start the server with `latin1` as its default character set:

```
mysqld --character-set-server=latin1
```

And then start the client with the default character set `utf8`:

```
mysql --default-character-set=utf8
```

The resulting settings can be seen by viewing the output of `SHOW VARIABLES`:

```
mysql> SHOW VARIABLES LIKE 'char%';
+--------------------------+----------------------------------------+
| Variable_name            | Value                                  |
+--------------------------+----------------------------------------+
| character_set_client     | utf8                                   |
| character_set_connection | utf8                                   |
| character_set_database   | latin1                                 |
| character_set_filesystem | binary                                 |
| character_set_results    | utf8                                   |
| character_set_server     | latin1                                 |
| character_set_system     | utf8                                   |
| character_sets_dir       | /usr/local/mysql/share/mysql/charsets/ |
+--------------------------+----------------------------------------+
```

Now stop the client, and stop the server using `mysqladmin`. Then start the server again, but this time tell it to skip the handshake like so:

```
mysqld --character-set-server=utf8 --skip-character-set-client-handshake
```

Start the client with `utf8` once again as the default character set, then display the resulting settings:

```
mysql> SHOW VARIABLES LIKE 'char%';
+--------------------------+----------------------------------------+
| Variable_name            | Value                                  |
+--------------------------+----------------------------------------+
| character_set_client     | latin1                                 |
| character_set_connection | latin1                                 |
| character_set_database   | latin1                                 |
| character_set_filesystem | binary                                 |
| character_set_results    | latin1                                 |
| character_set_server     | latin1                                 |
| character_set_system     | utf8                                   |
| character_sets_dir       | /usr/local/mysql/share/mysql/charsets/ |
+--------------------------+----------------------------------------+
```

As you can see by comparing the differing results from `SHOW VARIABLES`, the server ignores the client's initial settings if the `--skip-character-set-client-handshake` option is used.

A.11.11 Why do some `LIKE` and `FULLTEXT` searches with CJK characters fail?

For `LIKE` searches, there is a very simple problem with binary string column types such as `BINARY` and `BLOB`: we must know where characters end. With multibyte character sets, different characters might have different octet lengths. For example, in `utf8`, `A` requires one byte but ペ requires three bytes, as shown here:

```
+-------------------------+--------------------------+
| OCTET_LENGTH(_utf8 'A') | OCTET_LENGTH(_utf8 'ペ') |
+-------------------------+--------------------------+
```

```
|                      1 |                         3 |
+------------------------+---------------------------+
```

If we do not know where the first character in a string ends, we do not know where the second character begins, in which case even very simple searches such as `LIKE '_A%'` fail. The solution is to use a nonbinary string column type defined to have the proper CJK character set. For example: `mycol TEXT CHARACTER SET sjis`. Alternatively, convert to a CJK character set before comparing.

This is one reason why MySQL cannot permit encodings of nonexistent characters. If it is not strict about rejecting bad input, it has no way of knowing where characters end.

For `FULLTEXT` searches, we must know where words begin and end. With Western languages, this is rarely a problem because most (if not all) of these use an easy-to-identify word boundary: the space character. However, this is not usually the case with Asian writing. We could use arbitrary halfway measures, like assuming that all Han characters represent words, or (for Japanese) depending on changes from Katakana to Hiragana due to grammatical endings. However, the only sure solution requires a comprehensive word list, which means that we would have to include a dictionary in the server for each Asian language supported. This is simply not feasible.

A.11.12. How do I know whether character *X* is available in all character sets?

The majority of simplified Chinese and basic nonhalfwidth Japanese Kana characters appear in all CJK character sets. The following stored procedure accepts a `UCS-2` Unicode character, converts it to other character sets, and displays the results in hexadecimal.

```
DELIMITER //

CREATE PROCEDURE p_convert(ucs2_char CHAR(1) CHARACTER SET ucs2)
BEGIN

CREATE TABLE tj
             (ucs2 CHAR(1) character set ucs2,
              utf8 CHAR(1) character set utf8,
              big5 CHAR(1) character set big5,
              cp932 CHAR(1) character set cp932,
              eucjpms CHAR(1) character set eucjpms,
              euckr CHAR(1) character set euckr,
              gb2312 CHAR(1) character set gb2312,
              gbk CHAR(1) character set gbk,
              sjis CHAR(1) character set sjis,
              ujis CHAR(1) character set ujis);

INSERT INTO tj (ucs2) VALUES (ucs2_char);

UPDATE tj SET utf8=ucs2,
              big5=ucs2,
              cp932=ucs2,
              eucjpms=ucs2,
              euckr=ucs2,
              gb2312=ucs2,
              gbk=ucs2,
              sjis=ucs2,
              ujis=ucs2;

/* If there are conversion problems, UPDATE produces warnings. */

SELECT hex(ucs2) AS ucs2,
       hex(utf8) AS utf8,
       hex(big5) AS big5,
       hex(cp932) AS cp932,
       hex(eucjpms) AS eucjpms,
       hex(euckr) AS euckr,
       hex(gb2312) AS gb2312,
       hex(gbk) AS gbk,
       hex(sjis) AS sjis,
       hex(ujis) AS ujis
```

```
FROM tj;

DROP TABLE tj;

END//

DELIMITER ;
```

The input can be any single `ucs2` character, or it can be the code value (hexadecimal representation) of that character. For example, from Unicode's list of `ucs2` encodings and names (http://www.unicode.org/Public/UNIDATA/UnicodeData.txt), we know that the Katakana character *Pe* appears in all CJK character sets, and that its code value is `X'30DA'`. If we use this value as the argument to `p_convert()`, the result is as shown here:

```
mysql> CALL p_convert(X'30DA');
+------+--------+------+-------+---------+-------+--------+------+------+------+
| ucs2 | utf8   | big5 | cp932 | eucjpms | euckr | gb2312 | gbk  | sjis | ujis |
+------+--------+------+-------+---------+-------+--------+------+------+------+
| 30DA | E3839A | C772 | 8379  | A5DA    | ABDA  | A5DA   | A5DA | 8379 | A5DA |
+------+--------+------+-------+---------+-------+--------+------+------+------+
```

Since none of the column values is `3F` (that is, the question mark character, `?`), we know that every conversion worked.

A.11.13. Why do CJK strings sort incorrectly in Unicode? (I)

CJK sorting problems that occurred in older MySQL versions can be solved as of MySQL 8.0 by using the `utf8mb4` character set and the `utf8mb4_ja_0900_as_cs` collation.

A.11.14. Why do CJK strings sort incorrectly in Unicode? (II)

CJK sorting problems that occurred in older MySQL versions can be solved as of MySQL 8.0 by using the `utf8mb4` character set and the `utf8mb4_ja_0900_as_cs` collation.

A.11.15. Why are my supplementary characters rejected by MySQL?

Supplementary characters lie outside the Unicode *Basic Multilingual Plane / Plane 0*. BMP characters have code point values between `U+0000` and `U+FFFF`. Supplementary characters have code point values between `U+10000` and `U+10FFFF`.

To store supplementary characters, you must use a character set that permits them:

- The `utf8` and `ucs2` character sets support BMP characters only.

  The `utf8` character set permits only `UTF-8` characters that take up to three bytes. This has led to reports such as that found in Bug #12600, which we rejected as “not a bug”. With `utf8`, MySQL must truncate an input string when it encounters bytes that it does no understand. Otherwise, it is unknown how long the bad multibyte character is.

  One possible workaround is to use `ucs2` instead of `utf8`, in which case the “bad” characters are changed to question marks. However, no truncation takes place. You can also change the data type to `BLOB` or `BINARY`, which perform no validity checking.

- The `utf8mb4`, `utf16`, `utf16le`, and `utf32` character sets support BMP characters, as well as supplementary characters outside the BMP.

A.11.16. Should “CJK” be “CJKV”?

No. The term “CJKV” (*Chinese Japanese Korean Vietnamese*) refers to Vietnamese character sets which contain Han (originally Chinese) characters. MySQL supports the modern Vietnamese script with Western characters, but does not support the old Vietnamese script using Han characters.

As of MySQL 5.6, there are Vietnamese collations for Unicode character sets, as described in Section 10.10.1, “Unicode Character Sets”.

A.11.17 Does MySQL permit CJK characters to be used in database and table names?

Yes.

A.11.18 Where can I find translations of the MySQL Manual into Chinese, Japanese, and Korean?

The Japanese translation of the MySQL 5.6 manual can be downloaded from https://dev.mysql.com/doc/.

A.11.19 Where can I get help with CJK and related issues in MySQL?

The following resources are available:

- A listing of MySQL user groups can be found at https://wikis.oracle.com/display/mysql/List+of+MySQL+User+Groups.
- View feature requests relating to character set issues at http://tinyurl.com/y6xcuf.
- Visit the MySQL Character Sets, Collation, Unicode Forum. http://forums.mysql.com/ also provides foreign-language forums.

A.12 MySQL 8.0 FAQ: Connectors & APIs

For common questions, issues, and answers relating to the MySQL Connectors and other APIs, see the following areas of the Manual:

- Using C API Features
- Connector/ODBC Notes and Tips
- Connector/NET Programming
- MySQL Connector/J 8.0 Developer Guide

A.13 MySQL 8.0 FAQ: C API, libmysql

Frequently asked questions about MySQL C API and libmysql.

A.13.1 What is “MySQL Native C API”? What are typical benefits and use cases?

libmysql is a C-based API that you can use in C applications to connect with the MySQL database server. It is also itself used as the foundation for drivers for standard database APIs like ODBC, Perl's DBI, and Python's DB API.

A.13.2 Which version of libmysql should I use?

For MySQL 8.0, 5.7, 5.6, and 5.5, we recommend libmysql 8.0.

A.13.3 What if I want to use the “NoSQL” X DevAPI?

For C-language and X DevApi Document Store for MySQL 8.0, we recommend MySQL Connector/C++. Connector/C++ 8.0 has compatible C headers. (This is not applicable to MySQL 5.7 or before.)

A.13.4 How to I download libmysql?

- Linux: The Client Utilities Package is available from the MySQL Community Server download page.
- Repos: The Client Utilities Package is available from the Yum, APT, SuSE repositories.
- Windows: The Client Utilities Package is available from Windows Installer.

A.13.5 Where is the documentation?

See MySQL 8.0 C API Developer Guide.

A.13.6 How do I report bugs?

Please report any bugs or inconsistencies you observe to our Bugs Database. Select the C API Client as shown.

A.13.7 Is it possible to compile the library myself?

Compiling MySQL Server also compiles libmysqlclient; there is not a way to only compile libmysqlclient. For related information, see MySQL C API Implementations.

A.14 MySQL 8.0 FAQ: Replication

In the following section, we provide answers to questions that are most frequently asked about MySQL Replication.

A.14.1 Must the replica be connected to the source all the time?

No, it does not. The replica can go down or stay disconnected for hours or even days, and then reconnect and catch up on updates. For example, you can set up a source/replica relationship over a dial-up link where the link is up only sporadically and for short periods of time. The implication of this is that, at any given time, the replica is not guaranteed to be in synchrony with the source unless you take some special measures.

To ensure that catchup can occur for a replica that has been disconnected, you must not remove binary log files from the source that contain information that has not yet been replicated

to the replicas. Asynchronous replication can work only if the replica is able to continue reading the binary log from the point where it last read events.

A.14.2 Must I enable networking on my source and replica to enable replication?

Yes, networking must be enabled on the source and replica. If networking is not enabled, the replica cannot connect to the source and transfer the binary log. Verify that the `skip_networking` system variable has not been enabled in the configuration file for either server.

A.14.3 How do I know how late a replica is compared to the source? In other words, how do I know the date of the last statement replicated by the replica?

Check the `Seconds_Behind_Master` column in the output from `SHOW REPLICA | SLAVE STATUS`. See Section 17.1.7.1, "Checking Replication Status".

When the replication SQL thread executes an event read from the source, it modifies its own time to the event timestamp. (This is why `TIMESTAMP` is well replicated.) In the `Time` column in the output of `SHOW PROCESSLIST`, the number of seconds displayed for the replication SQL thread is the number of seconds between the timestamp of the last replicated event and the real time of the replica machine. You can use this to determine the date of the last replicated event. Note that if your replica has been disconnected from the source for one hour, and then reconnects, you may immediately see large `Time` values such as 3600 for the replication SQL thread in `SHOW PROCESSLIST`. This is because the replica is executing statements that are one hour old. See Section 17.2.3, "Replication Threads".

A.14.4 How do I force the source to block updates until the replica catches up?

Use the following procedure:

1. On the source, execute these statements:

   ```
   mysql> FLUSH TABLES WITH READ LOCK;
   mysql> SHOW MASTER STATUS;
   ```

   Record the replication coordinates (the current binary log file name and position) from the output of the `SHOW` statement.

2. On the replica, issue the following statement, where the arguments to the `SOURCE_POS_WAIT()` or `MASTER_POS_WAIT()` function are the replication coordinate values obtained in the previous step:

   ```
   mysql> SELECT MASTER_POS_WAIT('log_name', log_pos);

   Or from MySQL 8.0.26:
   mysql> SELECT SOURCE_POS_WAIT('log_name', log_pos);
   ```

   The `SELECT` statement blocks until the replica reaches the specified log file and position. At that point, the replica is in synchrony with the source and the statement returns.

3. On the source, issue the following statement to enable the source to begin processing updates again:

   ```
   mysql> UNLOCK TABLES;
   ```

A.14.5 What issues should I be aware of when setting up two-way replication?

MySQL replication currently does not support any locking protocol between source and replica to guarantee the atomicity of a distributed (cross-server) update. In other words, it is possible for client A to make an update to co-source 1, and in the meantime, before it propagates to co-source 2, client B could make an update to co-source 2 that makes the update of client A work differently than it did on co-source 1. Thus, when the update of client A makes it to co-source 2, it produces tables that are different from what you have on co-source 1, even after all the

updates from co-source 2 have also propagated. This means that you should not chain two servers together in a two-way replication relationship unless you are sure that your updates can safely happen in any order, or unless you take care of mis-ordered updates somehow in the client code.

You should also realize that two-way replication actually does not improve performance very much (if at all) as far as updates are concerned. Each server must do the same number of updates, just as you would have a single server do. The only difference is that there is a little less lock contention because the updates originating on another server are serialized in one replication thread. Even this benefit might be offset by network delays.

A.14.6. How can I use replication to improve performance of my system?

Set up one server as the source and direct all writes to it. Then configure as many replicas as you have the budget and rackspace for, and distribute the reads among the source and the replicas. You can also start the replicas with the `--skip-innodb` option, enable the `low_priority_updates` system variable, and set the `delay_key_write` system variable to `ALL` to get speed improvements on the replica end. In this case, the replica uses nontransactional `MyISAM` tables instead of `InnoDB` tables to get more speed by eliminating transactional overhead.

A.14.7. What should I do to prepare client code in my own applications to use performance-enhancing replication?

See the guide to using replication as a scale-out solution, Section 17.4.5, “Using Replication for Scale-Out”.

A.14.8. When and how much can MySQL replication improve the performance of my system?

MySQL replication is most beneficial for a system that processes frequent reads and infrequent writes. In theory, by using a single-source/multiple-replica setup, you can scale the system by adding more replicas until you either run out of network bandwidth, or your update load grows to the point that the source cannot handle it.

