The active `ssl_crl` value in the TSL context that the server uses for new connections. For notes about the relationship between this status variable and its corresponding system variable, see the description of `Current_tls_ca`.

This variable was added in MySQL 8.0.16.

Note

When you reload the TLS context, OpenSSL reloads the file containing the CRL (certificate revocation list) as part of the process. If the CRL file is large, the server allocates a large chunk of memory (ten times the file size), which is doubled while the new instance is being loaded and the old one has not yet been released. The process resident memory is not immediately reduced after a large allocation is freed, so if you issue the `ALTER INSTANCE RELOAD TLS` statement repeatedly with a large CRL file, the process resident memory usage may grow as a result of this.

- `Current_tls_crlpath`

  The active `ssl_crlpath` value in the TSL context that the server uses for new connections. For notes about the relationship between this status variable and its corresponding system variable, see the description of `Current_tls_ca`.

  This variable was added in MySQL 8.0.16.

- `Current_tls_key`

  The active `ssl_key` value in the TSL context that the server uses for new connections. For notes about the relationship between this status variable and its corresponding system variable, see the description of `Current_tls_ca`.

  This variable was added in MySQL 8.0.16.

- `Current_tls_version`

  The active `tls_version` value in the TSL context that the server uses for new connections. For notes about the relationship between this status variable and its corresponding system variable, see the description of `Current_tls_ca`.

  This variable was added in MySQL 8.0.16.

- `Delayed_errors`

  This status variable is deprecated (because `DELAYED` inserts are not supported); expect it to be removed in a future release.

- `Delayed_insert_threads`

  This status variable is deprecated (because `DELAYED` inserts are not supported); expect it to be removed in a future release.

- `Delayed_writes`

  This status variable is deprecated (because `DELAYED` inserts are not supported); expect it to be removed in a future release.

- `dragnet.Status`

  The result of the most recent assignment to the `dragnet.log_error_filter_rules` system variable, empty if no such assignment has occurred.

  This variable was added in MySQL 8.0.12.

- `Error_log_buffered_bytes`

  The number of bytes currently used in the Performance Schema `error_log` table. It is possible for the value to decrease, for example, if a new event cannot fit until discarding an old event, but the new event is smaller than the old one.

  This variable was added in MySQL 8.0.22.

- `Error_log_buffered_events`

  The number of events currently present in the Performance Schema `error_log` table. As with `Error_log_buffered_bytes`, it is possible for the value to decrease.

  This variable was added in MySQL 8.0.22.

- `Error_log_expired_events`

  The number of events discarded from the Performance Schema `error_log` table to make room for new events.

  This variable was added in MySQL 8.0.22.

- `Error_log_latest_write`

  The time of the last write to the Performance Schema `error_log` table.

  This variable was added in MySQL 8.0.22.

- `Flush_commands`

  The number of times the server flushes tables, whether because a user executed a `FLUSH TABLES` statement or due to internal server operation. It is also incremented by receipt of a `COM_REFRESH` packet. This is in contrast to `Com_flush`, which indicates how many `FLUSH` statements have been executed, whether `FLUSH TABLES`, `FLUSH LOGS`, and so forth.

- `Global_connection_memory`

  The memory used by all user connections to the server. Memory used by system threads or by the MySQL root account is included in the total, but such threads or users are not subject to disconnection due to memory usage. This memory is not calculated unless `global_connection_memory_tracking` is enabled (disabled by default). The Performance Schema must also be enabled.

  You can control (indirectly) the frequency with which this variable is updated by setting `connection_memory_chunk_size`.

  The `Global_connection_memory` status variable was introduced in MySQL 8.0.28.

- `group_replication_primary_member`

  Shows the primary member's UUID when the group is operating in single-primary mode. If the group is operating in multi-primary mode, shows an empty string.

  The `group_replication_primary_member` status variable is deprecated and is scheduled to be removed in a future version.

- `Handler_commit`

  The number of internal `COMMIT` statements.

- `Handler_delete`

  The number of times that rows have been deleted from tables.

- `Handler_external_lock`

  The server increments this variable for each call to its `external_lock()` function, which generally occurs at the beginning and end of access to a table instance. There might be differences among storage engines. This variable can be used, for example, to discover for a statement that accesses a partitioned table how many partitions were pruned before locking occurred: Check how much the counter increased for the statement, subtract 2 (2 calls for the table itself), then divide by 2 to get the number of partitions locked.

- `Handler_mrr_init`

  The number of times the server uses a storage engine's own Multi-Range Read implementation for table access.

- `Handler_prepare`

  A counter for the prepare phase of two-phase commit operations.

- `Handler_read_first`

  The number of times the first entry in an index was read. If this value is high, it suggests that the server is doing a lot of full index scans (for example, `SELECT col1 FROM foo`, assuming that `col1` is indexed).

- `Handler_read_key`

  The number of requests to read a row based on a key. If this value is high, it is a good indication that your tables are properly indexed for your queries.

- `Handler_read_last`

  The number of requests to read the last key in an index. With `ORDER BY`, the server issues a first-key request followed by several next-key requests, whereas with `ORDER BY DESC`, the server issues a last-key request followed by several previous-key requests.

- `Handler_read_next`

  The number of requests to read the next row in key order. This value is incremented if you are querying an index column with a range constraint or if you are doing an index scan.

- `Handler_read_prev`

  The number of requests to read the previous row in key order. This read method is mainly used to optimize `ORDER BY ... DESC`.

- `Handler_read_rnd`

  The number of requests to read a row based on a fixed position. This value is high if you are doing a lot of queries that require sorting of the result. You probably have a lot of queries that require MySQL to scan entire tables or you have joins that do not use keys properly.

- `Handler_read_rnd_next`

  The number of requests to read the next row in the data file. This value is high if you are doing a lot of table scans. Generally this suggests that your tables are not properly indexed or that your queries are not written to take advantage of the indexes you have.

- `Handler_rollback`

  The number of requests for a storage engine to perform a rollback operation.

- `Handler_savepoint`

  The number of requests for a storage engine to place a savepoint.

- `Handler_savepoint_rollback`

  The number of requests for a storage engine to roll back to a savepoint.

- `Handler_update`

  The number of requests to update a row in a table.

- `Handler_write`

  The number of requests to insert a row in a table.

- `Innodb_buffer_pool_dump_status`

  The progress of an operation to record the pages held in the `InnoDB` buffer pool, triggered by the setting of `innodb_buffer_pool_dump_at_shutdown` or `innodb_buffer_pool_dump_now`.

  For related information and examples, see Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”.

- `Innodb_buffer_pool_load_status`

  The progress of an operation to warm up the `InnoDB` buffer pool by reading in a set of pages corresponding to an earlier point in time, triggered by the setting of `innodb_buffer_pool_load_at_startup` or `innodb_buffer_pool_load_now`. If the operation introduces too much overhead, you can cancel it by setting `innodb_buffer_pool_load_abort`.

  For related information and examples, see Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”.

- `Innodb_buffer_pool_bytes_data`

  The total number of bytes in the `InnoDB` buffer pool containing data. The number includes both dirty and clean pages. For more accurate memory usage calculations than with `Innodb_buffer_pool_pages_data`, when compressed tables cause the buffer pool to hold pages of different sizes.

- `Innodb_buffer_pool_pages_data`

  The number of pages in the `InnoDB` buffer pool containing data. The number includes both dirty and clean pages. When using compressed tables, the reported `Innodb_buffer_pool_pages_data` value may be larger than `Innodb_buffer_pool_pages_total` (Bug #59550).

- `Innodb_buffer_pool_bytes_dirty`

  The total current number of bytes held in dirty pages in the `InnoDB` buffer pool. For more accurate memory usage calculations than with `Innodb_buffer_pool_pages_dirty`, when compressed tables cause the buffer pool to hold pages of different sizes.

- `Innodb_buffer_pool_pages_dirty`

  The current number of dirty pages in the `InnoDB` buffer pool.

- `Innodb_buffer_pool_pages_flushed`

  The number of requests to flush pages from the `InnoDB` buffer pool.

- `Innodb_buffer_pool_pages_free`

  The number of free pages in the `InnoDB` buffer pool.

- `Innodb_buffer_pool_pages_latched`

  The number of latched pages in the `InnoDB` buffer pool. These are pages currently being read or written, or that cannot be flushed or removed for some other reason. Calculation of this variable is expensive, so it is available only when the `UNIV_DEBUG` system is defined at server build time.

- `Innodb_buffer_pool_pages_misc`

  The number of pages in the `InnoDB` buffer pool that are busy because they have been allocated for administrative overhead, such as row locks or the adaptive hash index. This value can also be calculated as `Innodb_buffer_pool_pages_total` − `Innodb_buffer_pool_pages_free` − `Innodb_buffer_pool_pages_data`. When using compressed tables, `Innodb_buffer_pool_pages_misc` may report an out-of-bounds value (Bug #59550).

- `Innodb_buffer_pool_pages_total`

  The total size of the `InnoDB` buffer pool, in pages. When using compressed tables, the reported `Innodb_buffer_pool_pages_data` value may be larger than `Innodb_buffer_pool_pages_total` (Bug #59550)

- `Innodb_buffer_pool_read_ahead`

  The number of pages read into the `InnoDB` buffer pool by the read-ahead background thread.

- `Innodb_buffer_pool_read_ahead_evicted`

  The number of pages read into the `InnoDB` buffer pool by the read-ahead background thread that were subsequently evicted without having been accessed by queries.

- `Innodb_buffer_pool_read_ahead_rnd`

  The number of “random” read-aheads initiated by `InnoDB`. This happens when a query scans a large portion of a table but in random order.

- `Innodb_buffer_pool_read_requests`

  The number of logical read requests.

- `Innodb_buffer_pool_reads`

  The number of logical reads that `InnoDB` could not satisfy from the buffer pool, and had to read directly from disk.

- `Innodb_buffer_pool_resize_status`

  The status of an operation to resize the `InnoDB` buffer pool dynamically, triggered by setting the `innodb_buffer_pool_size` parameter dynamically. The `innodb_buffer_pool_size` parameter is dynamic, which allows you to resize the buffer pool without restarting the server. See Configuring InnoDB Buffer Pool Size Online for related information.

- `Innodb_buffer_pool_resize_status_code`

Reports status codes for tracking online buffer pool resizing operations. Each status code represents a stage in a resizing operation. Status codes include:

- 0: No Resize operation in progress
- 1: Starting Resize
- 2: Disabling AHI (Adaptive Hash Index)
- 3: Withdrawing Blocks
- 4: Acquiring Global Lock
- 5: Resizing Pool
- 6: Resizing Hash
- 7: Resizing Failed

You can use this status variable in conjunction with `Innodb_buffer_pool_resize_status_progress` to track the progress of each stage of a resizing operation. The `Innodb_buffer_pool_resize_status_progress` variable reports a percentage value indicating the progress of the current stage.

For more information, see Monitoring Online Buffer Pool Resizing Progress.

- `Innodb_buffer_pool_resize_status_progress`

  Reports a percentage value indicating the progress of the current stage of an online buffer pool resizing operation. This variable is used in conjunction with `Innodb_buffer_pool_resize_status_code`, which reports a status code indicating the current stage of an online buffer pool resizing operation.

  The percentage value is updated after each buffer pool instance is processed. As the status code (reported by `Innodb_buffer_pool_resize_status_code`) changes from one status to another, the percentage value is reset to 0.

  For related information, see Monitoring Online Buffer Pool Resizing Progress.

- `Innodb_buffer_pool_wait_free`

  Normally, writes to the `InnoDB` buffer pool happen in the background. When `InnoDB` needs to read or create a page and no clean pages are available, `InnoDB` flushes some dirty pages first and waits for that operation to finish. This counter counts instances of these waits. If `innodb_buffer_pool_size` has been set properly, this value should be small.

- `Innodb_buffer_pool_write_requests`

  The number of writes done to the `InnoDB` buffer pool.

- `Innodb_data_fsyncs`

  The number of `fsync()` operations so far. The frequency of `fsync()` calls is influenced by the setting of the `innodb_flush_method` configuration option.

  Counts the number of `fdatasync()` operations if `innodb_use_fdatasync` is enabled.

- `Innodb_data_pending_fsyncs`

  The current number of pending `fsync()` operations. The frequency of `fsync()` calls is influenced by the setting of the `innodb_flush_method` configuration option.

- `Innodb_data_pending_reads`

  The current number of pending reads.

- `Innodb_data_pending_writes`

  The current number of pending writes.

- `Innodb_data_read`

  The amount of data read since the server was started (in bytes).

- `Innodb_data_reads`

  The total number of data reads (OS file reads).

- `Innodb_data_writes`

  The total number of data writes.

- `Innodb_data_written`

  The amount of data written so far, in bytes.

- `Innodb_dblwr_pages_written`

  The number of pages that have been written to the doublewrite buffer. See Section 15.11.1, “InnoDB Disk I/O”.

- `Innodb_dblwr_writes`

  The number of doublewrite operations that have been performed. See Section 15.11.1, “InnoDB Disk I/O”.

- `Innodb_have_atomic_builtins`

  Indicates whether the server was built with atomic instructions.

- `Innodb_log_waits`

  The number of times that the log buffer was too small and a wait was required for it to be flushed before continuing.

- `Innodb_log_write_requests`

  The number of write requests for the `InnoDB` redo log.

- `Innodb_log_writes`

  The number of physical writes to the `InnoDB` redo log file.