To determine how many replicas you can use before the added benefits begin to level out, and how much you can improve performance of your site, you must know your query patterns, and determine empirically by benchmarking the relationship between the throughput for reads and writes on a typical source and a typical replica. The example here shows a rather simplified calculation of what you can get with replication for a hypothetical system. Let `reads` and `writes` denote the number of reads and writes per second, respectively.

Let's say that system load consists of 10% writes and 90% reads, and we have determined by benchmarking that `reads` is 1200 - 2 * `writes`. In other words, the system can do 1,200 reads per second with no writes, the average write is twice as slow as the average read, and the relationship is linear. Suppose that the source and each replica have the same capacity, and that we have one source and *N* replicas. Then we have for each server (source or replica):

`reads` = 1200 - 2 * `writes`

`reads` = 9 * `writes` / (*N* + 1) (reads are split, but writes replicated to all replicas)

9 * `writes` / (*N* + 1) + 2 * `writes` = 1200

`writes` = 1200 / (2 + 9/(*N* + 1))

The last equation indicates the maximum number of writes for *N* replicas, given a maximum possible read rate of 1,200 per second and a ratio of nine reads per write.

This analysis yields the following conclusions:

- If *N* = 0 (which means we have no replication), our system can handle about 1200/11 = 109 writes per second.
- If *N* = 1, we get up to 184 writes per second.
- If *N* = 8, we get up to 400 writes per second.
- If *N* = 17, we get up to 480 writes per second.
- Eventually, as *N* approaches infinity (and our budget negative infinity), we can get very close to 600 writes per second, increasing system throughput about 5.5 times. However, with only eight servers, we increase it nearly four times.

These computations assume infinite network bandwidth and neglect several other factors that could be significant on your system. In many cases, you may not be able to perform a computation similar to the one just shown that accurately predicts what happens on your system if you add *N* replicas. However, answering the following questions should help you decide whether and by how much replication may improve the performance of your system:

- What is the read/write ratio on your system?
- How much more write load can one server handle if you reduce the reads?
- For how many replicas do you have bandwidth available on your network?

A.14.9. How can I use replication to provide redundancy or high availability?

How you implement redundancy is entirely dependent on your application and circumstances. High-availability solutions (with automatic failover) require active monitoring and either custom scripts or third party tools to provide the failover support from the original MySQL server to the replica.

To handle the process manually, you should be able to switch from a failed source to a pre-configured replica by altering your application to talk to the new server or by adjusting the DNS for the MySQL server from the failed server to the new server.

For more information and some example solutions, see Section 17.4.8, “Switching Sources During Failover”.

A.14.10. How do I tell whether a replication source server is using statement-based or row-based binary logging format?

Check the value of the `binlog_format` system variable:

```
mysql> SHOW VARIABLES LIKE 'binlog_format';
```

The value shown is always one of `STATEMENT`, `ROW`, or `MIXED`. For `MIXED` mode, statement-based logging is used by default but replication switches automatically to row-based logging under certain conditions, such as unsafe statements. For information about when this may occur, see Section 5.4.4.3, “Mixed Binary Logging Format”.

A.14.11. How do I tell a replica to use row-based replication?

Replicas automatically know which format to use.

A.14.12. How do I prevent `GRANT` and `REVOKE` statements from replicating to replica machines?

Start the server with the `--replicate-wild-ignore-table=mysql.%` option to ignore replication for tables in the `mysql` database.

A.14.13. Does replication work on mixed operating systems (for example, the source runs on Linux while replicas run on macOS and Windows)?

Yes.

A.14.14 Does replication work on mixed hardware architectures (for example, the source runs on a 64-bit machine while replicas run on 32-bit machines)?

Yes.

A.15 MySQL 8.0 FAQ: MySQL Enterprise Thread Pool

A.15.1 What is the Thread Pool and what problem does it solve?

The MySQL Thread Pool is a MySQL server plugin that extends the default connection-handling capabilities of the MySQL server to limit the number of concurrently executing statements/queries and transactions to ensure that each has sufficient CPU and memory resources to fulfill its task. For MySQL 8.0, the Thread Pool plugin is included in MySQL Enterprise Edition, a commercial product.

The default thread-handling model in MySQL Server executes statements using one thread per client connection. As more clients connect to the server and execute statements, overall performance degrades. The Thread Pool plugin provides an alternative thread-handling model designed to reduce overhead and improve performance. The Thread Pool plugin increases server performance by efficiently managing statement execution threads for large numbers of client connections, especially on modern multi-CPU/Core systems.

For more information, see Section 5.6.3, “MySQL Enterprise Thread Pool”.

A.15.2 How does the Thread Pool limit and manage concurrent sessions and transactions for optimal performance and throughput?

The Thread Pool uses a “divide and conquer” approach to limiting and balancing concurrency. Unlike the default connection handling of the MySQL Server, the Thread Pool separates connections and threads, so there is no fixed relationship between connections and the threads that execute statements received from those connections. The Thread Pool then manages client connections within configurable thread groups, where they are prioritized and queued based on the nature of the work they were submitted to accomplish.

For more information, see Section 5.6.3.3, “Thread Pool Operation”.

A.15.3 How is the Thread Pool different from the client side Connection Pool?

The MySQL Connection Pool operates on the client side to ensure that a MySQL client does not constantly connect to and disconnect from the MySQL server. It is designed to cache idle connections in the MySQL client for use by other users as they are needed. This minimizes the overhead and expense of establishing and tearing down connections as queries are submitted to the MySQL server. The MySQL Connection Pool has no visibility as to the query handling capabilities or load of the back-end MySQL server. By contrast, the Thread Pool operates on the MySQL server side and is designed to manage the execution of inbound concurrent connections and queries as they are received from the client connections accessing the back-end MySQL database. Because of the separation of duties, the MySQL Connection Pool and Thread Pool are orthogonal and can be used independent of each other.

MySQL Connection Pooling via the MySQL Connectors is covered in Chapter 29, *Connectors and APIs*.

A.15.4 When should I use the Thread Pool?

There are a few rules of thumb to consider for optimal Thread Pool use cases:

The MySQL `Threads_running` variable keeps track of the number of concurrent statements currently executing in the MySQL Server. If this variable consistently exceeds a region where the server won't operate optimally (usually going beyond 40 for InnoDB workloads), the Thread Pool should be beneficial, especially in extreme parallel overload situations.

If you are using the `innodb_thread_concurrency` to limit the number of concurrently executing statements, you should find that the Thread Pool solves the same problem, only better, by assigning connections to thread groups, then queuing executions based on transactional content, user defined designations, and so forth.

Lastly, if your workload comprises mainly short queries, the Thread Pool should be beneficial.

To learn more, see Section 5.6.3.4, “Thread Pool Tuning”.

A.15.5 Are there recommended Thread Pool configurations?

The Thread Pool has a number of user case driven configuration parameters that affect its performance. To learn about these and tips on tuning, see Section 5.6.3.4, “Thread Pool Tuning”.

A.16 MySQL 8.0 FAQ: InnoDB Change Buffer

A.16.1 What types of operations modify secondary indexes and result in change buffering?

`INSERT`, `UPDATE`, and `DELETE` operations can modify secondary indexes. If an affected index page is not in the buffer pool, the changes can be buffered in the change buffer.

A.16.2 What is the benefit of the `InnoDB` change buffer?

Buffering secondary index changes when secondary index pages are not in the buffer pool avoids expensive random access I/O operations that would be required to immediately read in affected index pages from disk. Buffered changes can be applied later, in batches, as pages are read into the buffer pool by other read operations.

A.16.3 Does the change buffer support other types of indexes?

No. The change buffer only supports secondary indexes. Clustered indexes, full-text indexes, and spatial indexes are not supported. Full-text indexes have their own caching mechanism.

A.16.4 How much space does `InnoDB` use for the change buffer?

Prior to the introduction of the `innodb_change_buffer_max_size` configuration option in MySQL 5.6, the maximum size of the on-disk change buffer in the system tablespace was 1/3 of the `InnoDB` buffer pool size.

In MySQL 5.6 and later, the `innodb_change_buffer_max_size` configuration option defines the maximum size of the change buffer as a percentage of the total buffer pool size. By default, `innodb_change_buffer_max_size` is set to 25. The maximum setting is 50.

`InnoDB` does not buffer an operation if it would cause the on-disk change buffer to exceed the defined limit.

Change buffer pages are not required to persist in the buffer pool and may be evicted by LRU operations.

A.16.5 How do I determine the current size of the change buffer?

The current size of the change buffer is reported by `SHOW ENGINE INNODB STATUS \G`, under the `INSERT BUFFER AND ADAPTIVE HASH INDEX` heading. For example:

```
-------------------------------------
INSERT BUFFER AND ADAPTIVE HASH INDEX
-------------------------------------
Ibuf: size 1, free list len 0, seg size 2, 0 merges
```

Relevant data points include:

- `size`: The number of pages used within the change buffer. Change buffer size is equal to `seg size - (1 + free list len)`. The `1 +` value represents the change buffer header page.
- `seg size`: The size of the change buffer, in pages.

For information about monitoring change buffer status, see Section 15.5.2, “Change Buffer”.

A.16.6 When does change buffer merging occur?

- When a page is read into the buffer pool, buffered changes are merged upon completion of the read, before the page is made available.
- Change buffer merging is performed as a background task. The `innodb_io_capacity` parameter sets an upper limit on the I/O activity performed by `InnoDB` background tasks such as merging data from the change buffer.
- A change buffer merge is performed during crash recovery. Changes are applied from the change buffer (in the system tablespace) to leaf pages of secondary indexes as index pages are read into the buffer pool.
- The change buffer is fully durable and can survive a system crash. Upon restart, change buffer merge operations resume as part of normal operations.
- A full merge of the change buffer can be forced as part of a slow server shutdown using `--innodb-fast-shutdown=0`.

A.16.7 When is the change buffer flushed?

Updated pages are flushed by the same flushing mechanism that flushes the other pages that occupy the buffer pool.

A.16.8 When should the change buffer be used?

The change buffer is a feature designed to reduce random I/O to secondary indexes as indexes grow larger and no longer fit in the `InnoDB` buffer pool. Generally, the change buffer should be used when the entire data set does not fit into the buffer pool, when there is substantial DML activity that modifies secondary index pages, or when there are lots of secondary indexes that are regularly changed by DML activity.

A.16.9 When should the change buffer not be used?

You might consider disabling the change buffer if the entire data set fits within the `InnoDB` buffer pool, if you have relatively few secondary indexes, or if you are using solid-state storage, where

random reads are about as fast as sequential reads. Before making configuration changes, it is recommended that you run tests using a representative workload to determine if disabling the change buffer provides any benefit.

A.16.10 Where can I find additional information about the change buffer?

See Section 15.5.2, “Change Buffer”.

A.17 MySQL 8.0 FAQ: InnoDB Data-at-Rest Encryption

A.17.1 Is data decrypted for users who are authorized to see it?

Yes. InnoDB data-at-rest encryption is designed to transparently apply encryption within the database without impacting existing applications. Returning data in encrypted format would break most existing applications. InnoDB data-at-rest encryption provides the benefit of encryption without the overhead associated with traditional database encryption solutions, which would typically require expensive and substantial changes to applications, database triggers, and views.

A.17.2 What is the overhead associated with InnoDB data-at-rest encryption?

There is no additional storage overhead. According to internal benchmarks, performance overhead amounts to a single digit percentage difference.

A.17.3 What are the encryption algorithms used with InnoDB data-at-rest encryption?

InnoDB data-at-rest encryption supports the Advanced Encryption Standard (AES256) block-based encryption algorithm. It uses Electronic Codebook (ECB) block encryption mode for tablespace key encryption and Cipher Block Chaining (CBC) block encryption mode for data encryption.

A.17.4 Is it possible to use 3rd party encryption algorithms in place of the one provided by the InnoDB data-at-rest encryption feature?

No, it is not possible to use other encryption algorithms. The provided encryption algorithm is broadly accepted.

A.17.5 Can indexed columns be encrypted?

InnoDB data-at-rest encryption supports all indexes transparently.

A.17.6 What data types and data lengths does InnoDB data-at-rest encryption support?

`InnoDB` data-at-rest encryption supports all supported data types. There is no data length limitation.

A.17.7 Does data remain encrypted on the network?

Data encrypted by the `InnoDB` data-at-rest feature is decrypted when it is read from the tablespace file. Thus, if the data is on the network, it is in cleartext form. However, data on the network can be encrypted using MySQL network encryption, which encrypts data traveling to and from a database using SSL/TLS.

A.17.8 Does database memory contain cleartext or encrypted data?

With `InnoDB` data-at-rest encryption, in-memory data is decrypted, which provides complete transparency.

A.17.9 How do I know which data to encrypt?

Compliance with the PCI-DSS standard requires that credit card numbers (Primary Account Number, or 'PAN') be stored in encrypted form. Breach Notification Laws (for example, CA SB 1386, CA AB 1950, and similar laws in 43+ more US states) require encryption of first name, last name, driver license number, and other PII data. In early 2008, CA AB 1298 added medical and health insurance information to PII data. Additionally, industry specific privacy and security standards may require encryption of certain assets. For example, assets such as pharmaceutical research results, oil field exploration results, financial contracts, or personal data of law enforcement informants may require encryption. In the health care industry, the privacy of patient data, health records and X-ray images is of the highest importance.

A.17.10 How is `InnoDB` data-at-rest encryption different from encryption functions MySQL already provides?

There are symmetric and asymmetric encryption APIs in MySQL that can be used to manually encrypt data within the database. However, the application must manage encryption keys and perform required encryption and decryption operations by calling API functions. `InnoDB` data-at-rest encryption requires no application changes, is transparent to end users, and provides automated, built-in key management.

A.17.11 Does the transportable tablespaces feature work with `InnoDB` data-at-rest encryption?

Yes. It is supported for encrypted file-per-table tablespaces. For more information, see Exporting Encrypted Tablespaces.

A.17.12 Does compression work with `InnoDB` data-at-rest encryption?

Customers using `InnoDB` data-at-rest encryption receive the full benefit of compression because compression is applied before data blocks are encrypted.

A.17.13 Can I use `mysqlpump` or `mysqldump` with encrypted tables?

Yes. Because these utilities create logical backups, the data dumped from encrypted tables is not encrypted.

A.17.14 How do I change (rotate, re-key) the master encryption key?

`InnoDB` data-at-rest encryption uses a two tier key mechanism. When data-at-rest encryption is used, individual tablespace keys are stored in the header of the underlying tablespace data file. Tablespace keys are encrypted using the master encryption key. The master encryption key is generated when tablespace encryption is enabled, and is stored outside the database. The master encryption key is rotated using the `ALTER INSTANCE ROTATE INNODB MASTER KEY` statement, which generates a new master encryption key, stores the key, and rotates the key into use.

A.17.15 How do I migrate data from a cleartext `InnoDB` tablespace to an encrypted `InnoDB` tablespace?

Transferring data from one tablespace to another is not required. To encrypt data in an `InnoDB` file-per-table tablespace, run `ALTER TABLE tbl_name ENCRYPTION = 'Y'`. To encrypt a general tablespace or the `mysql` tablespace, run `ALTER TABLESPACE tablespace_name ENCRYPTION = 'Y'`. Encryption support for general tablespaces was introduced in MySQL 8.0.13. Encryption support for the `mysql` system tablespace is available as of MySQL 8.0.16.