- `Innodb_num_open_files`

  The number of files `InnoDB` currently holds open.

- `Innodb_os_log_fsyncs`

  The number of `fsync()` writes done to the `InnoDB` redo log files.

- `Innodb_os_log_pending_fsyncs`

  The number of pending `fsync()` operations for the `InnoDB` redo log files.

- `Innodb_os_log_pending_writes`

The number of pending writes to the InnoDB redo log files.

- Innodb_os_log_written

  The number of bytes written to the InnoDB redo log files.

- Innodb_page_size

  InnoDB page size (default 16KB). Many values are counted in pages; the page size enables them to be easily converted to bytes.

- Innodb_pages_created

  The number of pages created by operations on InnoDB tables.

- Innodb_pages_read

  The number of pages read from the InnoDB buffer pool by operations on InnoDB tables.

- Innodb_pages_written

  The number of pages written by operations on InnoDB tables.

- Innodb_redo_log_enabled

  Whether redo logging is enabled or disabled. See Disabling Redo Logging.

  This variable was added in MySQL 8.0.21.

- Innodb_redo_log_capacity_resized

  The total redo log capacity for all redo log files, in bytes, after the last completed capacity resize operation. The value includes ordinary and spare redo log files.

  If there is no pending resize down operation, Innodb_redo_log_capacity_resized should be equal to the innodb_redo_log_capacity setting. Resize up operations are instantaneous.

  For related information, see Section 15.6.5, “Redo Log”.

  This variable was added in MySQL 8.0.30.

- Innodb_redo_log_checkpoint_lsn

  The redo log checkpoint LSN. For related information, see Section 15.6.5, “Redo Log”.

  This variable was added in MySQL 8.0.30.

- Innodb_redo_log_current_lsn

  The current LSN represents the last written position in the redo log buffer. InnoDB writes data to the redo log buffer inside the MySQL process before requesting that the operating system write the data to the current redo log file. For related information, see Section 15.6.5, “Redo Log”.

  This variable was added in MySQL 8.0.30.

- Innodb_redo_log_flushed_to_disk_lsn

  The flushed-to-disk LSN. InnoDB first writes data to the redo log and then requests that the operating system flush the data to disk. The flushed-to-disk LSN represents the last position in the redo log that InnoDB knows has been flushed to disk. For related information, see Section 15.6.5, “Redo Log”.

  This variable was added in MySQL 8.0.30.

- `Innodb_redo_log_logical_size`

  A data size value, in bytes, representing the LSN range containing in-use redo log data, spanning from the oldest block required by redo log consumers to the latest written block. For related information, see Section 15.6.5, “Redo Log”.

  This variable was added in MySQL 8.0.30.

- `Innodb_redo_log_physical_size`

  The amount of disk space in bytes currently consumed by all redo log files on disk, excluding spare redo log files. For related information, see Section 15.6.5, “Redo Log”.

  This variable was added in MySQL 8.0.30.

- `Innodb_redo_log_read_only`

  Whether the redo log is read-only.

  This variable was added in MySQL 8.0.30.

- `Innodb_redo_log_resize_status`

  The redo log resize status indicating the current state of the redo log capacity resize mechanism. Possible values include:

  - `OK`: There are no issues and no pending redo log capacity resize operations.
  - `Resizing down`: A resize down operation is in progress.

  A resize up operation is instantaneous and therefore has no pending status.

  This variable was added in MySQL 8.0.30.

- `Innodb_redo_log_uuid`

  The redo log UUID.

  This variable was added in MySQL 8.0.30.

- `Innodb_row_lock_current_waits`

  The number of row locks currently waited for by operations on `InnoDB` tables.

- `Innodb_row_lock_time`

  The total time spent in acquiring row locks for `InnoDB` tables, in milliseconds.

- `Innodb_row_lock_time_avg`

  The average time to acquire a row lock for `InnoDB` tables, in milliseconds.

- `Innodb_row_lock_time_max`

  The maximum time to acquire a row lock for `InnoDB` tables, in milliseconds.

- `Innodb_row_lock_waits`

  The number of times operations on `InnoDB` tables had to wait for a row lock.

- `Innodb_rows_deleted`

  The number of rows deleted from `InnoDB` tables.

- `Innodb_rows_inserted`

  The number of rows inserted into `InnoDB` tables.

- `Innodb_rows_read`

  The number of rows read from `InnoDB` tables.

- `Innodb_rows_updated`

  The number of rows updated in `InnoDB` tables.

- `Innodb_system_rows_deleted`

  The number of rows deleted from `InnoDB` tables belonging to system-created schemas.

- `Innodb_system_rows_inserted`

  The number of rows inserted into `InnoDB` tables belonging to system-created schemas.

- `Innodb_system_rows_read`

  The number of rows read from `InnoDB` tables belonging to system-created schemas.

- `Innodb_truncated_status_writes`

  The number of times output from the `SHOW ENGINE INNODB STATUS` statement has been truncated.

- `Innodb_undo_tablespaces_active`

  The number of active undo tablespaces. Includes both implicit (`InnoDB`-created) and explicit (user-created) undo tablespaces. For information about undo tablespaces, see Section 15.6.3.4, “Undo Tablespaces”.

- `Innodb_undo_tablespaces_explicit`

  The number of user-created undo tablespaces. For information about undo tablespaces, see Section 15.6.3.4, “Undo Tablespaces”.

- `Innodb_undo_tablespaces_implicit`

  The number of undo tablespaces created by `InnoDB`. Two default undo tablespaces are created by `InnoDB` when the MySQL instance is initialized. For information about undo tablespaces, see Section 15.6.3.4, “Undo Tablespaces”.

- `Innodb_undo_tablespaces_total`

  The total number of undo tablespaces. Includes both implicit (`InnoDB`-created) and explicit (user-created) undo tablespaces, active and inactive. For information about undo tablespaces, see Section 15.6.3.4, “Undo Tablespaces”.

- `Key_blocks_not_flushed`

  The number of key blocks in the `MyISAM` key cache that have changed but have not yet been flushed to disk.

- `Key_blocks_unused`

  The number of unused blocks in the `MyISAM` key cache. You can use this value to determine how much of the key cache is in use; see the discussion of `key_buffer_size` in Section 5.1.8, “Server System Variables”.

- `Key_blocks_used`

The number of used blocks in the MyISAM key cache. This value is a high-water mark that indicates the maximum number of blocks that have ever been in use at one time.

- Key_read_requests

  The number of requests to read a key block from the MyISAM key cache.

- Key_reads

  The number of physical reads of a key block from disk into the MyISAM key cache. If Key_reads is large, then your key_buffer_size value is probably too small. The cache miss rate can be calculated as Key_reads/Key_read_requests.

- Key_write_requests

  The number of requests to write a key block to the MyISAM key cache.

- Key_writes

  The number of physical writes of a key block from the MyISAM key cache to disk.

- Last_query_cost

  The total cost of the last compiled query as computed by the query optimizer. This is useful for comparing the cost of different query plans for the same query. The default value of 0 means that no query has been compiled yet. The default value is 0. Last_query_cost has session scope.

  In MySQL 8.0.16 and later, this variable shows the cost of queries that have multiple query blocks, summing the cost estimates of each query block, estimating how many times non-cacheable subqueries are executed, and multiplying the cost of those query blocks by the number of subquery executions. (Bug #92766, Bug #28786951) Prior to MySQL 8.0.16, Last_query_cost was computed accurately only for simple, “flat” queries, but not for complex queries such as those containing subqueries or UNION. (For the latter, the value was set to 0.)

- Last_query_partial_plans

  The number of iterations the query optimizer made in execution plan construction for the previous query.

  Last_query_partial_plans has session scope.

- Locked_connects

  The number of attempts to connect to locked user accounts. For information about account locking and unlocking, see Section 6.2.20, “Account Locking”.

- Max_execution_time_exceeded

  The number of SELECT statements for which the execution timeout was exceeded.

- Max_execution_time_set

  The number of SELECT statements for which a nonzero execution timeout was set. This includes statements that include a nonzero MAX_EXECUTION_TIME optimizer hint, and statements that include no such hint but execute while the timeout indicated by the max_execution_time system variable is nonzero.

- Max_execution_time_set_failed

  The number of SELECT statements for which the attempt to set an execution timeout failed.

- Max_used_connections

The maximum number of connections that have been in use simultaneously since the server started.

- `Max_used_connections_time`

  The time at which `Max_used_connections` reached its current value.

- `Not_flushed_delayed_rows`

  This status variable is deprecated (because `DELAYED` inserts are not supported); expect it to be removed in a future release.

- `mecab_charset`

  The character set currently used by the MeCab full-text parser plugin. For related information, see Section 12.10.9, “MeCab Full-Text Parser Plugin”.

- `Ongoing_anonymous_transaction_count`

  Shows the number of ongoing transactions which have been marked as anonymous. This can be used to ensure that no further transactions are waiting to be processed.

- `Ongoing_anonymous_gtid_violating_transaction_count`

  This status variable is only available in debug builds. Shows the number of ongoing transactions which use `gtid_next=ANONYMOUS` and that violate GTID consistency.

- `Ongoing_automatic_gtid_violating_transaction_count`

  This status variable is only available in debug builds. Shows the number of ongoing transactions which use `gtid_next=AUTOMATIC` and that violate GTID consistency.

- `Open_files`

  The number of files that are open. This count includes regular files opened by the server. It does not include other types of files such as sockets or pipes. Also, the count does not include files that storage engines open using their own internal functions rather than asking the server level to do so.

- `Open_streams`

  The number of streams that are open (used mainly for logging).

- `Open_table_definitions`

  The number of cached table definitions.

- `Open_tables`

  The number of tables that are open.

- `Opened_files`

  The number of files that have been opened with `my_open()` (a `mysys` library function). Parts of the server that open files without using this function do not increment the count.

- `Opened_table_definitions`

  The number of table definitions that have been cached.

- `Opened_tables`

  The number of tables that have been opened. If `Opened_tables` is big, your `table_open_cache` value is probably too small.

- `Performance_schema_`*`xxx`*

Performance Schema status variables are listed in Section 27.16, “Performance Schema Status Variables”. These variables provide information about instrumentation that could not be loaded or created due to memory constraints.

- `Prepared_stmt_count`

  The current number of prepared statements. (The maximum number of statements is given by the `max_prepared_stmt_count` system variable.)

- `Queries`

  The number of statements executed by the server. This variable includes statements executed within stored programs, unlike the `Questions` variable. It does not count `COM_PING` or `COM_STATISTICS` commands.

  The discussion at the beginning of this section indicates how to relate this statement-counting status variable to other such variables.

- `Questions`

  The number of statements executed by the server. This includes only statements sent to the server by clients and not statements executed within stored programs, unlike the `Queries` variable. This variable does not count `COM_PING`, `COM_STATISTICS`, `COM_STMT_PREPARE`, `COM_STMT_CLOSE`, or `COM_STMT_RESET` commands.

  The discussion at the beginning of this section indicates how to relate this statement-counting status variable to other such variables.

- `Replica_open_temp_tables`

  From MySQL 8.0.26, use `Replica_open_temp_tables` in place of `Slave_open_temp_tables`, which is deprecated from that release. In releases before MySQL 8.0.26, use `Slave_open_temp_tables`.

  `Replica_open_temp_tables` shows the number of temporary tables that the replication SQL thread currently has open. If the value is greater than zero, it is not safe to shut down the replica; see Section 17.5.1.31, “Replication and Temporary Tables”. This variable reports the total count of open temporary tables for *all* replication channels.

- `Replica_rows_last_search_algorithm_used`

  From MySQL 8.0.26, use `Replica_rows_last_search_algorithm_used` in place of `Slave_rows_last_search_algorithm_used`, which is deprecated from that release. In releases before MySQL 8.0.26, use `Slave_rows_last_search_algorithm_used`.

  `Replica_rows_last_search_algorithm_used` shows the search algorithm that was most recently used by this replica to locate rows for row-based replication. The result shows whether the replica used indexes, a table scan, or hashing as the search algorithm for the last transaction executed on any channel.

  The method used depends on the setting for the `slave_rows_search_algorithms` system variable (which is now deprecated), and the keys that are available on the relevant table.

  This variable is available only for debug builds of MySQL.

- `Resource_group_supported`

  Indicates whether the resource group feature is supported.

  On some platforms or MySQL server configurations, resource groups are unavailable or have limitations. In particular, Linux systems might require a manual step for some installation methods. For details, see Resource Group Restrictions.

- `Rpl_semi_sync_master_clients`

  The number of semisynchronous replicas.

  `Rpl_semi_sync_master_clients` is available when the `rpl_semi_sync_master` (`semisync_master.so` library) plugin was installed on the replica to set up semisynchronous replication. If the `rpl_semi_sync_source` plugin (`semisync_source.so` library) was installed, `Rpl_semi_sync_source_clients` is available instead.

- `Rpl_semi_sync_master_net_avg_wait_time`

  The average time in microseconds the source waited for a replica reply. This variable is always `0`, and is deprecated; expect it to be removed in a future version.

  `Rpl_semi_sync_master_net_avg_wait_time` is available when the `rpl_semi_sync_master` (`semisync_master.so` library) plugin was installed on the replica to set up semisynchronous replication. If the `rpl_semi_sync_source` plugin (`semisync_source.so` library) was installed, `Rpl_semi_sync_source_net_avg_wait_time` is available instead.