A.18 MySQL 8.0 FAQ: Virtualization Support

A.18.1 Is MySQL supported on virtualized environments such as Oracle VM, VMWare, Docker, Microsoft Hyper-V, or others?

MySQL is supported on virtualized environments, but is certified only for Oracle VM. Contact Oracle Support for more information.

Be aware of potential problems when using virtualization software. The usual ones are related to performance, performance degradations, slowness, or unpredictability of disk, I/O, network, and memory.

Appendix B Error Messages and Common Problems

Table of Contents

This appendix describes the types of error information MySQL provides and how to obtain information about them. The final section is for troubleshooting. It describes common problems and errors that may occur and potential resolutions.

Additional Resources

Other error-related documentation includes:

- Information about configuring where and how the server writes the error log: Section 5.4.2, “The Error Log”
- Information about the character set used for error messages: Section 10.6, “Error Message Character Set”
- Information about the language used for error messages: Section 10.12, “Setting the Error Message Language”
- Information about errors related to `InnoDB`: Section 15.21.5, “InnoDB Error Handling”
- Information about errors specific to NDB Cluster: NDB Cluster API Errors; see also NDB API Errors and Error Handling, and MGM API Errors
- Descriptions of the error messages that the MySQL server and client programs generate: MySQL 8.0 Error Message Reference

B.1 Error Message Sources and Elements

This section discusses how error messages originate within MySQL and the elements they contain.

- Error Message Sources
- Error Message Elements
- Error Code Ranges

Error Message Sources

Error messages can originate on the server side or the client side:

- On the server side, error messages may occur during the startup and shutdown processes, as a result of issues that occur during SQL statement execution, and so forth.
  - The MySQL server writes some error messages to its error log. These indicate issues of interest to database administrators or that require DBA action.

  - The server sends other error messages to client programs. These indicate issues pertaining only to a particular client. The MySQL client library takes errors received from the server and makes them available to the host client program.

- Client-side error messages are generated from within the MySQL client library, usually involving problems communicating with the server.

Example server-side error messages written to the error log:

- This message produced during the startup process provides a status or progress indicator:

```
2018-10-28T13:01:32.735983Z 0 [Note] [MY-010303] [Server] Skipping
generation of SSL certificates as options related to SSL are specified.
```

- This message indicates an issue that requires DBA action:

```
2018-10-02T03:20:39.410387Z 768 [ERROR] [MY-010045] [Server] Event Scheduler:
[evtuser@localhost][myschema.e_daily] Unknown database 'mydb'
```

Example server-side error message sent to client programs, as displayed by the `mysql` client:

```
mysql> SELECT * FROM no_such_table;
ERROR 1146 (42S02): Table 'test.no_such_table' doesn't exist
```

Example client-side error message originating from within the client library, as displayed by the `mysql` client:

```
$> mysql -h no-such-host
ERROR 2005 (HY000): Unknown MySQL server host 'no-such-host' (-2)
```

Whether an error originates from within the client library or is received from the server, a MySQL client program may respond in varying ways. As just illustrated, the client may display the error message so the user can take corrective measures. The client may instead internally attempt to resolve or retry a failed operation, or take other action.

Error Message Elements

When an error occurs, error information includes several elements: an error code, SQLSTATE value, and message string. These elements have the following characteristics:

- Error code: This value is numeric. It is MySQL-specific and is not portable to other database systems.

  Each error number has a corresponding symbolic value. Examples:

  - The symbol for server error number `1146` is `ER_NO_SUCH_TABLE`.
  - The symbol for client error number `2005` is `CR_UNKNOWN_HOST`.

  The set of error codes used in error messages is partitioned into distinct ranges; see Error Code Ranges.

  Error codes are stable across General Availability (GA) releases of a given MySQL series. Before a series reaches GA status, new codes may still be under development and are subject to change.

- SQLSTATE value: This value is a five-character string (for example, `'42S02'`). SQLSTATE values are taken from ANSI SQL and ODBC and are more standardized than the numeric error codes. The first two characters of an SQLSTATE value indicate the error class:

  - Class = `'00'` indicates success.
  - Class = `'01'` indicates a warning.

- Class = `'02'` indicates “not found.” This is relevant within the context of cursors and is used to control what happens when a cursor reaches the end of a data set. This condition also occurs for `SELECT ... INTO var_list` statements that retrieve no rows.
- Class > `'02'` indicates an exception.

For server-side errors, not all MySQL error numbers have corresponding SQLSTATE values. In these cases, `'HY000'` (general error) is used.

For client-side errors, the SQLSTATE value is always `'HY000'` (general error), so it is not meaningful for distinguishing one client error from another.

- Message string: This string provides a textual description of the error.

Error Code Ranges

The set of error codes used in error messages is partitioned into distinct ranges, each with its own purpose:

- 1 to 999: Global error codes. This error code range is called “global” because it is a shared range that is used by the server as well as by clients.

  When an error in this range originates on the server side, the server writes it to the error log, padding the error code with leading zeros to six digits and adding a prefix of `MY-`.

  When an error in this range originates on the client side, the client library makes it available to the client program with no zero-padding or prefix.
- 1,000 to 1,999: Server error codes reserved for messages sent to clients.
- 2,000 to 2,999: Client error codes reserved for use by the client library.
- 3,000 to 4,999: Server error codes reserved for messages sent to clients.
- 5,000 to 5,999: Error codes reserved for use by X Plugin for messages sent to clients.
- 10,000 to 49,999: Server error codes reserved for messages to be written to the error log (not sent to clients).

  When an error in this range occurs, the server writes it to the error log, padding the error code with leading zeros to six digits and adding a prefix of `MY-`.
- 50,000 to 51,999: Error codes reserved for use by third parties.

The server handles error messages written to the error log differently from error messages sent to clients:

- When the server writes a message to the error log, it pads the error code with leading zeros to six digits and adds a prefix of `MY-` (examples: `MY-000022`, `MY-010048`).
- When the server sends a message to a client program, it adds no zero-padding or prefix to the error code (examples: `1036`, `3013`).

B.2 Error Information Interfaces

Error messages can originate on the server side or the client side, and each error message includes an error code, SQLSTATE value, and message string, as described in Section B.1, “Error Message Sources and Elements”. For lists of server-side, client-side, and global (shared between server and clients) errors, see MySQL 8.0 Error Message Reference.

For error checking from within programs, use error code numbers or symbols, not error message strings. Message strings do not change often, but it is possible. Also, if the database administrator

changes the language setting, that affects the language of message strings; see Section 10.12, “Setting the Error Message Language”.

Error information in MySQL is available in the server error log, at the SQL level, from within client programs, and at the command line.

- Error Log
- SQL Error Message Interface
- Client Error Message Interface
- Command-Line Error Message Interface

Error Log

On the server side, some messages are intended for the error log. For information about configuring where and how the server writes the log, see Section 5.4.2, “The Error Log”.

Other server error messages are intended to be sent to client programs and are available as described in Client Error Message Interface.

The range within which a particular error code lies determines whether the server writes an error message to the error log or sends it to clients. For information about these ranges, see Error Code Ranges.

SQL Error Message Interface

At the SQL level, there are several sources of error information in MySQL:

- SQL statement warning and error information is available through the `SHOW WARNINGS` and `SHOW ERRORS` statements. The `warning_count` system variable indicates the number of errors, warnings, and notes (with notes excluded if the `sql_notes` system variable is disabled). The `error_count` system variable indicates the number of errors. Its value excludes warnings and notes.
- The `GET DIAGNOSTICS` statement may be used to inspect the diagnostic information in the diagnostics area. See Section 13.6.7.3, “GET DIAGNOSTICS Statement”.
- `SHOW SLAVE STATUS` statement output includes information about replication errors occurring on replica servers.
- `SHOW ENGINE INNODB STATUS` statement output includes information about the most recent foreign key error if a `CREATE TABLE` statement for an `InnoDB` table fails.

Client Error Message Interface

Client programs receive errors from two sources:

- Errors that originate on the client side from within the MySQL client library.
- Errors that originate on the server side and are sent to the client by the server. These are received within the client library, which makes them available to the host client program.

The range within which a particular error code lies determines whether it originated from within the client library or was received by the client from the server. For information about these ranges, see Error Code Ranges.

Regardless of whether an error originates from within the client library or is received from the server, a MySQL client program obtains the error code, SQLSTATE value, message string, and other related information by calling C API functions in the client library:

- `mysql_errno()` returns the MySQL error code.
- `mysql_sqlstate()` returns the SQLSTATE value.
- `mysql_error()` returns the message string.
- `mysql_stmt_errno()`, `mysql_stmt_sqlstate()`, and `mysql_stmt_error()` are the corresponding error functions for prepared statements.
- `mysql_warning_count()` returns the number of errors, warnings, and notes for the most recent statement.

For descriptions of the client library error functions, see MySQL 8.0 C API Developer Guide.

A MySQL client program may respond to an error in varying ways. The client may display the error message so the user can take corrective measures, internally attempt to resolve or retry a failed operation, or take other action. For example, (using the `mysql` client), a failure to connect to the server might result in this message:

```
$> mysql -h no-such-host
ERROR 2005 (HY000): Unknown MySQL server host 'no-such-host' (-2)
```

Command-Line Error Message Interface

The `perror` program provides information from the command line about error numbers. See Section 4.8.2, “perror — Display MySQL Error Message Information”.

```
$> perror 1231
MySQL error code MY-001231 (ER_WRONG_VALUE_FOR_VAR): Variable '%-.64s'
can't be set to the value of '%-.200s'
```

For MySQL NDB Cluster errors, use `ndb_perror`. See Section 23.5.16, “ndb_perror — Obtain NDB Error Message Information”.

```
$> ndb_perror 323
NDB error code 323: Invalid nodegroup id, nodegroup already existing:
Permanent error: Application error
```

B.3 Problems and Common Errors

This section lists some common problems and error messages that you may encounter. It describes how to determine the causes of the problems and what to do to solve them.

B.3.1 How to Determine What Is Causing a Problem

When you run into a problem, the first thing you should do is to find out which program or piece of equipment is causing it:

- If you have one of the following symptoms, then it is probably a hardware problems (such as memory, motherboard, CPU, or hard disk) or kernel problem:
  - The keyboard does not work. This can normally be checked by pressing the Caps Lock key. If the Caps Lock light does not change, you have to replace your keyboard. (Before doing this, you should try to restart your computer and check all cables to the keyboard.)
  - The mouse pointer does not move.
  - The machine does not answer to a remote machine's pings.
  - Other programs that are not related to MySQL do not behave correctly.
  - Your system restarted unexpectedly. (A faulty user-level program should never be able to take down your system.)

In this case, you should start by checking all your cables and run some diagnostic tool to check your hardware! You should also check whether there are any patches, updates, or service packs for your operating system that could likely solve your problem. Check also that all your libraries (such as `glibc`) are up to date.

It is always good to use a machine with ECC memory to discover memory problems early.

- If your keyboard is locked up, you may be able to recover by logging in to your machine from another machine and executing `kbd_mode -a`.
- Please examine your system log file (`/var/log/messages` or similar) for reasons for your problem. If you think the problem is in MySQL, you should also examine MySQL's log files. See Section 5.4, “MySQL Server Logs”.
- If you do not think you have hardware problems, you should try to find out which program is causing problems. Try using `top`, `ps`, Task Manager, or some similar program, to check which program is taking all CPU or is locking the machine.
- Use `top`, `df`, or a similar program to check whether you are out of memory, disk space, file descriptors, or some other critical resource.
- If the problem is some runaway process, you can always try to kill it. If it does not want to die, there is probably a bug in the operating system.

If you have examined all other possibilities and concluded that the MySQL server or a MySQL client is causing the problem, it is time to create a bug report, see Section 1.5, “How to Report Bugs or Problems”. In the bug report, try to give a complete description of how the system is behaving and what you think is happening. Also state why you think that MySQL is causing the problem. Take into consideration all the situations described in this chapter. State any problems exactly how they appear when you examine your system. Use the “copy and paste” method for any output and error messages from programs and log files.

Try to describe in detail which program is not working and all symptoms you see. We have in the past received many bug reports that state only “the system does not work.” This provides us with no information about what could be the problem.

If a program fails, it is always useful to know the following information:

- Has the program in question made a segmentation fault (did it dump core)?
- Is the program taking up all available CPU time? Check with `top`. Let the program run for a while, it may simply be evaluating something computationally intensive.
- If the `mysqld` server is causing problems, can you get any response from it with `mysqladmin -u root ping` or `mysqladmin -u root processlist`?
- What does a client program say when you try to connect to the MySQL server? (Try with `mysql`, for example.) Does the client jam? Do you get any output from the program?

When sending a bug report, you should follow the outline described in Section 1.5, “How to Report Bugs or Problems”.

B.3.2 Common Errors When Using MySQL Programs

This section lists some errors that users frequently encounter when running MySQL programs. Although the problems show up when you try to run client programs, the solutions to many of the problems involves changing the configuration of the MySQL server.

B.3.2.1 Access denied

An `Access denied` error can have many causes. Often the problem is related to the MySQL accounts that the server permits client programs to use when connecting. See Section 6.2, “Access Control and Account Management”, and Section 6.2.22, “Troubleshooting Problems Connecting to MySQL”.

B.3.2.2 Can't connect to [local] MySQL server

A MySQL client on Unix can connect to the `mysqld` server in two different ways: By using a Unix socket file to connect through a file in the file system (default `/tmp/mysql.sock`), or by using TCP/IP, which connects through a port number. A Unix socket file connection is faster than TCP/IP, but can be used only when connecting to a server on the same computer. A Unix socket file is used if you do not specify a host name or if you specify the special host name `localhost`.

If the MySQL server is running on Windows, you can connect using TCP/IP. If the server is started with the `named_pipe` system variable enabled, you can also connect with named pipes if you run the client on the host where the server is running. The name of the named pipe is `MySQL` by default. If you do not give a host name when connecting to `mysqld`, a MySQL client first tries to connect to the named pipe. If that does not work, it connects to the TCP/IP port. You can force the use of named pipes on Windows by using `.` as the host name.

The error (2002) `Can't connect to ...` normally means that there is no MySQL server running on the system or that you are using an incorrect Unix socket file name or TCP/IP port number when trying to connect to the server. You should also check that the TCP/IP port you are using has not been blocked by a firewall or port blocking service.

The error (2003) `Can't connect to MySQL server on 'server' (10061)` indicates that the network connection has been refused. You should check that there is a MySQL server running, that it has network connections enabled, and that the network port you specified is the one configured on the server.

Start by checking whether there is a process named `mysqld` running on your server host. (Use `ps xa | grep mysqld` on Unix or the Task Manager on Windows.) If there is no such process, you should start the server. See Section 2.9.2, “Starting the Server”.

If a `mysqld` process is running, you can check it by trying the following commands. The port number or Unix socket file name might be different in your setup. `host_ip` represents the IP address of the machine where the server is running.

```
$> mysqladmin version
$> mysqladmin variables
$> mysqladmin -h `hostname` version variables
$> mysqladmin -h `hostname` --port=3306 version
$> mysqladmin -h host_ip version
$> mysqladmin --protocol=SOCKET --socket=/tmp/mysql.sock version
```

Note the use of backticks rather than forward quotation marks with the `hostname` command; these cause the output of `hostname` (that is, the current host name) to be substituted into the `mysqladmin` command. If you have no `hostname` command or are running on Windows, you can manually type the host name of your machine (without backticks) following the `-h` option. You can also try `-h 127.0.0.1` to connect with TCP/IP to the local host.

Make sure that the server has not been configured to ignore network connections or (if you are attempting to connect remotely) that it has not been configured to listen only locally on its network interfaces. If the server was started with the `skip_networking` system variable enabled, it cannot accept TCP/IP connections at all. If the server was started with the `bind_address` system variable set to `127.0.0.1`, it listens for TCP/IP connections only locally on the loopback interface and does not accept remote connections.

Check to make sure that there is no firewall blocking access to MySQL. Your firewall may be configured on the basis of the application being executed, or the port number used by MySQL for communication (3306 by default). Under Linux or Unix, check your IP tables (or similar) configuration

to ensure that the port has not been blocked. Under Windows, applications such as ZoneAlarm or Windows Firewall may need to be configured not to block the MySQL port.

Here are some reasons the `Can't connect to local MySQL server` error might occur:

- `mysqld` is not running on the local host. Check your operating system's process list to ensure the `mysqld` process is present.
- You're running a MySQL server on Windows with many TCP/IP connections to it. If you're experiencing that quite often your clients get that error, you can find a workaround here: Connection to MySQL Server Failing on Windows.
- Someone has removed the Unix socket file that `mysqld` uses (`/tmp/mysql.sock` by default). For example, you might have a `cron` job that removes old files from the `/tmp` directory. You can always run `mysqladmin version` to check whether the Unix socket file that `mysqladmin` is trying to use really exists. The fix in this case is to change the `cron` job to not remove `mysql.sock` or to place the socket file somewhere else. See Section B.3.3.6, “How to Protect or Change the MySQL Unix Socket File”.
- You have started the `mysqld` server with the `--socket=/path/to/socket` option, but forgotten to tell client programs the new name of the socket file. If you change the socket path name for the server, you must also notify the MySQL clients. You can do this by providing the same `--socket` option when you run client programs. You also need to ensure that clients have permission to access the `mysql.sock` file. To find out where the socket file is, you can do:

  ```
  $> netstat -ln | grep mysql
  ```

  See Section B.3.3.6, “How to Protect or Change the MySQL Unix Socket File”.
- You are using Linux and one server thread has died (dumped core). In this case, you must kill the other `mysqld` threads (for example, with `kill`) before you can restart the MySQL server. See Section B.3.3.3, “What to Do If MySQL Keeps Crashing”.
- The server or client program might not have the proper access privileges for the directory that holds the Unix socket file or the socket file itself. In this case, you must either change the access privileges for the directory or socket file so that the server and clients can access them, or restart `mysqld` with a `--socket` option that specifies a socket file name in a directory where the server can create it and where client programs can access it.

If you get the error message `Can't connect to MySQL server on some_host`, you can try the following things to find out what the problem is:

- Check whether the server is running on that host by executing `telnet some_host 3306` and pressing the Enter key a couple of times. (3306 is the default MySQL port number. Change the value if your server is listening to a different port.) If there is a MySQL server running and listening to the port, you should get a response that includes the server's version number. If you get an error such as `telnet: Unable to connect to remote host: Connection refused`, then there is no server running on the given port.
- If the server is running on the local host, try using `mysqladmin -h localhost variables` to connect using the Unix socket file. Verify the TCP/IP port number that the server is configured to listen to (it is the value of the `port` variable.)
- If you are running under Linux and Security-Enhanced Linux (SELinux) is enabled, see Section 6.7, “SELinux”.

Connection to MySQL Server Failing on Windows

When you're running a MySQL server on Windows with many TCP/IP connections to it, and you're experiencing that quite often your clients get a `Can't connect to MySQL server` error, the reason might be that Windows does not allow for enough ephemeral (short-lived) ports to serve those connections.

The purpose of TIME_WAIT is to keep a connection accepting packets even after the connection has been closed. This is because Internet routing can cause a packet to take a slow route to its destination and it may arrive after both sides have agreed to close. If the port is in use for a new connection, that packet from the old connection could break the protocol or compromise personal information from the original connection. The TIME_WAIT delay prevents this by ensuring that the port cannot be reused until after some time has been permitted for those delayed packets to arrive.

It is safe to reduce TIME_WAIT greatly on LAN connections because there is little chance of packets arriving at very long delays, as they could through the Internet with its comparatively large distances and latencies.

Windows permits ephemeral (short-lived) TCP ports to the user. After any port is closed, it remains in a TIME_WAIT status for 120 seconds. The port is not available again until this time expires. The default range of port numbers depends on the version of Windows, with a more limited number of ports in older versions:

- Windows through Server 2003: Ports in range 1025–5000
- Windows Vista, Server 2008, and newer: Ports in range 49152–65535

With a small stack of available TCP ports (5000) and a high number of TCP ports being open and closed over a short period of time along with the TIME_WAIT status you have a good chance for running out of ports. There are two ways to address this problem:

- Reduce the number of TCP ports consumed quickly by investigating connection pooling or persistent connections where possible
- Tune some settings in the Windows registry (see below)

Important

The following procedure involves modifying the Windows registry. Before you modify the registry, make sure to back it up and make sure that you understand how to restore it if a problem occurs. For information about how to back up, restore, and edit the registry, view the following article in the Microsoft Knowledge Base: http://support.microsoft.com/kb/256986/EN-US/.

1. Start Registry Editor (Regedt32.exe).
2. Locate the following key in the registry:

   ```
   HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
   ```

3. On the Edit menu, click Add Value, and then add the following registry value:

   ```
   Value Name: MaxUserPort
   Data Type: REG_DWORD
   Value: 65534
   ```

   This sets the number of ephemeral ports available to any user. The valid range is between 5000 and 65534 (decimal). The default value is 0x1388 (5000 decimal).

4. On the Edit menu, click Add Value, and then add the following registry value:

   ```
   Value Name: TcpTimedWaitDelay
   Data Type: REG_DWORD
   Value: 30
   ```

   This sets the number of seconds to hold a TCP port connection in TIME_WAIT state before closing. The valid range is between 30 and 300 decimal, although you may wish to check with Microsoft for the latest permitted values. The default value is 0x78 (120 decimal).

5. Quit Registry Editor.

6. Reboot the machine.

Note: Undoing the above should be as simple as deleting the registry entries you've created.

B.3.2.3 Lost connection to MySQL server

There are three likely causes for this error message.

Usually it indicates network connectivity trouble and you should check the condition of your network if this error occurs frequently. If the error message includes “during query,” this is probably the case you are experiencing.

Sometimes the “during query” form happens when millions of rows are being sent as part of one or more queries. If you know that this is happening, you should try increasing `net_read_timeout` from its default of 30 seconds to 60 seconds or longer, sufficient for the data transfer to complete.

More rarely, it can happen when the client is attempting the initial connection to the server. In this case, if your `connect_timeout` value is set to only a few seconds, you may be able to resolve the problem by increasing it to ten seconds, perhaps more if you have a very long distance or slow connection. You can determine whether you are experiencing this more uncommon cause by using `SHOW GLOBAL STATUS LIKE 'Aborted_connects'`. It increases by one for each initial connection attempt that the server aborts. You may see “reading authorization packet” as part of the error message; if so, that also suggests that this is the solution that you need.

If the cause is none of those just described, you may be experiencing a problem with `BLOB` values that are larger than `max_allowed_packet`, which can cause this error with some clients. Sometime you may see an `ER_NET_PACKET_TOO_LARGE` error, and that confirms that you need to increase `max_allowed_packet`.

B.3.2.4 Password Fails When Entered Interactively

MySQL client programs prompt for a password when invoked with a `--password` or `-p` option that has no following password value:

```
$> mysql -u user_name -p
Enter password:
```

On some systems, you may find that your password works when specified in an option file or on the command line, but not when you enter it interactively at the `Enter password:` prompt. This occurs when the library provided by the system to read passwords limits password values to a small number of characters (typically eight). That is a problem with the system library, not with MySQL. To work around it, change your MySQL password to a value that is eight or fewer characters long, or put your password in an option file.

B.3.2.5 Too many connections

If clients encounter `Too many connections` errors when attempting to connect to the `mysqld` server, all available connections are in use by other clients.

The permitted number of connections is controlled by the `max_connections` system variable. To support more connections, set `max_connections` to a larger value.

`mysqld` actually permits `max_connections` + 1 client connections. The extra connection is reserved for use by accounts that have the `CONNECTION_ADMIN` privilege (or the deprecated `SUPER` privilege). By granting the privilege to administrators and not to normal users (who should not need it), an administrator can connect to the server and use `SHOW PROCESSLIST` to diagnose problems even if the maximum number of unprivileged clients are connected. See Section 13.7.7.29, “SHOW PROCESSLIST Statement”.

The server also permits administrative connections on a dedicated interface. For more information about how the server handles client connections, see Section 5.1.12.1, “Connection Interfaces”.

B.3.2.6 Out of memory

If you issue a query using the `mysql` client program and receive an error like the following one, it means that `mysql` does not have enough memory to store the entire query result:

```
mysql: Out of memory at line 42, 'malloc.c'
mysql: needed 8136 byte (8k), memory in use: 12481367 bytes (12189k)
ERROR 2008: MySQL client ran out of memory
```

To remedy the problem, first check whether your query is correct. Is it reasonable that it should return so many rows? If not, correct the query and try again. Otherwise, you can invoke `mysql` with the `--quick` option. This causes it to use the `mysql_use_result()` C API function to retrieve the result set, which places less of a load on the client (but more on the server).

B.3.2.7 MySQL server has gone away

This section also covers the related `Lost connection to server during query` error.

The most common reason for the `MySQL server has gone away` error is that the server timed out and closed the connection. In this case, you normally get one of the following error codes (which one you get is operating system-dependent).

Error Code	Description
`CR_SERVER_GONE_ERROR`	The client couldn't send a question to the server.
`CR_SERVER_LOST`	The client didn't get an error when writing to the server, but it didn't get a full answer (or any answer) to the question.

By default, the server closes the connection after eight hours if nothing has happened. You can change the time limit by setting the `wait_timeout` variable when you start `mysqld`. See Section 5.1.8, “Server System Variables”.

If you have a script, you just have to issue the query again for the client to do an automatic reconnection. This assumes that you have automatic reconnection in the client enabled (which is the default for the `mysql` command-line client).

Some other common reasons for the `MySQL server has gone away` error are:

- You (or the db administrator) has killed the running thread with a `KILL` statement or a `mysqladmin kill` command.
- You tried to run a query after closing the connection to the server. This indicates a logic error in the application that should be corrected.
- A client application running on a different host does not have the necessary privileges to connect to the MySQL server from that host.
- You got a timeout from the TCP/IP connection on the client side. This may happen if you have been using the commands: `mysql_options(..., MYSQL_OPT_READ_TIMEOUT,...)` or `mysql_options(..., MYSQL_OPT_WRITE_TIMEOUT,...)`. In this case increasing the timeout may help solve the problem.
- You have encountered a timeout on the server side and the automatic reconnection in the client is disabled (the `reconnect` flag in the `MYSQL` structure is equal to 0).
- You are using a Windows client and the server had dropped the connection (probably because `wait_timeout` expired) before the command was issued.

  The problem on Windows is that in some cases MySQL does not get an error from the OS when writing to the TCP/IP connection to the server, but instead gets the error when trying to read the answer from the connection.

The solution to this is to either do a `mysql_ping()` on the connection if there has been a long time since the last query (this is what Connector/ODBC does) or set `wait_timeout` on the `mysqld` server so high that it in practice never times out.

- You can also get these errors if you send a query to the server that is incorrect or too large. If `mysqld` receives a packet that is too large or out of order, it assumes that something has gone wrong with the client and closes the connection. If you need big queries (for example, if you are working with big `BLOB` columns), you can increase the query limit by setting the server's `max_allowed_packet` variable, which has a default value of 64MB. You may also need to increase the maximum packet size on the client end. More information on setting the packet size is given in Section B.3.2.8, “Packet Too Large”.

  An `INSERT` or `REPLACE` statement that inserts a great many rows can also cause these sorts of errors. Either one of these statements sends a single request to the server irrespective of the number of rows to be inserted; thus, you can often avoid the error by reducing the number of rows sent per `INSERT` or `REPLACE`.

- It is also possible to see this error if host name lookups fail (for example, if the DNS server on which your server or network relies goes down). This is because MySQL is dependent on the host system for name resolution, but has no way of knowing whether it is working—from MySQL's point of view the problem is indistinguishable from any other network timeout.

  You may also see the `MySQL server has gone away` error if MySQL is started with the `skip_networking` system variable enabled.

  Another networking issue that can cause this error occurs if the MySQL port (default 3306) is blocked by your firewall, thus preventing any connections at all to the MySQL server.

- You can also encounter this error with applications that fork child processes, all of which try to use the same connection to the MySQL server. This can be avoided by using a separate connection for each child process.

- You have encountered a bug where the server died while executing the query.

You can check whether the MySQL server died and restarted by executing `mysqladmin version` and examining the server's uptime. If the client connection was broken because `mysqld` crashed and restarted, you should concentrate on finding the reason for the crash. Start by checking whether issuing the query again kills the server again. See Section B.3.3.3, “What to Do If MySQL Keeps Crashing”.

You can obtain more information about lost connections by starting `mysqld` with the `log_error_verbosity` system variable set to 3. This logs some of the disconnection messages in the `hostname.err` file. See Section 5.4.2, “The Error Log”.

If you want to create a bug report regarding this problem, be sure that you include the following information:

- Indicate whether the MySQL server died. You can find information about this in the server error log. See Section B.3.3.3, “What to Do If MySQL Keeps Crashing”.

- If a specific query kills `mysqld` and the tables involved were checked with `CHECK TABLE` before you ran the query, can you provide a reproducible test case? See Section 5.9, “Debugging MySQL”.

- What is the value of the `wait_timeout` system variable in the MySQL server? (`mysqladmin variables` gives you the value of this variable.)

- Have you tried to run `mysqld` with the general query log enabled to determine whether the problem query appears in the log? (See Section 5.4.3, “The General Query Log”.)

See also Section B.3.2.9, “Communication Errors and Aborted Connections”, and Section 1.5, “How to Report Bugs or Problems”.

B.3.2.8 Packet Too Large

A communication packet is a single SQL statement sent to the MySQL server, a single row that is sent to the client, or a binary log event sent from a replication source server to a replica.

The largest possible packet that can be transmitted to or from a MySQL 8.0 server or client is 1GB.

When a MySQL client or the `mysqld` server receives a packet bigger than `max_allowed_packet` bytes, it issues an `ER_NET_PACKET_TOO_LARGE` error and closes the connection. With some clients, you may also get a `Lost connection to MySQL server during query` error if the communication packet is too large.

Both the client and the server have their own `max_allowed_packet` variable, so if you want to handle big packets, you must increase this variable both in the client and in the server.

If you are using the `mysql` client program, its default `max_allowed_packet` variable is 16MB. To set a larger value, start `mysql` like this:

```
$> mysql --max_allowed_packet=32M
```

That sets the packet size to 32MB.