- `Rpl_semi_sync_master_net_wait_time`

  The total time in microseconds the source waited for replica replies. This variable is always `0`, and is deprecated; expect it to be removed in a future version.

  `Rpl_semi_sync_master_net_wait_time` is available when the `rpl_semi_sync_master` (`semisync_master.so` library) plugin was installed on the replica to set up semisynchronous replication. If the `rpl_semi_sync_source` plugin (`semisync_source.so` library) was installed, `Rpl_semi_sync_source_net_wait_time` is available instead.

- `Rpl_semi_sync_master_net_waits`

  The total number of times the source waited for replica replies.

  `Rpl_semi_sync_master_net_waits` is available when the `rpl_semi_sync_master` (`semisync_master.so` library) plugin was installed on the replica to set up semisynchronous replication. If the `rpl_semi_sync_source` plugin (`semisync_source.so` library) was installed, `Rpl_semi_sync_source_net_waits` is available instead.

- `Rpl_semi_sync_master_no_times`

  The number of times the source turned off semisynchronous replication.

  `Rpl_semi_sync_master_no_times` is available when the `rpl_semi_sync_master` (`semisync_master.so` library) plugin was installed on the replica to set up semisynchronous replication. If the `rpl_semi_sync_source` plugin (`semisync_source.so` library) was installed, `Rpl_semi_sync_source_no_times` is available instead.

- `Rpl_semi_sync_master_no_tx`

  The number of commits that were not acknowledged successfully by a replica.

  `Rpl_semi_sync_master_no_tx` is available when the `rpl_semi_sync_master` (`semisync_master.so` library) plugin was installed on the replica to set up semisynchronous

replication. If the `rpl_semi_sync_source` plugin (`semisync_source.so` library) was installed, `Rpl_semi_sync_source_no_tx` is available instead.

- `Rpl_semi_sync_master_status`

  Whether semisynchronous replication currently is operational on the source. The value is `ON` if the plugin has been enabled and a commit acknowledgment has occurred. It is `OFF` if the plugin is not enabled or the source has fallen back to asynchronous replication due to commit acknowledgment timeout.

  `Rpl_semi_sync_master_status` is available when the `rpl_semi_sync_master` (`semisync_master.so` library) plugin was installed on the replica to set up semisynchronous replication. If the `rpl_semi_sync_source` plugin (`semisync_source.so` library) was installed, `Rpl_semi_sync_source_status` is available instead.

- `Rpl_semi_sync_master_timefunc_failures`

  The number of times the source failed when calling time functions such as `gettimeofday()`.

  `Rpl_semi_sync_master_timefunc_failures` is available when the `rpl_semi_sync_master` (`semisync_master.so` library) plugin was installed on the replica to set up semisynchronous replication. If the `rpl_semi_sync_source` plugin (`semisync_source.so` library) was installed, `Rpl_semi_sync_source_timefunc_failures` is available instead.

- `Rpl_semi_sync_master_tx_avg_wait_time`

  The average time in microseconds the source waited for each transaction.

  `Rpl_semi_sync_master_tx_avg_wait_time` is available when the `rpl_semi_sync_master` (`semisync_master.so` library) plugin was installed on the replica to set up semisynchronous replication. If the `rpl_semi_sync_source` plugin (`semisync_source.so` library) was installed, `Rpl_semi_sync_source_tx_avg_wait_time` is available instead.

- `Rpl_semi_sync_master_tx_wait_time`

  The total time in microseconds the source waited for transactions.

  `Rpl_semi_sync_master_tx_wait_time` is available when the `rpl_semi_sync_master` (`semisync_master.so` library) plugin was installed on the replica to set up semisynchronous replication. If the `rpl_semi_sync_source` plugin (`semisync_source.so` library) was installed, `Rpl_semi_sync_source_tx_wait_time` is available instead.

- `Rpl_semi_sync_master_tx_waits`

  The total number of times the source waited for transactions.

  `Rpl_semi_sync_master_tx_waits` is available when the `rpl_semi_sync_master` (`semisync_master.so` library) plugin was installed on the replica to set up semisynchronous replication. If the `rpl_semi_sync_source` plugin (`semisync_source.so` library) was installed, `Rpl_semi_sync_source_tx_waits` is available instead.

- `Rpl_semi_sync_master_wait_pos_backtraverse`

  The total number of times the source waited for an event with binary coordinates lower than events waited for previously. This can occur when the order in which transactions start waiting for a reply is different from the order in which their binary log events are written.

  `Rpl_semi_sync_master_wait_pos_backtraverse` is available when the `rpl_semi_sync_master` (`semisync_master.so` library) plugin was installed on the replica to set up semisynchronous replication. If the `rpl_semi_sync_source` plugin (`semisync_source.so` library) was installed, `Rpl_semi_sync_source_wait_pos_backtraverse` is available instead.

- `Rpl_semi_sync_master_wait_sessions`

  The number of sessions currently waiting for replica replies.

  `Rpl_semi_sync_master_wait_sessions` is available when the `rpl_semi_sync_master` (`semisync_master.so` library) plugin was installed on the replica to set up semisynchronous replication. If the `rpl_semi_sync_source` plugin (`semisync_source.so` library) was installed, `Rpl_semi_sync_source_wait_sessions` is available instead.

- `Rpl_semi_sync_master_yes_tx`

  The number of commits that were acknowledged successfully by a replica.

  `Rpl_semi_sync_master_yes_tx` is available when the `rpl_semi_sync_master` (`semisync_master.so` library) plugin was installed on the replica to set up semisynchronous replication. If the `rpl_semi_sync_source` plugin (`semisync_source.so` library) was installed, `Rpl_semi_sync_source_yes_tx` is available instead.

- `Rpl_semi_sync_source_clients`

  The number of semisynchronous replicas.

  `Rpl_semi_sync_source_clients` is available when the `rpl_semi_sync_source` (`semisync_source.so` library) plugin was installed on the source to set up semisynchronous replication. If the `rpl_semi_sync_master` plugin (`semisync_master.so` library) was installed, `Rpl_semi_sync_master_clients` is available instead.

- `Rpl_semi_sync_source_net_avg_wait_time`

  The average time in microseconds the source waited for a replica reply. This variable is always `0`, and is deprecated; expect it to be removed in a future version.

  `Rpl_semi_sync_source_net_avg_wait_time` is available when the `rpl_semi_sync_source` (`semisync_source.so` library) plugin was installed on the source to set up semisynchronous replication. If the `rpl_semi_sync_master` plugin (`semisync_master.so` library) was installed, `Rpl_semi_sync_master_net_avg_wait_time` is available instead.

- `Rpl_semi_sync_source_net_wait_time`

  The total time in microseconds the source waited for replica replies. This variable is always `0`, and is deprecated; expect it to be removed in a future version.

  `Rpl_semi_sync_source_net_wait_time` is available when the `rpl_semi_sync_source` (`semisync_source.so` library) plugin was installed on the source to set up semisynchronous replication. If the `rpl_semi_sync_master` plugin (`semisync_master.so` library) was installed, `Rpl_semi_sync_master_net_wait_time` is available instead.

- `Rpl_semi_sync_source_net_waits`

  The total number of times the source waited for replica replies.

  `Rpl_semi_sync_source_net_waits` is available when the `rpl_semi_sync_source` (`semisync_source.so` library) plugin was installed on the source to set up semisynchronous replication. If the `rpl_semi_sync_master` plugin (`semisync_master.so` library) was installed, `Rpl_semi_sync_master_net_waits` is available instead.

- `Rpl_semi_sync_source_no_times`

  The number of times the source turned off semisynchronous replication.

  `Rpl_semi_sync_source_no_times` is available when the `rpl_semi_sync_source` (`semisync_source.so` library) plugin was installed on the source to set up semisynchronous

replication. If the `rpl_semi_sync_master` plugin (`semisync_master.so` library) was installed, `Rpl_semi_sync_master_no_times` is available instead.

- `Rpl_semi_sync_source_no_tx`

  The number of commits that were not acknowledged successfully by a replica.

  `Rpl_semi_sync_source_no_tx` is available when the `rpl_semi_sync_source` (`semisync_source.so` library) plugin was installed on the source to set up semisynchronous replication. If the `rpl_semi_sync_master` plugin (`semisync_master.so` library) was installed, `Rpl_semi_sync_master_no_tx` is available instead.

- `Rpl_semi_sync_source_status`

  Whether semisynchronous replication currently is operational on the source. The value is `ON` if the plugin has been enabled and a commit acknowledgment has occurred. It is `OFF` if the plugin is not enabled or the source has fallen back to asynchronous replication due to commit acknowledgment timeout.

  `Rpl_semi_sync_source_status` is available when the `rpl_semi_sync_source` (`semisync_source.so` library) plugin was installed on the source to set up semisynchronous replication. If the `rpl_semi_sync_master` plugin (`semisync_master.so` library) was installed, `Rpl_semi_sync_master_status` is available instead.

- `Rpl_semi_sync_source_timefunc_failures`

  The number of times the source failed when calling time functions such as `gettimeofday()`.

  `Rpl_semi_sync_source_timefunc_failures` is available when the `rpl_semi_sync_source` (`semisync_source.so` library) plugin was installed on the source to set up semisynchronous replication. If the `rpl_semi_sync_master` plugin (`semisync_master.so` library) was installed, `Rpl_semi_sync_master_timefunc_failures` is available instead.

- `Rpl_semi_sync_source_tx_avg_wait_time`

  The average time in microseconds the source waited for each transaction.

  `Rpl_semi_sync_source_tx_avg_wait_time` is available when the `rpl_semi_sync_source` (`semisync_source.so` library) plugin was installed on the source to set up semisynchronous replication. If the `rpl_semi_sync_master` plugin (`semisync_master.so` library) was installed, `Rpl_semi_sync_master_tx_avg_wait_time` is available instead.

- `Rpl_semi_sync_source_tx_wait_time`

  The total time in microseconds the source waited for transactions.

  `Rpl_semi_sync_source_tx_wait_time` is available when the `rpl_semi_sync_source` (`semisync_source.so` library) plugin was installed on the source to set up semisynchronous replication. If the `rpl_semi_sync_master` plugin (`semisync_master.so` library) was installed, `Rpl_semi_sync_master_tx_wait_time` is available instead.

- `Rpl_semi_sync_source_tx_waits`

  The total number of times the source waited for transactions.

  `Rpl_semi_sync_source_tx_waits` is available when the `rpl_semi_sync_source` (`semisync_source.so` library) plugin was installed on the source to set up semisynchronous replication. If the `rpl_semi_sync_master` plugin (`semisync_master.so` library) was installed, `Rpl_semi_sync_master_tx_waits` is available instead.

- `Rpl_semi_sync_source_wait_pos_backtraverse`

  The total number of times the source waited for an event with binary coordinates lower than events waited for previously. This can occur when the order in which transactions start waiting for a reply is different from the order in which their binary log events are written.

  `Rpl_semi_sync_source_wait_pos_backtraverse` is available when the `rpl_semi_sync_source` (`semisync_source.so` library) plugin was installed on the source to set up semisynchronous replication. If the `rpl_semi_sync_master` plugin (`semisync_master.so` library) was installed, `Rpl_semi_sync_master_wait_pos_backtraverse` is available instead.

- `Rpl_semi_sync_source_wait_sessions`

  The number of sessions currently waiting for replica replies.

  `Rpl_semi_sync_source_wait_sessions` is available when the `rpl_semi_sync_source` (`semisync_source.so` library) plugin was installed on the source to set up semisynchronous replication. If the `rpl_semi_sync_master` plugin (`semisync_master.so` library) was installed, `Rpl_semi_sync_master_wait_sessions` is available instead.

- `Rpl_semi_sync_source_yes_tx`

  The number of commits that were acknowledged successfully by a replica.

  `Rpl_semi_sync_source_yes_tx` is available when the `rpl_semi_sync_source` (`semisync_source.so` library) plugin was installed on the source to set up semisynchronous replication. If the `rpl_semi_sync_master` plugin (`semisync_master.so` library) was installed, `Rpl_semi_sync_master_yes_tx` is available instead.

- `Rpl_semi_sync_replica_status`

  Shows whether semisynchronous replication is currently operational on the replica. This is `ON` if the plugin has been enabled and the replication I/O (receiver) thread is running, `OFF` otherwise.

  `Rpl_semi_sync_replica_status` is available when the `rpl_semi_sync_replica` (`semisync_replica.so` library) plugin was installed on the replica to set up semisynchronous replication. If the `rpl_semi_sync_slave` plugin (`semisync_slave.so` library) was installed, `Rpl_semi_sync_slave_status` is available instead.

- `Rpl_semi_sync_slave_status`

  Shows whether semisynchronous replication is currently operational on the replica. This is `ON` if the plugin has been enabled and the replication I/O (receiver) thread is running, `OFF` otherwise.

  `Rpl_semi_sync_slave_status` is available when the `rpl_semi_sync_slave` (`semisync_slave.so` library) plugin was installed on the replica to set up semisynchronous replication. If the `rpl_semi_sync_replica` plugin (`semisync_replica.so` library) was installed, `Rpl_semi_sync_replica_status` is available instead.

- `Rsa_public_key`

  The value of this variable is the public key used by the `sha256_password` authentication plugin for RSA key pair-based password exchange. The value is nonempty only if the server successfully initializes the private and public keys in the files named by the `sha256_password_private_key_path` and `sha256_password_public_key_path` system variables. The value of `Rsa_public_key` comes from the latter file.