The server's default `max_allowed_packet` value is 64MB. You can increase this if the server needs to handle big queries (for example, if you are working with big `BLOB` columns). For example, to set the variable to 128MB, start the server like this:

```
$> mysqld --max_allowed_packet=128M
```

You can also use an option file to set `max_allowed_packet`. For example, to set the size for the server to 128MB, add the following lines in an option file:

```
[mysqld]
max_allowed_packet=128M
```

It is safe to increase the value of this variable because the extra memory is allocated only when needed. For example, `mysqld` allocates more memory only when you issue a long query or when `mysqld` must return a large result row. The small default value of the variable is a precaution to catch incorrect packets between the client and server and also to ensure that you do not run out of memory by using large packets accidentally.

You can also get strange problems with large packets if you are using large `BLOB` values but have not given `mysqld` access to enough memory to handle the query. If you suspect this is the case, try adding `ulimit -d 256000` to the beginning of the `mysqld_safe` script and restarting `mysqld`.

B.3.2.9 Communication Errors and Aborted Connections

If connection problems occur such as communication errors or aborted connections, use these sources of information to diagnose problems:

- The error log. See Section 5.4.2, “The Error Log”.
- The general query log. See Section 5.4.3, “The General Query Log”.
- The `Aborted_xxx` and `Connection_errors_xxx` status variables. See Section 5.1.10, “Server Status Variables”.
- The host cache, which is accessible using the Performance Schema `host_cache` table. See Section 5.1.12.3, “DNS Lookups and the Host Cache”, and Section 27.12.21.2, “The host_cache Table”.

If the `log_error_verbosity` system variable is set to 3, you might find messages like this in your error log:

```
[Note] Aborted connection 854 to db: 'employees' user: 'josh'
```

If a client is unable even to connect, the server increments the `Aborted_connects` status variable. Unsuccessful connection attempts can occur for the following reasons:

- A client attempts to access a database but has no privileges for it.
- A client uses an incorrect password.
- A connection packet does not contain the right information.
- It takes more than `connect_timeout` seconds to obtain a connect packet. See Section 5.1.8, “Server System Variables”.

If these kinds of things happen, it might indicate that someone is trying to break into your server! If the general query log is enabled, messages for these types of problems are logged to it.

If a client successfully connects but later disconnects improperly or is terminated, the server increments the `Aborted_clients` status variable, and logs an `Aborted connection` message to the error log. The cause can be any of the following:

- The client program did not call `mysql_close()` before exiting.
- The client had been sleeping more than `wait_timeout` or `interactive_timeout` seconds without issuing any requests to the server. See Section 5.1.8, “Server System Variables”.
- The client program ended abruptly in the middle of a data transfer.

Other reasons for problems with aborted connections or aborted clients:

- The `max_allowed_packet` variable value is too small or queries require more memory than you have allocated for `mysqld`. See Section B.3.2.8, “Packet Too Large”.
- Use of Ethernet protocol with Linux, both half and full duplex. Some Linux Ethernet drivers have this bug. You should test for this bug by transferring a huge file using FTP between the client and server machines. If a transfer goes in burst-pause-burst-pause mode, you are experiencing a Linux duplex syndrome. Switch the duplex mode for both your network card and hub/switch to either full duplex or to half duplex and test the results to determine the best setting.
- A problem with the thread library that causes interrupts on reads.
- Badly configured TCP/IP.
- Faulty Ethernets, hubs, switches, cables, and so forth. This can be diagnosed properly only by replacing hardware.

See also Section B.3.2.7, “MySQL server has gone away”.

B.3.2.10 The table is full

If a table-full error occurs, it may be that the disk is full or that the table has reached its maximum size. The effective maximum table size for MySQL databases is usually determined by operating system constraints on file sizes, not by MySQL internal limits. See Section 8.4.6, “Limits on Table Size”.

B.3.2.11 Can't create/write to file

If you get an error of the following type for some queries, it means that MySQL cannot create a temporary file for the result set in the temporary directory:

```
Can't create/write to file '\\sqla3fe_0.ism'.
```

The preceding error is a typical message for Windows; the Unix message is similar.

One fix is to start `mysqld` with the `--tmpdir` option or to add the option to the `[mysqld]` section of your option file. For example, to specify a directory of `C:\temp`, use these lines:

```
[mysqld]
tmpdir=C:/temp
```

The `C:\temp` directory must exist and have sufficient space for the MySQL server to write to. See Section 4.2.2.2, “Using Option Files”.

Another cause of this error can be permissions issues. Make sure that the MySQL server can write to the `tmpdir` directory.

Check also the error code that you get with `perror`. One reason the server cannot write to a table is that the file system is full:

```
$> perror 28
OS error code  28:  No space left on device
```

If you get an error of the following type during startup, it indicates that the file system or directory used for storing data files is write protected. Provided that the write error is to a test file, the error is not serious and can be safely ignored.

```
Can't create test file /usr/local/mysql/data/master.lower-test
```

B.3.2.12 Commands out of sync

If you get `Commands out of sync; you can't run this command now` in your client code, you are calling client functions in the wrong order.

This can happen, for example, if you are using `mysql_use_result()` and try to execute a new query before you have called `mysql_free_result()`. It can also happen if you try to execute two queries that return data without calling `mysql_use_result()` or `mysql_store_result()` in between.

B.3.2.13 Ignoring user

If you get the following error, it means that when `mysqld` was started or when it reloaded the grant tables, it found an account in the `user` table that had an invalid password.

`Found wrong password for user '`*`some_user`*`'@'`*`some_host`*`'; ignoring user`

As a result, the account is simply ignored by the permission system. To fix this problem, assign a new, valid password to the account.

B.3.2.14 Table 'tbl_name' doesn't exist

If you get either of the following errors, it usually means that no table exists in the default database with the given name:

```
Table 'tbl_name' doesn't exist
Can't find file: 'tbl_name' (errno: 2)
```

In some cases, it may be that the table does exist but that you are referring to it incorrectly:

- Because MySQL uses directories and files to store databases and tables, database and table names are case-sensitive if they are located on a file system that has case-sensitive file names.
- Even for file systems that are not case-sensitive, such as on Windows, all references to a given table within a query must use the same lettercase.

You can check which tables are in the default database with `SHOW TABLES`. See Section 13.7.7, “SHOW Statements”.

B.3.2.15 Can't initialize character set

You might see an error like this if you have character set problems:

```
MySQL Connection Failed: Can't initialize character set charset_name
```

This error can have any of the following causes:

- The character set is a multibyte character set and you have no support for the character set in the client. In this case, you need to recompile the client by running `CMake` with the `-DDEFAULT_CHARSET=charset_name` option. See Section 2.8.7, “MySQL Source-Configuration Options”.

  All standard MySQL binaries are compiled with support for all multibyte character sets.

- The character set is a simple character set that is not compiled into `mysqld`, and the character set definition files are not in the place where the client expects to find them.

  In this case, you need to use one of the following methods to solve the problem:

  - Recompile the client with support for the character set. See Section 2.8.7, “MySQL Source-Configuration Options”.
  - Specify to the client the directory where the character set definition files are located. For many clients, you can do this with the `--character-sets-dir` option.
  - Copy the character definition files to the path where the client expects them to be.

B.3.2.16 File Not Found and Similar Errors

If you get `ERROR 'file_name' not found (errno: 23)`, `Can't open file: file_name (errno: 24)`, or any other error with `errno 23` or `errno 24` from MySQL, it means that you have not allocated enough file descriptors for the MySQL server. You can use the `perror` utility to get a description of what the error number means:

```
$> perror 23
OS error code  23:  File table overflow
$> perror 24
OS error code  24:  Too many open files
$> perror 11
OS error code  11:  Resource temporarily unavailable
```

The problem here is that `mysqld` is trying to keep open too many files simultaneously. You can either tell `mysqld` not to open so many files at once or increase the number of file descriptors available to `mysqld`.

To tell `mysqld` to keep open fewer files at a time, you can make the table cache smaller by reducing the value of the `table_open_cache` system variable (the default value is 64). This may not entirely prevent running out of file descriptors because in some circumstances the server may attempt to extend the cache size temporarily, as described in Section 8.4.3.1, “How MySQL Opens and Closes Tables”. Reducing the value of `max_connections` also reduces the number of open files (the default value is 100).

To change the number of file descriptors available to `mysqld`, you can use the `--open-files-limit` option to `mysqld_safe` or set the `open_files_limit` system variable. See Section 5.1.8, “Server System Variables”. The easiest way to set these values is to add an option to your option file. See Section 4.2.2.2, “Using Option Files”. If you have an old version of `mysqld` that does not support setting the open files limit, you can edit the `mysqld_safe` script. There is a commented-out line `ulimit -n 256` in the script. You can remove the `#` character to uncomment this line, and change the number `256` to set the number of file descriptors to be made available to `mysqld`.

`--open-files-limit` and `ulimit` can increase the number of file descriptors, but only up to the limit imposed by the operating system. There is also a “hard” limit that can be overridden only if you start `mysqld_safe` or `mysqld` as `root` (just remember that you also need to start the server with the

`--user` option in this case so that it does not continue to run as `root` after it starts up). If you need to increase the operating system limit on the number of file descriptors available to each process, consult the documentation for your system.

Note

If you run the `tcsh` shell, `ulimit` does not work! `tcsh` also reports incorrect values when you ask for the current limits. In this case, you should start `mysqld_safe` using `sh`.

B.3.2.17 Table-Corruption Issues

If you have started `mysqld` with the `myisam_recover_options` system variable set, MySQL automatically checks and tries to repair `MyISAM` tables if they are marked as 'not closed properly' or 'crashed'. If this happens, MySQL writes an entry in the `hostname.err` file `'Warning: Checking table ...'` which is followed by `Warning: Repairing table` if the table needs to be repaired. If you get a lot of these errors, without `mysqld` having died unexpectedly just before, then something is wrong and needs to be investigated further.

When the server detects `MyISAM` table corruption, it writes additional information to the error log, such as the name and line number of the source file, and the list of threads accessing the table. Example: `Got an error from thread_id=1, mi_dynrec.c:368`. This is useful information to include in bug reports.

See also Section 5.1.7, “Server Command Options”, and Section 5.9.1.7, “Making a Test Case If You Experience Table Corruption”.

B.3.3 Administration-Related Issues

B.3.3.1 Problems with File Permissions

If you have problems with file permissions, the `UMASK` or `UMASK_DIR` environment variable might be set incorrectly when `mysqld` starts. For example, `mysqld` might issue the following error message when you create a table:

```
ERROR: Can't find file: 'path/with/file_name' (Errcode: 13)
```

The default `UMASK` and `UMASK_DIR` values are `0640` and `0750`, respectively. `mysqld` assumes that the value for `UMASK` or `UMASK_DIR` is in octal if it starts with a zero. For example, setting `UMASK=0600` is equivalent to `UMASK=384` because 0600 octal is 384 decimal.

Assuming that you start `mysqld` using `mysqld_safe`, change the default `UMASK` value as follows:

```
UMASK=384  # = 600 in octal
export UMASK
mysqld_safe &
```

Note

An exception applies for the error log file if you start `mysqld` using `mysqld_safe`, which does not respect `UMASK`: `mysqld_safe` may create the error log file if it does not exist prior to starting `mysqld`, and `mysqld_safe` uses a umask set to a strict value of `0137`. If this is unsuitable, create the error file manually with the desired access mode prior to executing `mysqld_safe`.

By default, `mysqld` creates database directories with an access permission value of `0750`. To modify this behavior, set the `UMASK_DIR` variable. If you set its value, new directories are created with the combined `UMASK` and `UMASK_DIR` values. For example, to give group access to all new directories, start `mysqld_safe` as follows:

```
UMASK_DIR=504  # = 770 in octal
export UMASK_DIR
```

```
mysqld_safe &
```

For additional details, see Section 4.9, “Environment Variables”.

B.3.3.2 How to Reset the Root Password

If you have never assigned a `root` password for MySQL, the server does not require a password at all for connecting as `root`. However, this is insecure. For instructions on assigning a password, see Section 2.9.4, “Securing the Initial MySQL Account”.

If you know the `root` password and want to change it, see Section 13.7.1.1, “ALTER USER Statement”, and Section 13.7.1.10, “SET PASSWORD Statement”.

If you assigned a `root` password previously but have forgotten it, you can assign a new password. The following sections provide instructions for Windows and Unix and Unix-like systems, as well as generic instructions that apply to any system.

Resetting the Root Password: Windows Systems

On Windows, use the following procedure to reset the password for the MySQL `'root'@'localhost'` account. To change the password for a `root` account with a different host name part, modify the instructions to use that host name.

1. Log on to your system as Administrator.
2. Stop the MySQL server if it is running. For a server that is running as a Windows service, go to the Services manager: From the **Start** menu, select **Control Panel**, then **Administrative Tools**, then **Services**. Find the MySQL service in the list and stop it.

   If your server is not running as a service, you may need to use the Task Manager to force it to stop.
3. Create a text file containing the password-assignment statement on a single line. Replace the password with the password that you want to use.

   ```
   ALTER USER 'root'@'localhost' IDENTIFIED BY 'MyNewPass';
   ```
4. Save the file. This example assumes that you name the file `C:\mysql-init.txt`.
5. Open a console window to get to the command prompt: From the **Start** menu, select **Run**, then enter `cmd` as the command to be run.
6. Start the MySQL server with the `init_file` system variable set to name the file (notice that the backslash in the option value is doubled):

   ```
   C:\> cd "C:\Program Files\MySQL\MySQL Server 8.0\bin"
   C:\> mysqld --init-file=C:\\mysql-init.txt
   ```

   If you installed MySQL to a different location, adjust the `cd` command accordingly.

   The server executes the contents of the file named by the `init_file` system variable at startup, changing the `'root'@'localhost'` account password.

   To have server output to appear in the console window rather than in a log file, add the `--console` option to the `mysqld` command.

   If you installed MySQL using the MySQL Installation Wizard, you may need to specify a `--defaults-file` option. For example:

   ```
   C:\> mysqld
           --defaults-file="C:\\ProgramData\\MySQL\\MySQL Server 8.0\\my.ini"
           --init-file=C:\\mysql-init.txt
   ```

   The appropriate `--defaults-file` setting can be found using the Services Manager: From the **Start** menu, select **Control Panel**, then **Administrative Tools**, then **Services**. Find the MySQL

service in the list, right-click it, and choose the `Properties` option. The `Path to executable` field contains the `--defaults-file` setting.

7. After the server has started successfully, delete `C:\mysql-init.txt`.

You should now be able to connect to the MySQL server as `root` using the new password. Stop the MySQL server and restart it normally. If you run the server as a service, start it from the Windows Services window. If you start the server manually, use whatever command you normally use.

Resetting the Root Password: Unix and Unix-Like Systems

On Unix, use the following procedure to reset the password for the MySQL `'root'@'localhost'` account. To change the password for a `root` account with a different host name part, modify the instructions to use that host name.

The instructions assume that you start the MySQL server from the Unix login account that you normally use for running it. For example, if you run the server using the `mysql` login account, you should log in as `mysql` before using the instructions. Alternatively, you can log in as `root`, but in this case you *must* start `mysqld` with the `--user=mysql` option. If you start the server as `root` without using `--user=mysql`, the server may create `root`-owned files in the data directory, such as log files, and these may cause permission-related problems for future server startups. If that happens, you must either change the ownership of the files to `mysql` or remove them.