  For information about `sha256_password`, see Section 6.4.1.3, “SHA-256 Pluggable Authentication”.

- `Secondary_engine_execution_count`

The number of queries offloaded to a secondary engine. This variable was added in MySQL 8.0.13.

For use with HeatWave. See MySQL HeatWave User Guide.

- `Select_full_join`

  The number of joins that perform table scans because they do not use indexes. If this value is not 0, you should carefully check the indexes of your tables.

- `Select_full_range_join`

  The number of joins that used a range search on a reference table.

- `Select_range`

  The number of joins that used ranges on the first table. This is normally not a critical issue even if the value is quite large.

- `Select_range_check`

  The number of joins without keys that check for key usage after each row. If this is not 0, you should carefully check the indexes of your tables.

- `Select_scan`

  The number of joins that did a full scan of the first table.

- `Slave_open_temp_tables`

  From MySQL 8.0.26, `Slave_open_temp_tables` is deprecated and the alias `Replica_open_temp_tables` should be used instead. In releases before MySQL 8.0.26, use `Slave_open_temp_tables`.

  `Slave_open_temp_tables` shows the number of temporary tables that the replication SQL thread currently has open. If the value is greater than zero, it is not safe to shut down the replica; see Section 17.5.1.31, “Replication and Temporary Tables”. This variable reports the total count of open temporary tables for *all* replication channels.

- `Slave_rows_last_search_algorithm_used`

  From MySQL 8.0.26, `Slave_rows_last_search_algorithm_used` is deprecated and the alias `Replica_rows_last_search_algorithm_used` should be used instead. In releases before MySQL 8.0.26, use `Slave_rows_last_search_algorithm_used`.

  `Slave_rows_last_search_algorithm_used` shows the search algorithm that was most recently used by this replica to locate rows for row-based replication. The result shows whether the replica used indexes, a table scan, or hashing as the search algorithm for the last transaction executed on any channel.

  The method used depends on the setting for the `slave_rows_search_algorithms` system variable, and the keys that are available on the relevant table.

  This variable is available only for debug builds of MySQL.

- `Slow_launch_threads`

  The number of threads that have taken more than `slow_launch_time` seconds to create.

- `Slow_queries`

  The number of queries that have taken more than `long_query_time` seconds. This counter increments regardless of whether the slow query log is enabled. For information about that log, see Section 5.4.5, “The Slow Query Log”.

- `Sort_merge_passes`

  The number of merge passes that the sort algorithm has had to do. If this value is large, you should consider increasing the value of the `sort_buffer_size` system variable.

- `Sort_range`

  The number of sorts that were done using ranges.

- `Sort_rows`

  The number of sorted rows.

- `Sort_scan`

  The number of sorts that were done by scanning the table.

- `Ssl_accept_renegotiates`

  The number of negotiates needed to establish the connection.

- `Ssl_accepts`

  The number of accepted SSL connections.

- `Ssl_callback_cache_hits`

  The number of callback cache hits.

- `Ssl_cipher`

  The current encryption cipher (empty for unencrypted connections).

- `Ssl_cipher_list`

  The list of possible SSL ciphers (empty for non-SSL connections). If MySQL supports TLSv1.3, the value includes the possible TLSv1.3 ciphersuites. See Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

- `Ssl_client_connects`

  The number of SSL connection attempts to an SSL-enabled replication source server.

- `Ssl_connect_renegotiates`

  The number of negotiates needed to establish the connection to an SSL-enabled replication source server.

- `Ssl_ctx_verify_depth`

  The SSL context verification depth (how many certificates in the chain are tested).

- `Ssl_ctx_verify_mode`

  The SSL context verification mode.

- `Ssl_default_timeout`

The default SSL timeout.

- `Ssl_finished_accepts`

  The number of successful SSL connections to the server.

- `Ssl_finished_connects`

  The number of successful replica connections to an SSL-enabled replication source server.

- `Ssl_server_not_after`

  The last date for which the SSL certificate is valid. To check SSL certificate expiration information, use this statement:

```
mysql> SHOW STATUS LIKE 'Ssl_server_not%';
+-----------------------+--------------------------+
| Variable_name         | Value                    |
+-----------------------+--------------------------+
| Ssl_server_not_after  | Apr 28 14:16:39 2025 GMT |
| Ssl_server_not_before | May  1 14:16:39 2015 GMT |
+-----------------------+--------------------------+
```

- `Ssl_server_not_before`

  The first date for which the SSL certificate is valid.

- `Ssl_session_cache_hits`

  The number of SSL session cache hits.

- `Ssl_session_cache_misses`

  The number of SSL session cache misses.

- `Ssl_session_cache_mode`

  The SSL session cache mode. When the value of the `ssl_session_cache_mode` server variable is `ON`, the value of the `Ssl_session_cache_mode` status variable is `SERVER`.

- `Ssl_session_cache_overflows`

  The number of SSL session cache overflows.

- `Ssl_session_cache_size`

  The SSL session cache size.

- `Ssl_session_cache_timeout`

  The timeout value in seconds of SSL sessions in the cache.

- `Ssl_session_cache_timeouts`

  The number of SSL session cache timeouts.

- `Ssl_sessions_reused`

  This is equal to 0 if TLS was not used in the current MySQL session, or if a TLS session has not been reused; otherwise it is equal to 1.

  `Ssl_sessions_reused` has session scope.

- `Ssl_used_session_cache_entries`

How many SSL session cache entries were used.

- `Ssl_verify_depth`

  The verification depth for replication SSL connections.

- `Ssl_verify_mode`

  The verification mode used by the server for a connection that uses SSL. The value is a bitmask; bits are defined in the `openssl/ssl.h` header file:

  ```
  # define SSL_VERIFY_NONE                 0x00
  # define SSL_VERIFY_PEER                 0x01
  # define SSL_VERIFY_FAIL_IF_NO_PEER_CERT 0x02
  # define SSL_VERIFY_CLIENT_ONCE          0x04
  ```

  `SSL_VERIFY_PEER` indicates that the server asks for a client certificate. If the client supplies one, the server performs verification and proceeds only if verification is successful. `SSL_VERIFY_CLIENT_ONCE` indicates that a request for the client certificate is performed only in the initial handshake.

- `Ssl_version`

  The SSL protocol version of the connection (for example, TLSv1). If the connection is not encrypted, the value is empty.

- `Table_locks_immediate`

  The number of times that a request for a table lock could be granted immediately.

- `Table_locks_waited`

  The number of times that a request for a table lock could not be granted immediately and a wait was needed. If this is high and you have performance problems, you should first optimize your queries, and then either split your table or tables or use replication.

- `Table_open_cache_hits`

  The number of hits for open tables cache lookups.

- `Table_open_cache_misses`

  The number of misses for open tables cache lookups.

- `Table_open_cache_overflows`

  The number of overflows for the open tables cache. This is the number of times, after a table is opened or closed, a cache instance has an unused entry and the size of the instance is larger than `table_open_cache` / `table_open_cache_instances`.

- `Tc_log_max_pages_used`

  For the memory-mapped implementation of the log that is used by `mysqld` when it acts as the transaction coordinator for recovery of internal XA transactions, this variable indicates the largest number of pages used for the log since the server started. If the product of `Tc_log_max_pages_used` and `Tc_log_page_size` is always significantly less than the log size, the size is larger than necessary and can be reduced. (The size is set by the `--log-tc-size` option. This variable is unused: It is unneeded for binary log-based recovery, and the memory-mapped recovery log method is not used unless the number of storage engines that are capable of two-phase commit and that support XA transactions is greater than one. (`InnoDB` is the only applicable engine.)

- `Tc_log_page_size`

The page size used for the memory-mapped implementation of the XA recovery log. The default value is determined using `getpagesize()`. This variable is unused for the same reasons as described for `Tc_log_max_pages_used`.

- `Tc_log_page_waits`

  For the memory-mapped implementation of the recovery log, this variable increments each time the server was not able to commit a transaction and had to wait for a free page in the log. If this value is large, you might want to increase the log size (with the `--log-tc-size` option). For binary log-based recovery, this variable increments each time the binary log cannot be closed because there are two-phase commits in progress. (The close operation waits until all such transactions are finished.)

- `Telemetry_traces_supported`

  Whether server telemetry traces is supported.

  For more information, see the *Server telemetry traces service* section in the MySQL Source Code documentation.

- `Threads_cached`

  The number of threads in the thread cache.

- `Threads_connected`

  The number of currently open connections.

- `Threads_created`

  The number of threads created to handle connections. If `Threads_created` is big, you may want to increase the `thread_cache_size` value. The cache miss rate can be calculated as `Threads_created`/`Connections`.

- `Threads_running`

  The number of threads that are not sleeping.

- `Tls_library_version`

  The runtime version of the OpenSSL library that is in use for this MySQL instance.

  This variable was added in MySQL 8.0.30.

- `Uptime`

  The number of seconds that the server has been up.

- `Uptime_since_flush_status`

  The number of seconds since the most recent `FLUSH STATUS` statement.

5.1.11 Server SQL Modes

The MySQL server can operate in different SQL modes, and can apply these modes differently for different clients, depending on the value of the `sql_mode` system variable. DBAs can set the global SQL mode to match site server operating requirements, and each application can set its session SQL mode to its own requirements.

Modes affect the SQL syntax MySQL supports and the data validation checks it performs. This makes it easier to use MySQL in different environments and to use MySQL together with other database servers.

- Setting the SQL Mode
- The Most Important SQL Modes
- Full List of SQL Modes
- Combination SQL Modes
- Strict SQL Mode
- Comparison of the IGNORE Keyword and Strict SQL Mode

For answers to questions often asked about server SQL modes in MySQL, see Section A.3, “MySQL 8.0 FAQ: Server SQL Mode”.

When working with `InnoDB` tables, consider also the `innodb_strict_mode` system variable. It enables additional error checks for `InnoDB` tables.

Setting the SQL Mode

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`.

To set the SQL mode at server startup, use the `--sql-mode="modes"` option on the command line, or `sql-mode="modes"` in an option file such as `my.cnf` (Unix operating systems) or `my.ini` (Windows). *`modes`* is a list of different modes separated by commas. To clear the SQL mode explicitly, set it to an empty string using `--sql-mode=""` on the command line, or `sql-mode=""` in an option file.

Note

MySQL installation programs may configure the SQL mode during the installation process.

If the SQL mode differs from the default or from what you expect, check for a setting in an option file that the server reads at startup.

To change the SQL mode at runtime, set the global or session `sql_mode` system variable using a `SET` statement:

```
SET GLOBAL sql_mode = 'modes';
SET SESSION sql_mode = 'modes';
```

Setting the `GLOBAL` variable requires the `SYSTEM_VARIABLES_ADMIN` privilege (or the deprecated `SUPER` privilege) and affects the operation of all clients that connect from that time on. Setting the `SESSION` variable affects only the current client. Each client can change its session `sql_mode` value at any time.

To determine the current global or session `sql_mode` setting, select its value:

```
SELECT @@GLOBAL.sql_mode;
SELECT @@SESSION.sql_mode;
```

Important

SQL mode and user-defined partitioning. Changing the server SQL mode after creating and inserting data into partitioned tables can cause major changes in the behavior of such tables, and could lead to loss or corruption of data. It is strongly recommended that you never change the SQL mode once you have created tables employing user-defined partitioning.

When replicating partitioned tables, differing SQL modes on the source and replica can also lead to problems. For best results, you should always use the same server SQL mode on the source and replica.

> For more information, see Section 24.6, “Restrictions and Limitations on Partitioning”.

The Most Important SQL Modes

The most important `sql_mode` values are probably these:

- `ANSI`

  This mode changes syntax and behavior to conform more closely to standard SQL. It is one of the special combination modes listed at the end of this section.

- `STRICT_TRANS_TABLES`

  If a value could not be inserted as given into a transactional table, abort the statement. For a nontransactional table, abort the statement if the value occurs in a single-row statement or the first row of a multiple-row statement. More details are given later in this section.

- `TRADITIONAL`

  Make MySQL behave like a “traditional” SQL database system. A simple description of this mode is “give an error instead of a warning” when inserting an incorrect value into a column. It is one of the special combination modes listed at the end of this section.

  Note

  With `TRADITIONAL` mode enabled, an `INSERT` or `UPDATE` aborts as soon as an error occurs. If you are using a nontransactional storage engine, this may not be what you want because data changes made prior to the error may not be rolled back, resulting in a “partially done” update.

When this manual refers to “strict mode,” it means a mode with either or both `STRICT_TRANS_TABLES` or `STRICT_ALL_TABLES` enabled.

Full List of SQL Modes

The following list describes all supported SQL modes:

- `ALLOW_INVALID_DATES`

  Do not perform full checking of dates. Check only that the month is in the range from 1 to 12 and the day is in the range from 1 to 31. This may be useful for Web applications that obtain year, month, and day in three different fields and store exactly what the user inserted, without date validation. This mode applies to `DATE` and `DATETIME` columns. It does not apply to `TIMESTAMP` columns, which always require a valid date.

  With `ALLOW_INVALID_DATES` disabled, the server requires that month and day values be legal, and not merely in the range 1 to 12 and 1 to 31, respectively. With strict mode disabled, invalid dates such as `'2004-04-31'` are converted to `'0000-00-00'` and a warning is generated. With strict mode enabled, invalid dates generate an error. To permit such dates, enable `ALLOW_INVALID_DATES`.

- `ANSI_QUOTES`

  Treat `"` as an identifier quote character (like the `` ` `` quote character) and not as a string quote character. You can still use `` ` `` to quote identifiers with this mode enabled. With `ANSI_QUOTES` enabled, you cannot use double quotation marks to quote literal strings because they are interpreted as identifiers.

- `ERROR_FOR_DIVISION_BY_ZERO`

The ERROR_FOR_DIVISION_BY_ZERO mode affects handling of division by zero, which includes MOD(N,0). For data-change operations (INSERT, UPDATE), its effect also depends on whether strict SQL mode is enabled.

- If this mode is not enabled, division by zero inserts NULL and produces no warning.
- If this mode is enabled, division by zero inserts NULL and produces a warning.
- If this mode and strict mode are enabled, division by zero produces an error, unless IGNORE is given as well. For INSERT IGNORE and UPDATE IGNORE, division by zero inserts NULL and produces a warning.

For SELECT, division by zero returns NULL. Enabling ERROR_FOR_DIVISION_BY_ZERO causes a warning to be produced as well, regardless of whether strict mode is enabled.

ERROR_FOR_DIVISION_BY_ZERO is deprecated. ERROR_FOR_DIVISION_BY_ZERO is not part of strict mode, but should be used in conjunction with strict mode and is enabled by default. A warning occurs if ERROR_FOR_DIVISION_BY_ZERO is enabled without also enabling strict mode or vice versa.

Because ERROR_FOR_DIVISION_BY_ZERO is deprecated, you should expect it to be removed in a future MySQL release as a separate mode name and its effect included in the effects of strict SQL mode.

- HIGH_NOT_PRECEDENCE

  The precedence of the NOT operator is such that expressions such as NOT a BETWEEN b AND c are parsed as NOT (a BETWEEN b AND c). In some older versions of MySQL, the expression was parsed as (NOT a) BETWEEN b AND c. The old higher-precedence behavior can be obtained by enabling the HIGH_NOT_PRECEDENCE SQL mode.

```
mysql> SET sql_mode = '';
mysql> SELECT NOT 1 BETWEEN -5 AND 5;
        -> 0
mysql> SET sql_mode = 'HIGH_NOT_PRECEDENCE';
mysql> SELECT NOT 1 BETWEEN -5 AND 5;
        -> 1
```

- IGNORE_SPACE

  Permit spaces between a function name and the (character. This causes built-in function names to be treated as reserved words. As a result, identifiers that are the same as function names must be quoted as described in Section 9.2, “Schema Object Names”. For example, because there is a COUNT() function, the use of count as a table name in the following statement causes an error:

```
mysql> CREATE TABLE count (i INT);
ERROR 1064 (42000): You have an error in your SQL syntax
```

  The table name should be quoted:

```
mysql> CREATE TABLE `count` (i INT);
Query OK, 0 rows affected (0.00 sec)
```

  The IGNORE_SPACE SQL mode applies to built-in functions, not to loadable functions or stored functions. It is always permissible to have spaces after a loadable function or stored function name, regardless of whether IGNORE_SPACE is enabled.

  For further discussion of IGNORE_SPACE, see Section 9.2.5, “Function Name Parsing and Resolution”.

- NO_AUTO_VALUE_ON_ZERO

  NO_AUTO_VALUE_ON_ZERO affects handling of AUTO_INCREMENT columns. Normally, you generate the next sequence number for the column by inserting either NULL or 0 into it. NO_AUTO_VALUE_ON_ZERO suppresses this behavior for 0 so that only NULL generates the next sequence number.

  This mode can be useful if 0 has been stored in a table's AUTO_INCREMENT column. (Storing 0 is not a recommended practice, by the way.) For example, if you dump the table with mysqldump and then reload it, MySQL normally generates new sequence numbers when it encounters the 0 values, resulting in a table with contents different from the one that was dumped. Enabling NO_AUTO_VALUE_ON_ZERO before reloading the dump file solves this problem. For this reason, mysqldump automatically includes in its output a statement that enables NO_AUTO_VALUE_ON_ZERO.

- NO_BACKSLASH_ESCAPES

  Enabling this mode disables the use of the backslash character (\) as an escape character within strings and identifiers. With this mode enabled, backslash becomes an ordinary character like any other, and the default escape sequence for LIKE expressions is changed so that no escape character is used.

- NO_DIR_IN_CREATE

  When creating a table, ignore all INDEX DIRECTORY and DATA DIRECTORY directives. This option is useful on replica servers.

- NO_ENGINE_SUBSTITUTION

  Control automatic substitution of the default storage engine when a statement such as CREATE TABLE or ALTER TABLE specifies a storage engine that is disabled or not compiled in.

  By default, NO_ENGINE_SUBSTITUTION is enabled.

  Because storage engines can be pluggable at runtime, unavailable engines are treated the same way:

  With NO_ENGINE_SUBSTITUTION disabled, for CREATE TABLE the default engine is used and a warning occurs if the desired engine is unavailable. For ALTER TABLE, a warning occurs and the table is not altered.

  With NO_ENGINE_SUBSTITUTION enabled, an error occurs and the table is not created or altered if the desired engine is unavailable.

- NO_UNSIGNED_SUBTRACTION

  Subtraction between integer values, where one is of type UNSIGNED, produces an unsigned result by default. If the result would otherwise have been negative, an error results:

```
mysql> SET sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT CAST(0 AS UNSIGNED) - 1;
ERROR 1690 (22003): BIGINT UNSIGNED value is out of range in '(cast(0 as unsigned) - 1)'
```

  If the NO_UNSIGNED_SUBTRACTION SQL mode is enabled, the result is negative:

```
mysql> SET sql_mode = 'NO_UNSIGNED_SUBTRACTION';
mysql> SELECT CAST(0 AS UNSIGNED) - 1;
+--------------------------+
| CAST(0 AS UNSIGNED) - 1 |
+--------------------------+
|                       -1 |
+--------------------------+
```

If the result of such an operation is used to update an `UNSIGNED` integer column, the result is clipped to the maximum value for the column type, or clipped to 0 if `NO_UNSIGNED_SUBTRACTION` is enabled. With strict SQL mode enabled, an error occurs and the column remains unchanged.

When `NO_UNSIGNED_SUBTRACTION` is enabled, the subtraction result is signed, *even if any operand is unsigned*. For example, compare the type of column `c2` in table `t1` with that of column `c2` in table `t2`:

```
mysql> SET sql_mode='';
mysql> CREATE TABLE test (c1 BIGINT UNSIGNED NOT NULL);
mysql> CREATE TABLE t1 SELECT c1 - 1 AS c2 FROM test;
mysql> DESCRIBE t1;
+-------+---------------------+------+-----+---------+-------+
| Field | Type                | Null | Key | Default | Extra |
+-------+---------------------+------+-----+---------+-------+
| c2    | bigint(21) unsigned | NO   |     | 0       |       |
+-------+---------------------+------+-----+---------+-------+

mysql> SET sql_mode='NO_UNSIGNED_SUBTRACTION';
mysql> CREATE TABLE t2 SELECT c1 - 1 AS c2 FROM test;
mysql> DESCRIBE t2;
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| c2    | bigint(21) | NO   |     | 0       |       |
+-------+------------+------+-----+---------+-------+
```

This means that `BIGINT UNSIGNED` is not 100% usable in all contexts. See Section 12.11, “Cast Functions and Operators”.

- `NO_ZERO_DATE`

  The `NO_ZERO_DATE` mode affects whether the server permits `'0000-00-00'` as a valid date. Its effect also depends on whether strict SQL mode is enabled.

  - If this mode is not enabled, `'0000-00-00'` is permitted and inserts produce no warning.
  - If this mode is enabled, `'0000-00-00'` is permitted and inserts produce a warning.
  - If this mode and strict mode are enabled, `'0000-00-00'` is not permitted and inserts produce an error, unless `IGNORE` is given as well. For `INSERT IGNORE` and `UPDATE IGNORE`, `'0000-00-00'` is permitted and inserts produce a warning.

  `NO_ZERO_DATE` is deprecated. `NO_ZERO_DATE` is not part of strict mode, but should be used in conjunction with strict mode and is enabled by default. A warning occurs if `NO_ZERO_DATE` is enabled without also enabling strict mode or vice versa.

  Because `NO_ZERO_DATE` is deprecated, you should expect it to be removed in a future MySQL release as a separate mode name and its effect included in the effects of strict SQL mode.

- `NO_ZERO_IN_DATE`

  The `NO_ZERO_IN_DATE` mode affects whether the server permits dates in which the year part is nonzero but the month or day part is 0. (This mode affects dates such as `'2010-00-01'` or `'2010-01-00'`, but not `'0000-00-00'`. To control whether the server permits `'0000-00-00'`,

use the NO_ZERO_DATE mode.) The effect of NO_ZERO_IN_DATE also depends on whether strict SQL mode is enabled.

- If this mode is not enabled, dates with zero parts are permitted and inserts produce no warning.
- If this mode is enabled, dates with zero parts are inserted as '0000-00-00' and produce a warning.
- If this mode and strict mode are enabled, dates with zero parts are not permitted and inserts produce an error, unless IGNORE is given as well. For INSERT IGNORE and UPDATE IGNORE, dates with zero parts are inserted as '0000-00-00' and produce a warning.

NO_ZERO_IN_DATE is deprecated. NO_ZERO_IN_DATE is not part of strict mode, but should be used in conjunction with strict mode and is enabled by default. A warning occurs if NO_ZERO_IN_DATE is enabled without also enabling strict mode or vice versa.

Because NO_ZERO_IN_DATE is deprecated, you should expect it to be removed in a future MySQL release as a separate mode name and its effect included in the effects of strict SQL mode.

- ONLY_FULL_GROUP_BY

  Reject queries for which the select list, HAVING condition, or ORDER BY list refer to nonaggregated columns that are neither named in the GROUP BY clause nor are functionally dependent on (uniquely determined by) GROUP BY columns.

  A MySQL extension to standard SQL permits references in the HAVING clause to aliased expressions in the select list. The HAVING clause can refer to aliases regardless of whether ONLY_FULL_GROUP_BY is enabled.

  For additional discussion and examples, see Section 12.20.3, “MySQL Handling of GROUP BY”.

- PAD_CHAR_TO_FULL_LENGTH

  By default, trailing spaces are trimmed from CHAR column values on retrieval. If PAD_CHAR_TO_FULL_LENGTH is enabled, trimming does not occur and retrieved CHAR values are padded to their full length. This mode does not apply to VARCHAR columns, for which trailing spaces are retained on retrieval.

  Note

  As of MySQL 8.0.13, PAD_CHAR_TO_FULL_LENGTH is deprecated. Expect it to be removed in a future version of MySQL.

```
mysql> CREATE TABLE t1 (c1 CHAR(10));
Query OK, 0 rows affected (0.37 sec)

mysql> INSERT INTO t1 (c1) VALUES('xy');
Query OK, 1 row affected (0.01 sec)

mysql> SET sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT c1, CHAR_LENGTH(c1) FROM t1;
+------+----------------+
| c1   | CHAR_LENGTH(c1) |
+------+----------------+
| xy   |              2 |
+------+----------------+
1 row in set (0.00 sec)

mysql> SET sql_mode = 'PAD_CHAR_TO_FULL_LENGTH';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT c1, CHAR_LENGTH(c1) FROM t1;
```

```
+------------+-----------------+
| c1         | CHAR_LENGTH(c1) |
+------------+-----------------+
| xy         |              10 |
+------------+-----------------+
1 row in set (0.00 sec)
```

- `PIPES_AS_CONCAT`

  Treat `||` as a string concatenation operator (same as `CONCAT()`) rather than as a synonym for `OR`.

- `REAL_AS_FLOAT`

  Treat `REAL` as a synonym for `FLOAT`. By default, MySQL treats `REAL` as a synonym for `DOUBLE`.

- `STRICT_ALL_TABLES`

  Enable strict SQL mode for all storage engines. Invalid data values are rejected. For details, see Strict SQL Mode.

- `STRICT_TRANS_TABLES`

  Enable strict SQL mode for transactional storage engines, and when possible for nontransactional storage engines. For details, see Strict SQL Mode.

- `TIME_TRUNCATE_FRACTIONAL`

  Control whether rounding or truncation occurs when inserting a `TIME`, `DATE`, or `TIMESTAMP` value with a fractional seconds part into a column having the same type but fewer fractional digits. The default behavior is to use rounding. If this mode is enabled, truncation occurs instead. The following sequence of statements illustrates the difference:

  ```
  CREATE TABLE t (id INT, tval TIME(1));
  SET sql_mode='';
  INSERT INTO t (id, tval) VALUES(1, 1.55);
  SET sql_mode='TIME_TRUNCATE_FRACTIONAL';
  INSERT INTO t (id, tval) VALUES(2, 1.55);
  ```

  The resulting table contents look like this, where the first value has been subject to rounding and the second to truncation:

  ```
  mysql> SELECT id, tval FROM t ORDER BY id;
  +------+------------+
  | id   | tval       |
  +------+------------+
  |    1 | 00:00:01.6 |
  |    2 | 00:00:01.5 |
  +------+------------+
  ```

  See also Section 11.2.6, “Fractional Seconds in Time Values”.

Combination SQL Modes

The following special modes are provided as shorthand for combinations of mode values from the preceding list.