1. Log on to your system as the Unix user that the MySQL server runs as (for example, `mysql`).
2. Stop the MySQL server if it is running. Locate the `.pid` file that contains the server's process ID. The exact location and name of this file depend on your distribution, host name, and configuration. Common locations are `/var/lib/mysql/`, `/var/run/mysqld/`, and `/usr/local/mysql/data/`. Generally, the file name has an extension of `.pid` and begins with either `mysqld` or your system's host name.

   Stop the MySQL server by sending a normal `kill` (not `kill -9`) to the `mysqld` process. Use the actual path name of the `.pid` file in the following command:

   ```
   $> kill `cat /mysql-data-directory/host_name.pid`
   ```

   Use backticks (not forward quotation marks) with the `cat` command. These cause the output of `cat` to be substituted into the `kill` command.
3. Create a text file containing the password-assignment statement on a single line. Replace the password with the password that you want to use.

   ```
   ALTER USER 'root'@'localhost' IDENTIFIED BY 'MyNewPass';
   ```
4. Save the file. This example assumes that you name the file `/home/me/mysql-init`. The file contains the password, so do not save it where it can be read by other users. If you are not logged in as `mysql` (the user the server runs as), make sure that the file has permissions that permit `mysql` to read it.
5. Start the MySQL server with the `init_file` system variable set to name the file:

   ```
   $> mysqld --init-file=/home/me/mysql-init &
   ```

   The server executes the contents of the file named by the `init_file` system variable at startup, changing the `'root'@'localhost'` account password.

   Other options may be necessary as well, depending on how you normally start your server. For example, `--defaults-file` may be needed before the `init_file` argument.
6. After the server has started successfully, delete `/home/me/mysql-init`.

You should now be able to connect to the MySQL server as `root` using the new password. Stop the server and restart it normally.

Resetting the Root Password: Generic Instructions

The preceding sections provide password-resetting instructions specifically for Windows and Unix and Unix-like systems. Alternatively, on any platform, you can reset the password using the `mysql` client (but this approach is less secure):

1. Stop the MySQL server if necessary, then restart it with the `--skip-grant-tables` option. This enables anyone to connect without a password and with all privileges, and disables account-management statements such as `ALTER USER` and `SET PASSWORD`. Because this is insecure, if the server is started with the `--skip-grant-tables` option, it also disables remote connections by enabling `skip_networking`.

2. Connect to the MySQL server using the `mysql` client; no password is necessary because the server was started with `--skip-grant-tables`:

   ```
   $> mysql
   ```

3. In the `mysql` client, tell the server to reload the grant tables so that account-management statements work:

   ```
   mysql> FLUSH PRIVILEGES;
   ```

   Then change the `'root'@'localhost'` account password. Replace the password with the password that you want to use. To change the password for a `root` account with a different host name part, modify the instructions to use that host name.

   ```
   mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'MyNewPass';
   ```

You should now be able to connect to the MySQL server as `root` using the new password. Stop the server and restart it normally (without the `--skip-grant-tables` option and without enabling the `skip_networking` system variable).

B.3.3.3 What to Do If MySQL Keeps Crashing

Each MySQL version is tested on many platforms before it is released. This does not mean that there are no bugs in MySQL, but if there are bugs, they should be very few and can be hard to find. If you have a problem, it always helps if you try to find out exactly what crashes your system, because you have a much better chance of getting the problem fixed quickly.

First, you should try to find out whether the problem is that the `mysqld` server dies or whether your problem has to do with your client. You can check how long your `mysqld` server has been up by executing `mysqladmin version`. If `mysqld` has died and restarted, you may find the reason by looking in the server's error log. See Section 5.4.2, “The Error Log”.

On some systems, you can find in the error log a stack trace of where `mysqld` died. Note that the variable values written in the error log may not always be 100% correct.

If you find that `mysqld` fails at startup during `InnoDB` recovery, refer to Section 15.21.2, “Troubleshooting Recovery Failures”.

Many unexpected server exits are caused by corrupted data files or index files. MySQL updates the files on disk with the `write()` system call after every SQL statement and before the client is notified about the result. (This is not true if you are running with the `delay_key_write` system variable enabled, in which case data files are written but not index files.) This means that data file contents are safe even if `mysqld` crashes, because the operating system ensures that the unflushed data is written to disk. You can force MySQL to flush everything to disk after every SQL statement by starting `mysqld` with the `--flush` option.

The preceding means that normally you should not get corrupted tables unless one of the following happens:

- The MySQL server or the server host was killed in the middle of an update.

- You have found a bug in `mysqld` that caused it to die in the middle of an update.
- Some external program is manipulating data files or index files at the same time as `mysqld` without locking the table properly.
- You are running many `mysqld` servers using the same data directory on a system that does not support good file system locks (normally handled by the `lockd` lock manager), or you are running multiple servers with external locking disabled.
- You have a crashed data file or index file that contains very corrupt data that confused `mysqld`.
- You have found a bug in the data storage code. This isn't likely, but it is at least possible. In this case, you can try to change the storage engine to another engine by using `ALTER TABLE` on a repaired copy of the table.

Because it is very difficult to know why something is crashing, first try to check whether things that work for others result in an unexpected exit for you. Try the following things:

- Stop the `mysqld` server with `mysqladmin shutdown`, run `myisamchk --silent --force */*.MYI` from the data directory to check all `MyISAM` tables, and restart `mysqld`. This ensures that you are running from a clean state. See Chapter 5, *MySQL Server Administration*.
- Start `mysqld` with the general query log enabled (see Section 5.4.3, “The General Query Log”). Then try to determine from the information written to the log whether some specific query kills the server. About 95% of all bugs are related to a particular query. Normally, this is one of the last queries in the log file just before the server restarts. See Section 5.4.3, “The General Query Log”. If you can repeatedly kill MySQL with a specific query, even when you have checked all tables just before issuing it, then you have isolated the bug and should submit a bug report for it. See Section 1.5, “How to Report Bugs or Problems”.
- Try to make a test case that we can use to repeat the problem. See Section 5.9, “Debugging MySQL”.
- Try the `fork_big.pl` script. (It is located in the `tests` directory of source distributions.)
- Configuring MySQL for debugging makes it much easier to gather information about possible errors if something goes wrong. Reconfigure MySQL with the `-DWITH_DEBUG=1` option to `CMake` and then recompile. See Section 5.9, “Debugging MySQL”.
- Make sure that you have applied the latest patches for your operating system.
- Use the `--skip-external-locking` option to `mysqld`. On some systems, the `lockd` lock manager does not work properly; the `--skip-external-locking` option tells `mysqld` not to use external locking. (This means that you cannot run two `mysqld` servers on the same data directory and that you must be careful if you use `myisamchk`. Nevertheless, it may be instructive to try the option as a test.)
- If `mysqld` appears to be running but not responding, try `mysqladmin -u root processlist`. Sometimes `mysqld` is not hung even though it seems unresponsive. The problem may be that all connections are in use, or there may be some internal lock problem. `mysqladmin -u root processlist` usually is able to make a connection even in these cases, and can provide useful information about the current number of connections and their status.
- Run the command `mysqladmin -i 5 status` or `mysqladmin -i 5 -r status` in a separate window to produce statistics while running other queries.
- Try the following:

  1. Start `mysqld` from `gdb` (or another debugger). See Section 5.9, “Debugging MySQL”.
  2. Run your test scripts.

3. Print the backtrace and the local variables at the three lowest levels. In `gdb`, you can do this with the following commands when `mysqld` has crashed inside `gdb`:

   ```
   backtrace
   info local
   up
   info local
   up
   info local
   ```

   With `gdb`, you can also examine which threads exist with `info threads` and switch to a specific thread with `thread N`, where `N` is the thread ID.

- Try to simulate your application with a Perl script to force MySQL to exit or misbehave.
- Send a normal bug report. See Section 1.5, “How to Report Bugs or Problems”. Be even more detailed than usual. Because MySQL works for many people, the crash might result from something that exists only on your computer (for example, an error that is related to your particular system libraries).
- If you have a problem with tables containing dynamic-length rows and you are using only `VARCHAR` columns (not `BLOB` or `TEXT` columns), you can try to change all `VARCHAR` to `CHAR` with `ALTER TABLE`. This forces MySQL to use fixed-size rows. Fixed-size rows take a little extra space, but are much more tolerant to corruption.

  The current dynamic row code has been in use for several years with very few problems, but dynamic-length rows are by nature more prone to errors, so it may be a good idea to try this strategy to see whether it helps.
- Consider the possibility of hardware faults when diagnosing problems. Defective hardware can be the cause of data corruption. Pay particular attention to your memory and disk subsystems when troubleshooting hardware.

B.3.3.4 How MySQL Handles a Full Disk

This section describes how MySQL responds to disk-full errors (such as “no space left on device”), and to quota-exceeded errors (such as “write failed” or “user block limit reached”).

This section is relevant for writes to `MyISAM` tables. It also applies for writes to binary log files and binary log index file, except that references to “row” and “record” should be understood to mean “event.”

When a disk-full condition occurs, MySQL does the following:

- It checks once every minute to see whether there is enough space to write the current row. If there is enough space, it continues as if nothing had happened.
- Every 10 minutes it writes an entry to the log file, warning about the disk-full condition.

To alleviate the problem, take the following actions:

- To continue, you only have to free enough disk space to insert all records.
- Alternatively, to abort the thread, use `mysqladmin kill`. The thread is aborted the next time it checks the disk (in one minute).
- Other threads might be waiting for the table that caused the disk-full condition. If you have several “locked” threads, killing the one thread that is waiting on the disk-full condition enables the other threads to continue.

Exceptions to the preceding behavior are when you use `REPAIR TABLE` or `OPTIMIZE TABLE` or when the indexes are created in a batch after `LOAD DATA` or after an `ALTER TABLE` statement. All of

these statements may create large temporary files that, if left to themselves, would cause big problems for the rest of the system. If the disk becomes full while MySQL is doing any of these operations, it removes the big temporary files and mark the table as crashed. The exception is that for `ALTER TABLE`, the old table is left unchanged.

B.3.3.5 Where MySQL Stores Temporary Files

On Unix, MySQL uses the value of the `TMPDIR` environment variable as the path name of the directory in which to store temporary files. If `TMPDIR` is not set, MySQL uses the system default, which is usually `/tmp`, `/var/tmp`, or `/usr/tmp`.

On Windows, MySQL checks in order the values of the `TMPDIR`, `TEMP`, and `TMP` environment variables. For the first one found to be set, MySQL uses it and does not check those remaining. If none of `TMPDIR`, `TEMP`, or `TMP` are set, MySQL uses the Windows system default, which is usually `C:\windows\temp\`.

If the file system containing your temporary file directory is too small, you can use the `mysqld --tmpdir` option to specify a directory in a file system where you have enough space.

The `--tmpdir` option can be set to a list of several paths that are used in round-robin fashion. Paths should be separated by colon characters (`:`) on Unix and semicolon characters (`;`) on Windows.

Note

To spread the load effectively, these paths should be located on different *physical* disks, not different partitions of the same disk.

If the MySQL server is acting as a replica, you can set the system variable `replica_load_tmpdir` (from MySQL 8.0.26) or `slave_load_tmpdir` (before MySQL 8.0.26) to specify a separate directory for holding temporary files when replicating `LOAD DATA` statements. This directory should be in a disk-based file system (not a memory-based file system) so that the temporary files used to replicate LOAD DATA can survive machine restarts. The directory also should not be one that is cleared by the operating system during the system startup process. However, replication can now continue after a restart if the temporary files have been removed.

MySQL arranges that temporary files are removed if `mysqld` is terminated. On platforms that support it (such as Unix), this is done by unlinking the file after opening it. The disadvantage of this is that the name does not appear in directory listings and you do not see a big temporary file that fills up the file system in which the temporary file directory is located. (In such cases, `lsof +L1` may be helpful in identifying large files associated with `mysqld`.)

When sorting (`ORDER BY` or `GROUP BY`), MySQL normally uses one or two temporary files. The maximum disk space required is determined by the following expression:

```
(length of what is sorted + sizeof(row pointer))
* number of matched rows
* 2
```

The row pointer size is usually four bytes, but may grow in the future for really big tables.

For some statements, MySQL creates temporary SQL tables that are not hidden and have names that begin with `#sql`.

Some `SELECT` queries creates temporary SQL tables to hold intermediate results.

DDL operations that rebuild the table and are not performed online using the `ALGORITHM=INPLACE` technique create a temporary copy of the original table in the same directory as the original table.

Online DDL operations may use temporary log files for recording concurrent DML, temporary sort files when creating an index, and temporary intermediate tables files when rebuilding the table. For more information, see Section 15.12.3, “Online DDL Space Requirements”.

InnoDB user-created temporary tables and on-disk internal temporary tables are created in a temporary tablespace file named ibtmp1 in the MySQL data directory. For more information, see Section 15.6.3.5, “Temporary Tablespaces”.

See also Section 15.15.7, “InnoDB INFORMATION_SCHEMA Temporary Table Info Table”.

The optional EXTENDED modifier causes SHOW TABLES to list hidden tables created by failed ALTER TABLE statements. See Section 13.7.7.39, “SHOW TABLES Statement”.

B.3.3.6 How to Protect or Change the MySQL Unix Socket File

The default location for the Unix socket file that the server uses for communication with local clients is /tmp/mysql.sock. (For some distribution formats, the directory might be different, such as /var/lib/mysql for RPMs.)

On some versions of Unix, anyone can delete files in the /tmp directory or other similar directories used for temporary files. If the socket file is located in such a directory on your system, this might cause problems.

On most versions of Unix, you can protect your /tmp directory so that files can be deleted only by their owners or the superuser (root). To do this, set the sticky bit on the /tmp directory by logging in as root and using the following command:

```
$> chmod +t /tmp
```

You can check whether the sticky bit is set by executing ls -ld /tmp. If the last permission character is t, the bit is set.

Another approach is to change the place where the server creates the Unix socket file. If you do this, you should also let client programs know the new location of the file. You can specify the file location in several ways:

- Specify the path in a global or local option file. For example, put the following lines in /etc/my.cnf:

  ```
  [mysqld]
  socket=/path/to/socket

  [client]
  socket=/path/to/socket
  ```

  See Section 4.2.2.2, “Using Option Files”.

- Specify a --socket option on the command line to mysqld_safe and when you run client programs.

- Set the MYSQL_UNIX_PORT environment variable to the path of the Unix socket file.

- Recompile MySQL from source to use a different default Unix socket file location. Define the path to the file with the MYSQL_UNIX_ADDR option when you run CMake. See Section 2.8.7, “MySQL Source-Configuration Options”.

You can test whether the new socket location works by attempting to connect to the server with this command:

```
$> mysqladmin --socket=/path/to/socket version
```

B.3.3.7 Time Zone Problems

If you have a problem with SELECT NOW() returning values in UTC and not your local time, you have to tell the server your current time zone. The same applies if UNIX_TIMESTAMP() returns the wrong value. This should be done for the environment in which the server runs (for example, in mysqld_safe or mysql.server). See Section 4.9, “Environment Variables”.

You can set the time zone for the server with the `--timezone=timezone_name` option to `mysqld_safe`. You can also set it by setting the `TZ` environment variable before you start `mysqld`.