- `ANSI`

  Equivalent to `REAL_AS_FLOAT`, `PIPES_AS_CONCAT`, `ANSI_QUOTES`, `IGNORE_SPACE`, and `ONLY_FULL_GROUP_BY`.

  `ANSI` mode also causes the server to return an error for queries where a set function *`S`* with an outer reference *`S(outer_ref)`* cannot be aggregated in the outer query against which the outer reference has been resolved. This is such a query:

  ```
  SELECT * FROM t1 WHERE t1.a IN (SELECT MAX(t1.b) FROM t2 WHERE ...);
  ```

Here, `MAX(t1.b)` cannot aggregated in the outer query because it appears in the `WHERE` clause of that query. Standard SQL requires an error in this situation. If `ANSI` mode is not enabled, the server treats *`S(outer_ref)`* in such queries the same way that it would interpret *`S(const)`*.

See Section 1.6, “MySQL Standards Compliance”.

- `TRADITIONAL`

  `TRADITIONAL` is equivalent to `STRICT_TRANS_TABLES`, `STRICT_ALL_TABLES`, `NO_ZERO_IN_DATE`, `NO_ZERO_DATE`, `ERROR_FOR_DIVISION_BY_ZERO`, and `NO_ENGINE_SUBSTITUTION`.

Strict SQL Mode

Strict mode controls how MySQL handles invalid or missing values in data-change statements such as `INSERT` or `UPDATE`. A value can be invalid for several reasons. For example, it might have the wrong data type for the column, or it might be out of range. A value is missing when a new row to be inserted does not contain a value for a non-`NULL` column that has no explicit `DEFAULT` clause in its definition. (For a `NULL` column, `NULL` is inserted if the value is missing.) Strict mode also affects DDL statements such as `CREATE TABLE`.

If strict mode is not in effect, MySQL inserts adjusted values for invalid or missing values and produces warnings (see Section 13.7.7.42, “SHOW WARNINGS Statement”). In strict mode, you can produce this behavior by using `INSERT IGNORE` or `UPDATE IGNORE`.

For statements such as `SELECT` that do not change data, invalid values generate a warning in strict mode, not an error.

Strict mode produces an error for attempts to create a key that exceeds the maximum key length. When strict mode is not enabled, this results in a warning and truncation of the key to the maximum key length.

Strict mode does not affect whether foreign key constraints are checked. `foreign_key_checks` can be used for that. (See Section 5.1.8, “Server System Variables”.)

Strict SQL mode is in effect if either `STRICT_ALL_TABLES` or `STRICT_TRANS_TABLES` is enabled, although the effects of these modes differ somewhat:

- For transactional tables, an error occurs for invalid or missing values in a data-change statement when either `STRICT_ALL_TABLES` or `STRICT_TRANS_TABLES` is enabled. The statement is aborted and rolled back.
- For nontransactional tables, the behavior is the same for either mode if the bad value occurs in the first row to be inserted or updated: The statement is aborted and the table remains unchanged. If the statement inserts or modifies multiple rows and the bad value occurs in the second or later row, the result depends on which strict mode is enabled:
  - For `STRICT_ALL_TABLES`, MySQL returns an error and ignores the rest of the rows. However, because the earlier rows have been inserted or updated, the result is a partial update. To avoid this, use single-row statements, which can be aborted without changing the table.
  - For `STRICT_TRANS_TABLES`, MySQL converts an invalid value to the closest valid value for the column and inserts the adjusted value. If a value is missing, MySQL inserts the implicit default value for the column data type. In either case, MySQL generates a warning rather than an error and continues processing the statement. Implicit defaults are described in Section 11.6, “Data Type Default Values”.

Strict mode affects handling of division by zero, zero dates, and zeros in dates as follows:

- Strict mode affects handling of division by zero, which includes `MOD(N,0)`:

For data-change operations (`INSERT`, `UPDATE`):

- If strict mode is not enabled, division by zero inserts `NULL` and produces no warning.
- If strict mode is enabled, division by zero produces an error, unless `IGNORE` is given as well. For `INSERT IGNORE` and `UPDATE IGNORE`, division by zero inserts `NULL` and produces a warning.

For `SELECT`, division by zero returns `NULL`. Enabling strict mode causes a warning to be produced as well.

- Strict mode affects whether the server permits `'0000-00-00'` as a valid date:
  - If strict mode is not enabled, `'0000-00-00'` is permitted and inserts produce no warning.
  - If strict mode is enabled, `'0000-00-00'` is not permitted and inserts produce an error, unless `IGNORE` is given as well. For `INSERT IGNORE` and `UPDATE IGNORE`, `'0000-00-00'` is permitted and inserts produce a warning.
- Strict mode affects whether the server permits dates in which the year part is nonzero but the month or day part is 0 (dates such as `'2010-00-01'` or `'2010-01-00'`):
  - If strict mode is not enabled, dates with zero parts are permitted and inserts produce no warning.
  - If strict mode is enabled, dates with zero parts are not permitted and inserts produce an error, unless `IGNORE` is given as well. For `INSERT IGNORE` and `UPDATE IGNORE`, dates with zero parts are inserted as `'0000-00-00'` (which is considered valid with `IGNORE`) and produce a warning.

For more information about strict mode with respect to `IGNORE`, see Comparison of the IGNORE Keyword and Strict SQL Mode.

Strict mode affects handling of division by zero, zero dates, and zeros in dates in conjunction with the `ERROR_FOR_DIVISION_BY_ZERO`, `NO_ZERO_DATE`, and `NO_ZERO_IN_DATE` modes.

Comparison of the IGNORE Keyword and Strict SQL Mode

This section compares the effect on statement execution of the `IGNORE` keyword (which downgrades errors to warnings) and strict SQL mode (which upgrades warnings to errors). It describes which statements they affect, and which errors they apply to.

The following table presents a summary comparison of statement behavior when the default is to produce an error versus a warning. An example of when the default is to produce an error is inserting a `NULL` into a `NOT NULL` column. An example of when the default is to produce a warning is inserting a value of the wrong data type into a column (such as inserting the string `'abc'` into an integer column).

Operational Mode	When Statement Default is Error	When Statement Default is Warning
Without `IGNORE` or strict SQL mode	Error	Warning
With `IGNORE`	Warning	Warning (same as without `IGNORE` or strict SQL mode)
With strict SQL mode	Error (same as without `IGNORE` or strict SQL mode)	Error
With `IGNORE` and strict SQL mode	Warning	Warning

One conclusion to draw from the table is that when the `IGNORE` keyword and strict SQL mode are both in effect, `IGNORE` takes precedence. This means that, although `IGNORE` and strict SQL mode can be considered to have opposite effects on error handling, they do not cancel when used together.

- The Effect of IGNORE on Statement Execution
- The Effect of Strict SQL Mode on Statement Execution

The Effect of IGNORE on Statement Execution

Several statements in MySQL support an optional `IGNORE` keyword. This keyword causes the server to downgrade certain types of errors and generate warnings instead. For a multiple-row statement, downgrading an error to a warning may enable a row to be processed. Otherwise, `IGNORE` causes the statement to skip to the next row instead of aborting. (For nonignorable errors, an error occurs regardless of the `IGNORE` keyword.)

Example: If the table `t` has a primary key column `i` containing unique values, attempting to insert the same value of `i` into multiple rows normally produces a duplicate-key error:

```
mysql> CREATE TABLE t (i INT NOT NULL PRIMARY KEY);
mysql> INSERT INTO t (i) VALUES(1),(1);
ERROR 1062 (23000): Duplicate entry '1' for key 't.PRIMARY'
```

With `IGNORE`, the row containing the duplicate key still is not inserted, but a warning occurs instead of an error:

```
mysql> INSERT IGNORE INTO t (i) VALUES(1),(1);
Query OK, 1 row affected, 1 warning (0.01 sec)
Records: 2  Duplicates: 1  Warnings: 1

mysql> SHOW WARNINGS;
+---------+------+-----------------------------------------+
| Level   | Code | Message                                 |
+---------+------+-----------------------------------------+
| Warning | 1062 | Duplicate entry '1' for key 't.PRIMARY' |
+---------+------+-----------------------------------------+
1 row in set (0.00 sec)
```

Example: If the table `t2` has a `NOT NULL` column `id`, attempting to insert `NULL` produces an error in strict SQL mode:

```
mysql> CREATE TABLE t2 (id INT NOT NULL);
mysql> INSERT INTO t2 (id) VALUES(1),(NULL),(3);
ERROR 1048 (23000): Column 'id' cannot be null
mysql> SELECT * FROM t2;
Empty set (0.00 sec)
```

If the SQL mode is not strict, `IGNORE` causes the `NULL` to be inserted as the column implicit default (0 in this case), which enables the row to be handled without skipping it:

```
mysql> INSERT INTO t2 (id) VALUES(1),(NULL),(3);
mysql> SELECT * FROM t2;
+----+
| id |
+----+
|  1 |
|  0 |
|  3 |
+----+
```

These statements support the `IGNORE` keyword:

- `CREATE TABLE ... SELECT`: `IGNORE` does not apply to the `CREATE TABLE` or `SELECT` parts of the statement but to inserts into the table of rows produced by the `SELECT`. Rows that duplicate an existing row on a unique key value are discarded.
- `DELETE`: `IGNORE` causes MySQL to ignore errors during the process of deleting rows.
- `INSERT`: With `IGNORE`, rows that duplicate an existing row on a unique key value are discarded. Rows set to values that would cause data conversion errors are set to the closest valid values instead.

For partitioned tables where no partition matching a given value is found, `IGNORE` causes the insert operation to fail silently for rows containing the unmatched value.

- `LOAD DATA`, `LOAD XML`: With `IGNORE`, rows that duplicate an existing row on a unique key value are discarded.
- `UPDATE`: With `IGNORE`, rows for which duplicate-key conflicts occur on a unique key value are not updated. Rows updated to values that would cause data conversion errors are updated to the closest valid values instead.

The `IGNORE` keyword applies to the following ignorable errors:

```
ER_BAD_NULL_ERROR
ER_DUP_ENTRY
ER_DUP_ENTRY_WITH_KEY_NAME
ER_DUP_KEY
ER_NO_PARTITION_FOR_GIVEN_VALUE
ER_NO_PARTITION_FOR_GIVEN_VALUE_SILENT
ER_NO_REFERENCED_ROW_2
ER_ROW_DOES_NOT_MATCH_GIVEN_PARTITION_SET
ER_ROW_IS_REFERENCED_2
ER_SUBQUERY_NO_1_ROW
ER_VIEW_CHECK_FAILED
```

The Effect of Strict SQL Mode on Statement Execution

The MySQL server can operate in different SQL modes, and can apply these modes differently for different clients, depending on the value of the `sql_mode` system variable. In “strict” SQL mode, the server upgrades certain warnings to errors.

For example, in non-strict SQL mode, inserting the string `'abc'` into an integer column results in conversion of the value to 0 and a warning:

```
mysql> SET sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t (i) VALUES('abc');
Query OK, 1 row affected, 1 warning (0.01 sec)

mysql> SHOW WARNINGS;
+---------+------+-------------------------------------------------------+
| Level   | Code | Message                                               |
+---------+------+-------------------------------------------------------+
| Warning | 1366 | Incorrect integer value: 'abc' for column 'i' at row 1 |
+---------+------+-------------------------------------------------------+
1 row in set (0.00 sec)
```

In strict SQL mode, the invalid value is rejected with an error:

```
mysql> SET sql_mode = 'STRICT_ALL_TABLES';
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t (i) VALUES('abc');
ERROR 1366 (HY000): Incorrect integer value: 'abc' for column 'i' at row 1
```

For more information about possible settings of the `sql_mode` system variable, see Section 5.1.11, “Server SQL Modes”.

Strict SQL mode applies to the following statements under conditions for which some value might be out of range or an invalid row is inserted into or deleted from a table:

- `ALTER TABLE`
- `CREATE TABLE`
- `CREATE TABLE ... SELECT`

- `DELETE` (both single table and multiple table)
- `INSERT`
- `LOAD DATA`
- `LOAD XML`
- `SELECT SLEEP()`
- `UPDATE` (both single table and multiple table)

Within stored programs, individual statements of the types just listed execute in strict SQL mode if the program was defined while strict mode was in effect.

Strict SQL mode applies to the following errors, which represent a class of errors in which an input value is either invalid or missing. A value is invalid if it has the wrong data type for the column or might be out of range. A value is missing if a new row to be inserted does not contain a value for a `NOT NULL` column that has no explicit `DEFAULT` clause in its definition.

```
ER_BAD_NULL_ERROR
ER_CUT_VALUE_GROUP_CONCAT
ER_DATA_TOO_LONG
ER_DATETIME_FUNCTION_OVERFLOW
ER_DIVISION_BY_ZERO
ER_INVALID_ARGUMENT_FOR_LOGARITHM
ER_NO_DEFAULT_FOR_FIELD
ER_NO_DEFAULT_FOR_VIEW_FIELD
ER_TOO_LONG_KEY
ER_TRUNCATED_WRONG_VALUE
ER_TRUNCATED_WRONG_VALUE_FOR_FIELD
ER_WARN_DATA_OUT_OF_RANGE
ER_WARN_NULL_TO_NOTNULL
ER_WARN_TOO_FEW_RECORDS
ER_WRONG_ARGUMENTS
ER_WRONG_VALUE_FOR_TYPE
WARN_DATA_TRUNCATED
```

Note

Because continued MySQL development defines new errors, there may be errors not in the preceding list to which strict SQL mode applies.

5.1.12 Connection Management

This section describes how MySQL Server manages connections. This includes a description of the available connection interfaces, how the server uses connection handler threads, details about the administrative connection interface, and management of DNS lookups.

5.1.12.1 Connection Interfaces

This section describes aspects of how the MySQL server manages client connections.

- Network Interfaces and Connection Manager Threads
- Client Connection Thread Management
- Connection Volume Management

Network Interfaces and Connection Manager Threads

The server is capable of listening for client connections on multiple network interfaces. Connection manager threads handle client connection requests on the network interfaces that the server listens to:

- On all platforms, one manager thread handles TCP/IP connection requests.

- On Unix, the same manager thread also handles Unix socket file connection requests.
- On Windows, one manager thread handles shared-memory connection requests, and another handles named-pipe connection requests.
- On all platforms, an additional network interface may be enabled to accept administrative TCP/IP connection requests. This interface can use the manager thread that handles “ordinary” TCP/IP requests, or a separate thread.

The server does not create threads to handle interfaces that it does not listen to. For example, a Windows server that does not have support for named-pipe connections enabled does not create a thread to handle them.

Individual server plugins or components may implement their own connection interface:

- X Plugin enables MySQL Server to communicate with clients using X Protocol. See Section 20.5, “X Plugin”.

Client Connection Thread Management

Connection manager threads associate each client connection with a thread dedicated to it that handles authentication and request processing for that connection. Manager threads create a new thread when necessary but try to avoid doing so by consulting the thread cache first to see whether it contains a thread that can be used for the connection. When a connection ends, its thread is returned to the thread cache if the cache is not full.

In this connection thread model, there are as many threads as there are clients currently connected, which has some disadvantages when server workload must scale to handle large numbers of connections. For example, thread creation and disposal becomes expensive. Also, each thread requires server and kernel resources, such as stack space. To accommodate a large number of simultaneous connections, the stack size per thread must be kept small, leading to a situation where it is either too small or the server consumes large amounts of memory. Exhaustion of other resources can occur as well, and scheduling overhead can become significant.

MySQL Enterprise Edition includes a thread pool plugin that provides an alternative thread-handling model designed to reduce overhead and improve performance. It implements a thread pool that increases server performance by efficiently managing statement execution threads for large numbers of client connections. See Section 5.6.3, “MySQL Enterprise Thread Pool”.

To control and monitor how the server manages threads that handle client connections, several system and status variables are relevant. (See Section 5.1.8, “Server System Variables”, and Section 5.1.10, “Server Status Variables”.)

- The `thread_cache_size` system variable determines the thread cache size. By default, the server autosizes the value at startup, but it can be set explicitly to override this default. A value of 0 disables caching, which causes a thread to be set up for each new connection and disposed of when the connection terminates. To enable *`N`* inactive connection threads to be cached, set `thread_cache_size` to *`N`* at server startup or at runtime. A connection thread becomes inactive when the client connection with which it was associated terminates.
- To monitor the number of threads in the cache and how many threads have been created because a thread could not be taken from the cache, check the `Threads_cached` and `Threads_created` status variables.
- When the thread stack is too small, this limits the complexity of the SQL statements the server can handle, the recursion depth of stored procedures, and other memory-consuming actions. To set a stack size of *`N`* bytes for each thread, start the server with `thread_stack` set to *`N`*.

Connection Volume Management

To control the maximum number of clients the server permits to connect simultaneously, set the `max_connections` system variable at server startup or at runtime. It may be necessary to increase

`max_connections` if more clients attempt to connect simultaneously then the server is configured to handle (see Section B.3.2.5, “Too many connections”). If the server refuses a connection because the `max_connections` limit is reached, it increments the `Connection_errors_max_connections` status variable.

`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”.

As of MySQL 8.0.14, the server also permits administrative connections on an administrative network interface, which you can set up using a dedicated IP address and port. See Section 5.1.12.2, “Administrative Connection Management”.

The Group Replication plugin interacts with MySQL Server using internal sessions to perform SQL API operations. In releases to MySQL 8.0.18, these sessions count towards the client connections limit specified by the `max_connections` server system variable. In those releases, if the server has reached the `max_connections` limit when Group Replication is started or attempts to perform an operation, the operation is unsuccessful and Group Replication or the server itself might stop. From MySQL 8.0.19, Group Replication's internal sessions are handled separately from client connections, so they do not count towards the `max_connections` limit and are not refused if the server has reached this limit.

The maximum number of client connections MySQL supports (that is, the maximum value to which `max_connections` can be set) depends on several factors:

- The quality of the thread library on a given platform.
- The amount of RAM available.
- The amount of RAM is used for each connection.
- The workload from each connection.
- The desired response time.
- The number of file descriptors available.

Linux or Solaris should be able to support at least 500 to 1000 simultaneous connections routinely and as many as 10,000 connections if you have many gigabytes of RAM available and the workload from each is low or the response time target undemanding.

Increasing the `max_connections` value increases the number of file descriptors that `mysqld` requires. If the required number of descriptors are not available, the server reduces the value of `max_connections`. For comments on file descriptor limits, see Section 8.4.3.1, “How MySQL Opens and Closes Tables”.

Increasing the `open_files_limit` system variable may be necessary, which may also require raising the operating system limit on how many file descriptors can be used by MySQL. Consult your operating system documentation to determine whether it is possible to increase the limit and how to do so. See also Section B.3.2.16, “File Not Found and Similar Errors”.

5.1.12.2 Administrative Connection Management

As mentioned in Connection Volume Management, to allow for the need to perform administrative operations even when `max_connections` connections are already established on the interfaces used for ordinary connections, the MySQL server permits a single administrative connection to users who have the `CONNECTION_ADMIN` privilege (or the deprecated `SUPER` privilege).

Additionally, as of MySQL 8.0.14, the server permits dedicating a TCP/IP port for administrative connections, as described in the following sections.

- Administrative Interface Characteristics
- Administrative Interface Support for Encrypted Connections

Administrative Interface Characteristics

The administrative connection interface has these characteristics:

- The server enables the interface only if the `admin_address` system variable is set at startup to indicate the IP address for it. If `admin_address` is not set, the server maintains no administrative interface.
- The `admin_port` system variable specifies the interface TCP/IP port number (default 33062).
- There is no limit on the number of administrative connections, but connections are permitted only for users who have the `SERVICE_CONNECTION_ADMIN` privilege.
- The `create_admin_listener_thread` system variable enables DBAs to choose at startup whether the administrative interface has its own separate thread. The default is `OFF`; that is, the manager thread for ordinary connections on the main interface also handles connections for the administrative interface.

These lines in the server `my.cnf` file enable the administrative interface on the loopback interface and configure it to use port number 33064 (that is, a port different from the default):

```
[mysqld]
admin_address=127.0.0.1
admin_port=33064
```

MySQL client programs connect to either the main or administrative interface by specifying appropriate connection parameters. If the server running on the local host is using the default TCP/IP port numbers of 3306 and 33062 for the main and administrative interfaces, these commands connect to those interfaces:

```
mysql --protocol=TCP --port=3306
mysql --protocol=TCP --port=33062
```

Administrative Interface Support for Encrypted Connections

Prior to MySQL 8.0.21, the administrative interface supports encrypted connections using the connection-encryption configuration that applies to the main interface. As of MySQL 8.0.21, the administrative interface has its own configuration parameters for encrypted connections. These correspond to the main interface parameters but enable independent configuration of encrypted connections for the administrative interface:

- The `admin_tls_xxx` and `admin_ssl_xxx` system variables are like the `tls_xxx` and `ssl_xxx` system variables, but they configure the TLS context for the administrative interface rather than the main interface.
- The `--admin-ssl` option is like the `--ssl` option, but it enables or disables support for encrypted connections on the administrative interface rather than the main interface.

  Because support for encrypted connections is enabled by default, it is normally unnecessary to specify `--admin-ssl`. As of MySQL 8.0.26, `--admin-ssl` is deprecated and subject to removal in a future MySQL version.

For general information about configuring connection-encryption support, see Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”, and Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”. That discussion is written for the main connection interface, but the parameter

names are similar for the administrative connection interface. Use that discussion together with the following remarks, which provide information specific to the administrative interface.

TLS configuration for the administrative interface follows these rules:

- If `--admin-ssl` is enabled (the default), the administrative interface supports encrypted connections. For connections on the interface, the applicable TLS context depends on whether any nondefault administrative TLS parameter is configured:
  - If all administrative TLS parameters have their default values, the administrative interface uses the same TLS context as the main interface.
  - If any administrative TLS parameter has a nondefault value, the administrative interface uses the TLS context defined by its own parameters. (This is the case if any `admin_tls_xxx` or `admin_ssl_xxx` system variable is set to a value different from its default.) If a valid TLS context cannot be created from those parameters, the administrative interface falls back to the main interface TLS context.
- If `--admin-ssl` is disabled (for example, by specifying `--admin-ssl=OFF`, encrypted connections to the administrative interface are disabled. This is true even if administrative TLS parameters have nondefault values because disabling `--admin-ssl` takes precedence.

  It is also possible to disable encrypted connections on the administrative interface without specifying `--admin-ssl` in negated form. Set the `admin_tls_version` system variable to the empty value to indicate that no TLS versions are supported. For example, these lines in the server `my.cnf` file disable encrypted connections on the administrative interface:

```
[mysqld]
admin_tls_version=''
```

Examples:

- This configuration in the server `my.cnf` file enables the administrative interface, but does not set any of the TLS parameters specific to that interface:

```
[mysqld]
admin_address=127.0.0.1
```

  As a result, the administrative interface supports encrypted connections (because encryption is supported by default when the administrative interface is enabled), and uses the main interface TLS context. When clients connect to the administrative interface, they should use the same certificate and key files as for ordinary connections on the main interface. For example (enter the command on a single line):

```
mysql --protocol=TCP --port=33062
      --ssl-ca=ca.pem
      --ssl-cert=client-cert.pem
      --ssl-key=client-key.pem
```

- This server configuration enables the administrative interface and sets the TLS certificate and key file parameters specific to that interface:

```
[mysqld]
admin_address=127.0.0.1
admin_ssl_ca=admin-ca.pem
admin_ssl_cert=admin-server-cert.pem
admin_ssl_key=admin-server-key.pem
```

  As a result, the administrative interface supports encrypted connections using its own TLS context. When clients connect to the administrative interface, they should use certificate and key files specific to that interface. For example (enter the command on a single line):

```
mysql --protocol=TCP --port=33062
      --ssl-ca=admin-ca.pem
```

```
      --ssl-cert=admin-client-cert.pem
      --ssl-key=admin-client-key.pem
```

5.1.12.3 DNS Lookups and the Host Cache

The MySQL server maintains an in-memory host cache that contains information about clients: IP address, host name, and error information. The Performance Schema `host_cache` table exposes the contents of the host cache so that it can be examined using `SELECT` statements. This may help you diagnose the causes of connection problems. See Section 27.12.21.2, “The host_cache Table”.

The following sections discuss how the host cache works, as well as other topics such as how to configure and monitor the cache.

- Host Cache Operation
- Configuring the Host Cache
- Monitoring the Host Cache
- Flushing the Host Cache
- Dealing with Blocked Hosts

Host Cache Operation

The server uses the host cache only for non-localhost TCP connections. It does not use the cache for TCP connections established using a loopback interface address (for example, `127.0.0.1` or `::1`), or for connections established using a Unix socket file, named pipe, or shared memory.

The server uses the host cache for several purposes:

- By caching the results of IP-to-host name lookups, the server avoids doing a Domain Name System (DNS) lookup for each client connection. Instead, for a given host, it needs to perform a lookup only for the first connection from that host.
- The cache contains information about errors that occur during the client connection process. Some errors are considered “blocking.” If too many of these occur successively from a given host without a successful connection, the server blocks further connections from that host. The `max_connect_errors` system variable determines the permitted number of successive errors before blocking occurs.

For each applicable new client connection, the server uses the client IP address to check whether the client host name is in the host cache. If so, the server refuses or continues to process the connection request depending on whether or not the host is blocked. If the host is not in the cache, the server attempts to resolve the host name. First, it resolves the IP address to a host name and resolves that host name back to an IP address. Then it compares the result to the original IP address to ensure that they are the same. The server stores information about the result of this operation in the host cache. If the cache is full, the least recently used entry is discarded.

The server performs host name resolution using the `gethostbyaddr()` and `gethostbyname()` system calls.

The server handles entries in the host cache like this:

1. When the first TCP client connection reaches the server from a given IP address, a new cache entry is created to record the client IP, host name, and client lookup validation flag. Initially, the host name is set to `NULL` and the flag is false. This entry is also used for subsequent client TCP connections from the same originating IP.
2. If the validation flag for the client IP entry is false, the server attempts an IP-to-host name-to-IP DNS resolution. If that is successful, the host name is updated with the resolved host name and the validation flag is set to true. If resolution is unsuccessful, the action taken depends on whether

the error is permanent or transient. For permanent failures, the host name remains `NULL` and the validation flag is set to true. For transient failures, the host name and validation flag remain unchanged. (In this case, another DNS resolution attempt occurs the next time a client connects from this IP.)

3. If an error occurs while processing an incoming client connection from a given IP address, the server updates the corresponding error counters in the entry for that IP. For a description of the errors recorded, see Section 27.12.21.2, “The host_cache Table”.

To unblock blocked hosts, flush the host cache; see Dealing with Blocked Hosts.