The permissible values for `--timezone` or `TZ` are system dependent. Consult your operating system documentation to see what values are acceptable.

B.3.4 Query-Related Issues

B.3.4.1 Case Sensitivity in String Searches

For nonbinary strings (`CHAR`, `VARCHAR`, `TEXT`), string searches use the collation of the comparison operands. For binary strings (`BINARY`, `VARBINARY`, `BLOB`), comparisons use the numeric values of the bytes in the operands; this means that for alphabetic characters, comparisons are case-sensitive.

A comparison between a nonbinary string and binary string is treated as a comparison of binary strings.

Simple comparison operations (`>=, >, =, <, <=`, sorting, and grouping) are based on each character's "sort value." Characters with the same sort value are treated as the same character. For example, if `e` and `é` have the same sort value in a given collation, they compare as equal.

The default character set and collation are `utf8mb4` and `utf8mb4_0900_ai_ci`, so nonbinary string comparisons are case-insensitive by default. This means that if you search with *`col_name`* `LIKE 'a%'`, you get all column values that start with `A` or `a`. To make this search case-sensitive, make sure that one of the operands has a case-sensitive or binary collation. For example, if you are comparing a column and a string that both have the `utf8mb4` character set, you can use the `COLLATE` operator to cause either operand to have the `utf8mb4_0900_as_cs` or `utf8mb4_bin` collation:

```
col_name COLLATE utf8mb4_0900_as_cs LIKE 'a%'
col_name LIKE 'a%' COLLATE utf8mb4_0900_as_cs
col_name COLLATE utf8mb4_bin LIKE 'a%'
col_name LIKE 'a%' COLLATE utf8mb4_bin
```

If you want a column always to be treated in case-sensitive fashion, declare it with a case-sensitive or binary collation. See Section 13.1.20, "CREATE TABLE Statement".

To cause a case-sensitive comparison of nonbinary strings to be case-insensitive, use `COLLATE` to name a case-insensitive collation. The strings in the following example normally are case-sensitive, but `COLLATE` changes the comparison to be case-insensitive:

```
mysql> SET NAMES 'utf8mb4';
mysql> SET @s1 = 'MySQL' COLLATE utf8mb4_bin,
           @s2 = 'mysql' COLLATE utf8mb4_bin;
mysql> SELECT @s1 = @s2;
+-----------+
| @s1 = @s2 |
+-----------+
|         0 |
+-----------+
mysql> SELECT @s1 COLLATE utf8mb4_0900_ai_ci = @s2;
+-------------------------------------+
| @s1 COLLATE utf8mb4_0900_ai_ci = @s2 |
+-------------------------------------+
|                                   1 |
+-------------------------------------+
```

A binary string is case-sensitive in comparisons. To compare the string as case-insensitive, convert it to a nonbinary string and use `COLLATE` to name a case-insensitive collation:

```
mysql> SET @s = BINARY 'MySQL';
mysql> SELECT @s = 'mysql';
+--------------+
| @s = 'mysql' |
+--------------+
|            0 |
```

```
+-------------+
mysql> SELECT CONVERT(@s USING utf8mb4) COLLATE utf8mb4_0900_ai_ci = 'mysql';
+-------------------------------------------------------------------+
| CONVERT(@s USING utf8mb4) COLLATE utf8mb4_0900_ai_ci = 'mysql' |
+-------------------------------------------------------------------+
|                                                                 1 |
+-------------------------------------------------------------------+
```

To determine whether a value is compared as a nonbinary or binary string, use the `COLLATION()` function. This example shows that `VERSION()` returns a string that has a case-insensitive collation, so comparisons are case-insensitive:

```
mysql> SELECT COLLATION(VERSION());
+----------------------+
| COLLATION(VERSION()) |
+----------------------+
| utf8mb3_general_ci   |
+----------------------+
```

For binary strings, the collation value is `binary`, so comparisons are case sensitive. One context in which you can expect to see `binary` is for compression functions, which return binary strings as a general rule: string:

```
mysql> SELECT COLLATION(COMPRESS('x'));
+--------------------------+
| COLLATION(COMPRESS('x')) |
+--------------------------+
| binary                   |
+--------------------------+
```

To check the sort value of a string, the `WEIGHT_STRING()` may be helpful. See Section 12.8, “String Functions and Operators”.

B.3.4.2 Problems Using DATE Columns

The format of a `DATE` value is `'YYYY-MM-DD'`. According to standard SQL, no other format is permitted. You should use this format in `UPDATE` expressions and in the `WHERE` clause of `SELECT` statements. For example:

```
SELECT * FROM t1 WHERE date >= '2003-05-05';
```

As a convenience, MySQL automatically converts a date to a number if the date is used in numeric context and vice versa. MySQL also permits a “relaxed” string format when updating and in a `WHERE` clause that compares a date to a `DATE`, `DATETIME`, or `TIMESTAMP` column. “Relaxed” format means that any punctuation character may be used as the separator between parts. For example, `'2004-08-15'` and `'2004#08#15'` are equivalent. MySQL can also convert a string containing no separators (such as `'20040815'`), provided it makes sense as a date.

When you compare a `DATE`, `TIME`, `DATETIME`, or `TIMESTAMP` to a constant string with the `<`, `<=`, `=`, `>=`, `>`, or `BETWEEN` operators, MySQL normally converts the string to an internal long integer for faster comparison (and also for a bit more “relaxed” string checking). However, this conversion is subject to the following exceptions:

- When you compare two columns
- When you compare a `DATE`, `TIME`, `DATETIME`, or `TIMESTAMP` column to an expression
- When you use any comparison method other than those just listed, such as `IN` or `STRCMP()`.

For those exceptions, the comparison is done by converting the objects to strings and performing a string comparison.

To be on the safe side, assume that strings are compared as strings and use the appropriate string functions if you want to compare a temporal value to a string.

The special “zero” date `'0000-00-00'` can be stored and retrieved as `'0000-00-00'`. When a `'0000-00-00'` date is used through Connector/ODBC, it is automatically converted to `NULL` because ODBC cannot handle that kind of date.

Because MySQL performs the conversions just described, the following statements work (assume that `idate` is a `DATE` column):

```
INSERT INTO t1 (idate) VALUES (19970505);
INSERT INTO t1 (idate) VALUES ('19970505');
INSERT INTO t1 (idate) VALUES ('97-05-05');
INSERT INTO t1 (idate) VALUES ('1997.05.05');
INSERT INTO t1 (idate) VALUES ('1997 05 05');
INSERT INTO t1 (idate) VALUES ('0000-00-00');

SELECT idate FROM t1 WHERE idate >= '1997-05-05';
SELECT idate FROM t1 WHERE idate >= 19970505;
SELECT MOD(idate,100) FROM t1 WHERE idate >= 19970505;
SELECT idate FROM t1 WHERE idate >= '19970505';
```

However, the following statement does not work:

```
SELECT idate FROM t1 WHERE STRCMP(idate,'20030505')=0;
```

`STRCMP()` is a string function, so it converts `idate` to a string in *`'YYYY-MM-DD'`* format and performs a string comparison. It does not convert `'20030505'` to the date `'2003-05-05'` and perform a date comparison.

If you enable the `ALLOW_INVALID_DATES` SQL mode, MySQL permits you to store dates that are given only limited checking: MySQL requires only that the day is in the range from 1 to 31 and the month is in the range from 1 to 12. This makes MySQL very convenient for Web applications where you obtain year, month, and day in three different fields and you want to store exactly what the user inserted (without date validation).

MySQL permits you to store dates where the day or month and day are zero. This is convenient if you want to store a birthdate in a `DATE` column and you know only part of the date. To disallow zero month or day parts in dates, enable the `NO_ZERO_IN_DATE` mode.

MySQL permits you to store a “zero” value of `'0000-00-00'` as a “dummy date.” This is in some cases more convenient than using `NULL` values. If a date to be stored in a `DATE` column cannot be converted to any reasonable value, MySQL stores `'0000-00-00'`. To disallow `'0000-00-00'`, enable the `NO_ZERO_DATE` mode.

To have MySQL check all dates and accept only legal dates (unless overridden by `IGNORE`), set the `sql_mode` system variable to `"NO_ZERO_IN_DATE,NO_ZERO_DATE"`.

B.3.4.3 Problems with NULL Values

The concept of the `NULL` value is a common source of confusion for newcomers to SQL, who often think that `NULL` is the same thing as an empty string `''`. This is not the case. For example, the following statements are completely different:

```
mysql> INSERT INTO my_table (phone) VALUES (NULL);
mysql> INSERT INTO my_table (phone) VALUES ('');
```

Both statements insert a value into the `phone` column, but the first inserts a `NULL` value and the second inserts an empty string. The meaning of the first can be regarded as “phone number is not known” and the meaning of the second can be regarded as “the person is known to have no phone, and thus no phone number.”

To help with `NULL` handling, you can use the `IS NULL` and `IS NOT NULL` operators and the `IFNULL()` function.

In SQL, the `NULL` value is never true in comparison to any other value, even `NULL`. An expression that contains `NULL` always produces a `NULL` value unless otherwise indicated in the documentation for the operators and functions involved in the expression. All columns in the following example return `NULL`:

```
mysql> SELECT NULL, 1+NULL, CONCAT('Invisible',NULL);
```

To search for column values that are `NULL`, you cannot use an `expr = NULL` test. The following statement returns no rows, because `expr = NULL` is never true for any expression:

```
mysql> SELECT * FROM my_table WHERE phone = NULL;
```

To look for `NULL` values, you must use the `IS NULL` test. The following statements show how to find the `NULL` phone number and the empty phone number:

```
mysql> SELECT * FROM my_table WHERE phone IS NULL;
mysql> SELECT * FROM my_table WHERE phone = '';
```

See Section 3.3.4.6, “Working with NULL Values”, for additional information and examples.

You can add an index on a column that can have `NULL` values if you are using the `MyISAM`, `InnoDB`, or `MEMORY` storage engine. Otherwise, you must declare an indexed column `NOT NULL`, and you cannot insert `NULL` into the column.

When reading data with `LOAD DATA`, empty or missing columns are updated with `''`. To load a `NULL` value into a column, use `\N` in the data file. The literal word `NULL` may also be used under some circumstances. See Section 13.2.9, “LOAD DATA Statement”.

When using `DISTINCT`, `GROUP BY`, or `ORDER BY`, all `NULL` values are regarded as equal.

When using `ORDER BY`, `NULL` values are presented first, or last if you specify `DESC` to sort in descending order.

Aggregate (group) functions such as `COUNT()`, `MIN()`, and `SUM()` ignore `NULL` values. The exception to this is `COUNT(*)`, which counts rows and not individual column values. For example, the following statement produces two counts. The first is a count of the number of rows in the table, and the second is a count of the number of non-`NULL` values in the `age` column:

```
mysql> SELECT COUNT(*), COUNT(age) FROM person;
```

For some data types, MySQL handles `NULL` values in special ways. For example, if you insert `NULL` into an integer or floating-point column that has the `AUTO_INCREMENT` attribute, the next number in the sequence is inserted. Under certain conditions, if you insert `NULL` into a `TIMESTAMP` column, the current date and time is inserted; this behavior depends in part on the server SQL mode (see Section 5.1.11, “Server SQL Modes”) as well as the value of the `explicit_defaults_for_timestamp` system variable.

B.3.4.4 Problems with Column Aliases

An alias can be used in a query select list to give a column a different name. You can use the alias in `GROUP BY`, `ORDER BY`, or `HAVING` clauses to refer to the column:

```
SELECT SQRT(a*b) AS root FROM tbl_name
  GROUP BY root HAVING root > 0;
SELECT id, COUNT(*) AS cnt FROM tbl_name
  GROUP BY id HAVING cnt > 0;
SELECT id AS 'Customer identity' FROM tbl_name;
```

Standard SQL disallows references to column aliases in a `WHERE` clause. This restriction is imposed because when the `WHERE` clause is evaluated, the column value may not yet have been determined. For example, the following query is illegal:

```
SELECT id, COUNT(*) AS cnt FROM tbl_name
  WHERE cnt > 0 GROUP BY id;
```

The `WHERE` clause determines which rows should be included in the `GROUP BY` clause, but it refers to the alias of a column value that is not known until after the rows have been selected, and grouped by the `GROUP BY`.

In the select list of a query, a quoted column alias can be specified using identifier or string quoting characters:

```
SELECT 1 AS `one`, 2 AS 'two';
```

Elsewhere in the statement, quoted references to the alias must use identifier quoting or the reference is treated as a string literal. For example, this statement groups by the values in column `id`, referenced using the alias `` `a` ``:

```
SELECT id AS 'a', COUNT(*) AS cnt FROM tbl_name
  GROUP BY `a`;
```

This statement groups by the literal string `'a'` and does not work as you may expect:

```
SELECT id AS 'a', COUNT(*) AS cnt FROM tbl_name
  GROUP BY 'a';
```

B.3.4.5 Rollback Failure for Nontransactional Tables

If you receive the following message when trying to perform a `ROLLBACK`, it means that one or more of the tables you used in the transaction do not support transactions:

```
Warning: Some non-transactional changed tables couldn't be rolled back
```

These nontransactional tables are not affected by the `ROLLBACK` statement.

If you were not deliberately mixing transactional and nontransactional tables within the transaction, the most likely cause for this message is that a table you thought was transactional actually is not. This can happen if you try to create a table using a transactional storage engine that is not supported by your `mysqld` server (or that was disabled with a startup option). If `mysqld` does not support a storage engine, it instead creates the table as a `MyISAM` table, which is nontransactional.

You can check the storage engine for a table by using either of these statements:

```
SHOW TABLE STATUS LIKE 'tbl_name';
SHOW CREATE TABLE tbl_name;
```

See Section 13.7.7.38, “SHOW TABLE STATUS Statement”, and Section 13.7.7.10, “SHOW CREATE TABLE Statement”.

To check which storage engines your `mysqld` server supports, use this statement:

```
SHOW ENGINES;
```

See Section 13.7.7.16, “SHOW ENGINES Statement” for full details.

B.3.4.6 Deleting Rows from Related Tables

If the total length of the `DELETE` statement for `related_table` is more than the default value of the `max_allowed_packet` system variable, you should split it into smaller parts and execute multiple `DELETE` statements. You probably get the fastest `DELETE` by specifying only 100 to 1,000 `related_column` values per statement if the `related_column` is indexed. If the `related_column` isn't indexed, the speed is independent of the number of arguments in the `IN` clause.

B.3.4.7 Solving Problems with No Matching Rows

If you have a complicated query that uses many tables but that returns no rows, you should use the following procedure to find out what is wrong:

1. Test the query with `EXPLAIN` to check whether you can find something that is obviously wrong. See Section 13.8.2, “EXPLAIN Statement”.

2. Select only those columns that are used in the `WHERE` clause.

3. Remove one table at a time from the query until it returns some rows. If the tables are large, it is a good idea to use `LIMIT 10` with the query.

4. Issue a `SELECT` for the column that should have matched a row against the table that was last removed from the query.

5. If you are comparing `FLOAT` or `DOUBLE` columns with numbers that have decimals, you cannot use equality (`=`) comparisons. This problem is common in most computer languages because not all floating-point values can be stored with exact precision. In some cases, changing the `FLOAT` to a `DOUBLE` fixes this. See Section B.3.4.8, “Problems with Floating-Point Values”.

6. If you still cannot figure out what is wrong, create a minimal test that can be run with `mysql test < query.sql` that shows your problems. You can create a test file by dumping the tables with `mysqldump --quick db_name tbl_name_1 ... tbl_name_n > query.sql`. Open the file in an editor, remove some insert lines (if there are more than needed to demonstrate the problem), and add your `SELECT` statement at the end of the file.

   Verify that the test file demonstrates the problem by executing these commands:

   ```
   $> mysqladmin create test2
   $> mysql test2 < query.sql
   ```

   Attach the test file to a bug report, which you can file using the instructions in Section 1.5, “How to Report Bugs or Problems”.

B.3.4.8 Problems with Floating-Point Values

Floating-point numbers sometimes cause confusion because they are approximate and not stored as exact values. A floating-point value as written in an SQL statement may not be the same as the value represented internally. Attempts to treat floating-point values as exact in comparisons may lead to problems. They are also subject to platform or implementation dependencies. The `FLOAT` and `DOUBLE` data types are subject to these issues. For `DECIMAL` columns, MySQL performs operations with a precision of 65 decimal digits, which should solve most common inaccuracy problems.

The following example uses `DOUBLE` to demonstrate how calculations that are done using floating-point operations are subject to floating-point error.

```
mysql> CREATE TABLE t1 (i INT, d1 DOUBLE, d2 DOUBLE);
mysql> INSERT INTO t1 VALUES (1, 101.40, 21.40), (1, -80.00, 0.00),
    -> (2, 0.00, 0.00), (2, -13.20, 0.00), (2, 59.60, 46.40),
    -> (2, 30.40, 30.40), (3, 37.00, 7.40), (3, -29.60, 0.00),
    -> (4, 60.00, 15.40), (4, -10.60, 0.00), (4, -34.00, 0.00),
    -> (5, 33.00, 0.00), (5, -25.80, 0.00), (5, 0.00, 7.20),
    -> (6, 0.00, 0.00), (6, -51.40, 0.00);

mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b
    -> FROM t1 GROUP BY i HAVING a <> b;

+------+-------+------+
| i    | a     | b    |
+------+-------+------+
|    1 |  21.4 | 21.4 |
|    2 |  76.8 | 76.8 |
|    3 |   7.4 |  7.4 |
|    4 |  15.4 | 15.4 |
|    5 |   7.2 |  7.2 |
|    6 | -51.4 |    0 |
+------+-------+------+
```

The result is correct. Although the first five records look like they should not satisfy the comparison (the values of `a` and `b` do not appear to be different), they may do so because the difference between the numbers shows up around the tenth decimal or so, depending on factors such as computer architecture or the compiler version or optimization level. For example, different CPUs may evaluate floating-point numbers differently.

If columns `d1` and `d2` had been defined as `DECIMAL` rather than `DOUBLE`, the result of the `SELECT` query would have contained only one row—the last one shown above.

The correct way to do floating-point number comparison is to first decide on an acceptable tolerance for differences between the numbers and then do the comparison against the tolerance value. For example, if we agree that floating-point numbers should be regarded the same if they are same within a precision of one in ten thousand (0.0001), the comparison should be written to find differences larger than the tolerance value:

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
    -> GROUP BY i HAVING ABS(a - b) > 0.0001;
+------+-------+------+
| i    | a     | b    |
+------+-------+------+
|    6 | -51.4 |    0 |
+------+-------+------+
1 row in set (0.00 sec)
```

Conversely, to get rows where the numbers are the same, the test should find differences within the tolerance value:

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
    -> GROUP BY i HAVING ABS(a - b) <= 0.0001;
+------+------+------+
| i    | a    | b    |
+------+------+------+
|    1 | 21.4 | 21.4 |
|    2 | 76.8 | 76.8 |
|    3 |  7.4 |  7.4 |
|    4 | 15.4 | 15.4 |
|    5 |  7.2 |  7.2 |
+------+------+------+
5 rows in set (0.03 sec)
```

Floating-point values are subject to platform or implementation dependencies. Suppose that you execute the following statements:

```
CREATE TABLE t1(c1 FLOAT(53,0), c2 FLOAT(53,0));
INSERT INTO t1 VALUES('1e+52','-1e+52');
SELECT * FROM t1;
```

On some platforms, the `SELECT` statement returns `inf` and `-inf`. On others, it returns `0` and `-0`.

An implication of the preceding issues is that if you attempt to create a replica by dumping table contents with `mysqldump` on the source and reloading the dump file into the replica, tables containing floating-point columns might differ between the two hosts.

B.3.5 Optimizer-Related Issues

MySQL uses a cost-based optimizer to determine the best way to resolve a query. In many cases, MySQL can calculate the best possible query plan, but sometimes MySQL does not have enough information about the data at hand and has to make “educated” guesses about the data.

For the cases when MySQL does not do the "right" thing, tools that you have available to help MySQL are:

- Use the `EXPLAIN` statement to get information about how MySQL processes a query. To use it, just add the keyword `EXPLAIN` to the front of your `SELECT` statement:

  ```
  mysql> EXPLAIN SELECT * FROM t1, t2 WHERE t1.i = t2.i;
  ```

  `EXPLAIN` is discussed in more detail in Section 13.8.2, “EXPLAIN Statement”.

- Use `ANALYZE TABLE` *`tbl_name`* to update the key distributions for the scanned table. See Section 13.7.3.1, “ANALYZE TABLE Statement”.

- Use `FORCE INDEX` for the scanned table to tell MySQL that table scans are very expensive compared to using the given index:

```
SELECT * FROM t1, t2 FORCE INDEX (index_for_column)
WHERE t1.col_name=t2.col_name;
```

  `USE INDEX` and `IGNORE INDEX` may also be useful. See Section 8.9.4, “Index Hints”.

- Global and table-level `STRAIGHT_JOIN`. See Section 13.2.13, “SELECT Statement”.

- You can tune global or thread-specific system variables. For example, start `mysqld` with the `--max-seeks-for-key=1000` option or use `SET max_seeks_for_key=1000` to tell the optimizer to assume that no key scan causes more than 1,000 key seeks. See Section 5.1.8, “Server System Variables”.

B.3.6 Table Definition-Related Issues

B.3.6.1 Problems with ALTER TABLE

If you get a duplicate-key error when using `ALTER TABLE` to change the character set or collation of a character column, the cause is either that the new column collation maps two keys to the same value or that the table is corrupted. In the latter case, you should run `REPAIR TABLE` on the table. `REPAIR TABLE` works for `MyISAM`, `ARCHIVE`, and `CSV` tables.

If you use `ALTER TABLE` on a transactional table or if you are using Windows, `ALTER TABLE` unlocks the table if you had done a `LOCK TABLE` on it. This is done because `InnoDB` and these operating systems cannot drop a table that is in use.

B.3.6.2 TEMPORARY Table Problems

Temporary tables created with `CREATE TEMPORARY TABLE` have the following limitations:

- `TEMPORARY` tables are supported only by the `InnoDB`, `MEMORY`, `MyISAM`, and `MERGE` storage engines.

- Temporary tables are not supported for NDB Cluster.

- The `SHOW TABLES` statement does not list `TEMPORARY` tables.

- To rename `TEMPORARY` tables, `RENAME TABLE` does not work. Use `ALTER TABLE` instead:

```
ALTER TABLE old_name RENAME new_name;
```

- You cannot refer to a `TEMPORARY` table more than once in the same query. For example, the following does not work:

```
SELECT * FROM temp_table JOIN temp_table AS t2;
```

  The statement produces this error:

```
ERROR 1137: Can't reopen table: 'temp_table'
```

  You can work around this issue if your query permits use of a common table expression (CTE) rather than a `TEMPORARY` table. For example, this fails with the `Can't reopen table` error:

```
CREATE TEMPORARY TABLE t SELECT 1 AS col_a, 2 AS col_b;
SELECT * FROM t AS t1 JOIN t AS t2;
```

  To avoid the error, use a `WITH` clause that defines a CTE, rather than the `TEMPORARY` table:

```
WITH cte AS (SELECT 1 AS col_a, 2 AS col_b)
SELECT * FROM cte AS t1 JOIN cte AS t2;
```

- The `Can't reopen table` error also occurs if you refer to a temporary table multiple times in a stored function under different aliases, even if the references occur in different statements within the function. It may occur for temporary tables created outside stored functions and referred to across multiple calling and callee functions.
- If a `TEMPORARY` is created with the same name as an existing non-`TEMPORARY` table, the non-`TEMPORARY` table is hidden until the `TEMPORARY` table is dropped, even if the tables use different storage engines.
- There are known issues in using temporary tables with replication. See Section 17.5.1.31, “Replication and Temporary Tables”, for more information.

B.3.7 Known Issues in MySQL

This section lists known issues in recent versions of MySQL.

For information about platform-specific issues, see the installation and debugging instructions in Section 2.1, “General Installation Guidance”, and Section 5.9, “Debugging MySQL”.

The following problems are known:

- Subquery optimization for `IN` is not as effective as for `=`.
- Even if you use `lower_case_table_names=2` (which enables MySQL to remember the case used for databases and table names), MySQL does not remember the case used for database names for the function `DATABASE()` or within the various logs (on case-insensitive systems).
- Dropping a `FOREIGN KEY` constraint does not work in replication because the constraint may have another name on the replica.
- `REPLACE` (and `LOAD DATA` with the `REPLACE` option) does not trigger `ON DELETE CASCADE`.
- `DISTINCT` with `ORDER BY` does not work inside `GROUP_CONCAT()` if you do not use all and only those columns that are in the `DISTINCT` list.
- When inserting a big integer value (between 2^{63} and $2^{64}-1$) into a decimal or string column, it is inserted as a negative value because the number is evaluated in signed integer context.
- With statement-based binary logging, the source server writes the executed queries to the binary log. This is a very fast, compact, and efficient logging method that works perfectly in most cases. However, it is possible for the data on the source and replica to become different if a query is designed in such a way that the data modification is nondeterministic (generally not a recommended practice, even outside of replication).

  For example:

  - `CREATE TABLE ... SELECT` or `INSERT ... SELECT` statements that insert zero or `NULL` values into an `AUTO_INCREMENT` column.
  - `DELETE` if you are deleting rows from a table that has foreign keys with `ON DELETE CASCADE` properties.
  - `REPLACE ... SELECT`, `INSERT IGNORE ... SELECT` if you have duplicate key values in the inserted data.

  If and only if the preceding queries have no `ORDER BY` clause guaranteeing a deterministic order.

  For example, for `INSERT ... SELECT` with no `ORDER BY`, the `SELECT` may return rows in a different order (which results in a row having different ranks, hence getting a different number in the `AUTO_INCREMENT` column), depending on the choices made by the optimizers on the source and replica.

A query is optimized differently on the source and replica only if:

- The table is stored using a different storage engine on the source than on the replica. (It is possible to use different storage engines on the source and replica. For example, you can use `InnoDB` on the source, but `MyISAM` on the replica if the replica has less available disk space.)
- MySQL buffer sizes (`key_buffer_size`, and so on) are different on the source and replica.
- The source and replica run different MySQL versions, and the optimizer code differs between these versions.

This problem may also affect database restoration using `mysqlbinlog|mysql`.

The easiest way to avoid this problem is to add an `ORDER BY` clause to the aforementioned nondeterministic queries to ensure that the rows are always stored or modified in the same order. Using row-based or mixed logging format also avoids the problem.

- Log file names are based on the server host name if you do not specify a file name with the startup option. To retain the same log file names if you change your host name to something else, you must explicitly use options such as `--log-bin=`*`old_host_name`*`-bin`. See Section 5.1.7, “Server Command Options”. Alternatively, rename the old files to reflect your host name change. If these are binary logs, you must edit the binary log index file and fix the binary log file names there as well. (The same is true for the relay logs on a replica.)
- `mysqlbinlog` does not delete temporary files left after a `LOAD DATA` statement. See Section 4.6.9, “mysqlbinlog — Utility for Processing Binary Log Files”.
- `RENAME` does not work with `TEMPORARY` tables or tables used in a `MERGE` table.
- When using `SET CHARACTER SET`, you cannot use translated characters in database, table, and column names.
- Prior to MySQL 8.0.17, you cannot use `_` or `%` with `ESCAPE` in `LIKE ... ESCAPE`.
- The server uses only the first `max_sort_length` bytes when comparing data values. This means that values cannot reliably be used in `GROUP BY`, `ORDER BY`, or `DISTINCT` if they differ only after the first `max_sort_length` bytes. To work around this, increase the variable value. The default value of `max_sort_length` is 1024 and can be changed at server startup time or at runtime.
- Numeric calculations are done with `BIGINT` or `DOUBLE` (both are normally 64 bits long). Which precision you get depends on the function. The general rule is that bit functions are performed with `BIGINT` precision, `IF()` and `ELT()` with `BIGINT` or `DOUBLE` precision, and the rest with `DOUBLE` precision. You should try to avoid using unsigned long long values if they resolve to be larger than 63 bits (9223372036854775807) for anything other than bit fields.
- You can have up to 255 `ENUM` and `SET` columns in one table.
- In `MIN()`, `MAX()`, and other aggregate functions, MySQL currently compares `ENUM` and `SET` columns by their string value rather than by the string's relative position in the set.
- In an `UPDATE` statement, columns are updated from left to right. If you refer to an updated column, you get the updated value instead of the original value. For example, the following statement increments `KEY` by `2`, **not** `1`:

```
mysql> UPDATE tbl_name SET KEY=KEY+1,KEY=KEY+1;
```

- You can refer to multiple temporary tables in the same query, but you cannot refer to any given temporary table more than once. For example, the following does not work:

```
mysql> SELECT * FROM temp_table, temp_table AS t2;
ERROR 1137: Can't reopen table: 'temp_table'
```

- The optimizer may handle `DISTINCT` differently when you are using “hidden” columns in a join than when you are not. In a join, hidden columns are counted as part of the result (even if they are not shown), whereas in normal queries, hidden columns do not participate in the `DISTINCT` comparison.

  An example of this is:

  ```
  SELECT DISTINCT mp3id FROM band_downloads
         WHERE userid = 9 ORDER BY id DESC;
  ```

  and

  ```
  SELECT DISTINCT band_downloads.mp3id
         FROM band_downloads,band_mp3
         WHERE band_downloads.userid = 9
         AND band_mp3.id = band_downloads.mp3id
         ORDER BY band_downloads.id DESC;
  ```

  In the second case, you may get two identical rows in the result set (because the values in the hidden `id` column may differ).

  Note that this happens only for queries that do not have the `ORDER BY` columns in the result.

- If you execute a `PROCEDURE` on a query that returns an empty set, in some cases the `PROCEDURE` does not transform the columns.

- Creation of a table of type `MERGE` does not check whether the underlying tables are compatible types.

- If you use `ALTER TABLE` to add a `UNIQUE` index to a table used in a `MERGE` table and then add a normal index on the `MERGE` table, the key order is different for the tables if there was an old, non-`UNIQUE` key in the table. This is because `ALTER TABLE` puts `UNIQUE` indexes before normal indexes to be able to detect duplicate keys as early as possible.

Appendix C Indexes

Table of Contents

General Index

Symbols

A