It is possible for a blocked host to become unblocked even without flushing the host cache if activity from other hosts occurs:

- If the cache is full when a connection arrives from a client IP not in the cache, the server discards the least recently used cache entry to make room for the new entry.
- If the discarded entry is for a blocked host, that host becomes unblocked.

Some connection errors are not associated with TCP connections, occur very early in the connection process (even before an IP address is known), or are not specific to any particular IP address (such as out-of-memory conditions). For information about these errors, check the `Connection_errors_xxx` status variables (see Section 5.1.10, “Server Status Variables”).

Configuring the Host Cache

The host cache is enabled by default. The `host_cache_size` system variable controls its size, as well as the size of the Performance Schema `host_cache` table that exposes the cache contents. The cache size can be set at server startup and changed at runtime. For example, to set the size to 100 at startup, put these lines in the server `my.cnf` file:

```
[mysqld]
host_cache_size=200
```

To change the size to 300 at runtime, do this:

```
SET GLOBAL host_cache_size=300;
```

Setting `host_cache_size` to 0, either at server startup or at runtime, disables the host cache. With the cache disabled, the server performs a DNS lookup every time a client connects.

Changing the cache size at runtime causes an implicit host cache flushing operation that clears the host cache, truncates the `host_cache` table, and unblocks any blocked hosts; see Flushing the Host Cache.

Using the `--skip-host-cache` option is similar to setting the `host_cache_size` system variable to 0, but `host_cache_size` is more flexible because it can also be used to resize, enable, and disable the host cache at runtime, not just at server startup. Starting the server with `--skip-host-cache` does not prevent runtime changes to the value of `host_cache_size`, but such changes have no effect and the cache is not re-enabled even if `host_cache_size` is set larger than 0.

To disable DNS host name lookups, start the server with the `skip_name_resolve` system variable enabled. In this case, the server uses only IP addresses and not host names to match connecting hosts to rows in the MySQL grant tables. Only accounts specified in those tables using IP addresses can be used. (A client may not be able to connect if no account exists that specifies the client IP address.)

If you have a very slow DNS and many hosts, you might be able to improve performance either by enabling `skip_name_resolve` to disable DNS lookups, or by increasing the value of `host_cache_size` to make the host cache larger.

To disallow TCP/IP connections entirely, start the server with the `skip_networking` system variable enabled.

To adjust the permitted number of successive connection errors before host blocking occurs, set the `max_connect_errors` system variable. For example, to set the value at startup put these lines in the server `my.cnf` file:

```
[mysqld]
max_connect_errors=10000
```

To change the value at runtime, do this:

```
SET GLOBAL max_connect_errors=10000;
```

Monitoring the Host Cache

The Performance Schema `host_cache` table exposes the contents of the host cache. This table can be examined using `SELECT` statements, which may help you diagnose the causes of connection problems. For information about this table, see Section 27.12.21.2, “The host_cache Table”.

Flushing the Host Cache

Flushing the host cache might be advisable or desirable under these conditions:

- Some of your client hosts change IP address.
- The error message `Host 'host_name' is blocked` occurs for connections from legitimate hosts. (See Dealing with Blocked Hosts.)

Flushing the host cache has these effects:

- It clears the in-memory host cache.
- It removes all rows from the Performance Schema `host_cache` table that exposes the cache contents.
- It unblocks any blocked hosts. This enables further connection attempts from those hosts.

To flush the host cache, use any of these methods:

- Change the value of the `host_cache_size` system variable. This requires the `SYSTEM_VARIABLES_ADMIN` privilege (or the deprecated `SUPER` privilege).
- Execute a `TRUNCATE TABLE` statement that truncates the Performance Schema `host_cache` table. This requires the `DROP` privilege for the table.
- Execute a `FLUSH HOSTS` statement. This requires the `RELOAD` privilege.
- Execute a `mysqladmin flush-hosts` command. This requires the `DROP` privilege for the Performance Schema `host_cache` table or the `RELOAD` privilege.

Dealing with Blocked Hosts

The server uses the host cache to track errors that occur during the client connection process. If the following error occurs, it means that `mysqld` has received many connection requests from the given host that were interrupted in the middle:

```
Host 'host_name' is blocked because of many connection errors.
Unblock with 'mysqladmin flush-hosts'
```

The value of the `max_connect_errors` system variable determines how many successive interrupted connection requests the server permits before blocking a host. After `max_connect_errors` failed requests without a successful connection, the server assumes that something is wrong (for example, that someone is trying to break in), and blocks the host from further connection requests.

To unblock blocked hosts, flush the host cache; see Flushing the Host Cache.

Alternatively, to avoid having the error message occur, set `max_connect_errors` as described in Configuring the Host Cache. The default value of `max_connect_errors` is 100. Increasing `max_connect_errors` to a large value makes it less likely that a host reaches the threshold and becomes blocked. However, if the `Host 'host_name' is blocked` error message occurs, first verify that there is nothing wrong with TCP/IP connections from the blocked hosts. It does no good to increase the value of `max_connect_errors` if there are network problems.

5.1.13 IPv6 Support

Support for IPv6 in MySQL includes these capabilities:

- MySQL Server can accept TCP/IP connections from clients connecting over IPv6. For example, this command connects over IPv6 to the MySQL server on the local host:

  ```
  $> mysql -h ::1
  ```

  To use this capability, two things must be true:

  - Your system must be configured to support IPv6. See Section 5.1.13.1, “Verifying System Support for IPv6”.
  - The default MySQL server configuration permits IPv6 connections in addition to IPv4 connections. To change the default configuration, start the server with the `bind_address` system variable set to an appropriate value. See Section 5.1.8, “Server System Variables”.

- MySQL account names permit IPv6 addresses to enable DBAs to specify privileges for clients that connect to the server over IPv6. See Section 6.2.4, “Specifying Account Names”. IPv6 addresses can be specified in account names in statements such as `CREATE USER`, `GRANT`, and `REVOKE`. For example:

  ```
  mysql> CREATE USER 'bill'@'::1' IDENTIFIED BY 'secret';
  mysql> GRANT SELECT ON mydb.* TO 'bill'@'::1';
  ```

- IPv6 functions enable conversion between string and internal format IPv6 address formats, and checking whether values represent valid IPv6 addresses. For example, `INET6_ATON()` and `INET6_NTOA()` are similar to `INET_ATON()` and `INET_NTOA()`, but handle IPv6 addresses in addition to IPv4 addresses. See Section 12.24, “Miscellaneous Functions”.

- From MySQL 8.0.14, Group Replication group members can use IPv6 addresses for communications within the group. A group can contain a mix of members using IPv6 and members using IPv4. See Section 18.5.5, “Support For IPv6 And For Mixed IPv6 And IPv4 Groups”.

The following sections describe how to set up MySQL so that clients can connect to the server over IPv6.

5.1.13.1 Verifying System Support for IPv6

Before MySQL Server can accept IPv6 connections, the operating system on your server host must support IPv6. As a simple test to determine whether that is true, try this command:

```
$> ping6 ::1
16 bytes from ::1, icmp_seq=0 hlim=64 time=0.171 ms
16 bytes from ::1, icmp_seq=1 hlim=64 time=0.077 ms
...
```

To produce a description of your system's network interfaces, invoke `ifconfig -a` and look for IPv6 addresses in the output.

If your host does not support IPv6, consult your system documentation for instructions on enabling it. It might be that you need only reconfigure an existing network interface to add an IPv6 address. Or a more extensive change might be needed, such as rebuilding the kernel with IPv6 options enabled.

These links may be helpful in setting up IPv6 on various platforms:

- Windows
- Gentoo Linux
- Ubuntu Linux
- Linux (Generic)
- macOS

5.1.13.2 Configuring the MySQL Server to Permit IPv6 Connections

The MySQL server listens on one or more network sockets for TCP/IP connections. Each socket is bound to one address, but it is possible for an address to map onto multiple network interfaces.

Set the `bind_address` system variable at server startup to specify the TCP/IP connections that a server instance accepts. As of MySQL 8.0.13, you can specify multiple values for this option, including any combination of IPv6 addresses, IPv4 addresses, and host names that resolve to IPv6 or IPv4 addresses. Alternatively, you can specify one of the wildcard address formats that permit listening on multiple network interfaces. A value of *, which is the default, or a value of `::`, permit both IPv4 and IPv6 connections on all server host IPv4 and IPv6 interfaces. For more information, see the `bind_address` description in Section 5.1.8, “Server System Variables”.

5.1.13.3 Connecting Using the IPv6 Local Host Address

The following procedure shows how to configure MySQL to permit IPv6 connections by clients that connect to the local server using the `::1` local host address. The instructions given here assume that your system supports IPv6.

1. Start the MySQL server with an appropriate `bind_address` setting to permit it to accept IPv6 connections. For example, put the following lines in the server option file and restart the server:

   ```
   [mysqld]
   bind_address = *
   ```

   Specifying * (or `::`) as the value for `bind_address` permits both IPv4 and IPv6 connections on all server host IPv4 and IPv6 interfaces. If you want to bind the server to a specific list of addresses, you can do this as of MySQL 8.0.13 by specifying a comma-separated list of values for `bind_address`. This example specifies the local host addresses for both IPv4 and IPv6:

   ```
   [mysqld]
   bind_address = 127.0.0.1,::1
   ```

   For more information, see the `bind_address` description in Section 5.1.8, “Server System Variables”.

2. As an administrator, connect to the server and create an account for a local user who can connect from the `::1` local IPv6 host address:

   ```
   mysql> CREATE USER 'ipv6user'@'::1' IDENTIFIED BY 'ipv6pass';
   ```

   For the permitted syntax of IPv6 addresses in account names, see Section 6.2.4, “Specifying Account Names”. In addition to the `CREATE USER` statement, you can issue `GRANT` statements that give specific privileges to the account, although that is not necessary for the remaining steps in this procedure.

3. Invoke the `mysql` client to connect to the server using the new account:

   ```
   $> mysql -h ::1 -u ipv6user -pipv6pass
   ```

4. Try some simple statements that show connection information:

   ```
   mysql> STATUS
   ```

```
...
Connection:   ::1 via TCP/IP
...

mysql> SELECT CURRENT_USER(), @@bind_address;
+---------------+---------------+
| CURRENT_USER() | @@bind_address |
+---------------+---------------+
| ipv6user@::1   | ::             |
+---------------+---------------+
```

5.1.13.4 Connecting Using IPv6 Nonlocal Host Addresses

The following procedure shows how to configure MySQL to permit IPv6 connections by remote clients. It is similar to the preceding procedure for local clients, but the server and client hosts are distinct and each has its own nonlocal IPv6 address. The example uses these addresses:

```
Server host: 2001:db8:0:f101::1
Client host: 2001:db8:0:f101::2
```

These addresses are chosen from the nonroutable address range recommended by IANA for documentation purposes and suffice for testing on your local network. To accept IPv6 connections from clients outside the local network, the server host must have a public address. If your network provider assigns you an IPv6 address, you can use that. Otherwise, another way to obtain an address is to use an IPv6 broker; see Section 5.1.13.5, “Obtaining an IPv6 Address from a Broker”.

1. Start the MySQL server with an appropriate `bind_address` setting to permit it to accept IPv6 connections. For example, put the following lines in the server option file and restart the server:

   ```
   [mysqld]
   bind_address = *
   ```

   Specifying * (or `::`) as the value for `bind_address` permits both IPv4 and IPv6 connections on all server host IPv4 and IPv6 interfaces. If you want to bind the server to a specific list of addresses, you can do this as of MySQL 8.0.13 by specifying a comma-separated list of values for `bind_address`. This example specifies an IPv4 address as well as the required server host IPv6 address:

   ```
   [mysqld]
   bind_address = 198.51.100.20,2001:db8:0:f101::1
   ```

   For more information, see the `bind_address` description in Section 5.1.8, “Server System Variables”.

2. On the server host (`2001:db8:0:f101::1`), create an account for a user who can connect from the client host (`2001:db8:0:f101::2`):

   ```
   mysql> CREATE USER 'remoteipv6user'@'2001:db8:0:f101::2' IDENTIFIED BY 'remoteipv6pass';
   ```

3. On the client host (`2001:db8:0:f101::2`), invoke the `mysql` client to connect to the server using the new account:

   ```
   $> mysql -h 2001:db8:0:f101::1 -u remoteipv6user -premoteipv6pass
   ```

4. Try some simple statements that show connection information:

   ```
   mysql> STATUS
   ...
   Connection:   2001:db8:0:f101::1 via TCP/IP
   ...

   mysql> SELECT CURRENT_USER(), @@bind_address;
   +----------------------------------+----------------+
   | CURRENT_USER()                   | @@bind_address |
   +----------------------------------+----------------+
   | remoteipv6user@2001:db8:0:f101::2 | ::             |
   ```

```
+----------------------------------+---------------+
```

5.1.13.5 Obtaining an IPv6 Address from a Broker

If you do not have a public IPv6 address that enables your system to communicate over IPv6 outside your local network, you can obtain one from an IPv6 broker. The Wikipedia IPv6 Tunnel Broker page lists several brokers and their features, such as whether they provide static addresses and the supported routing protocols.

After configuring your server host to use a broker-supplied IPv6 address, start the MySQL server with an appropriate `bind_address` setting to permit the server to accept IPv6 connections. You can specify * (or `::`) as the `bind_address` value, or bind the server to the specific IPv6 address provided by the broker. For more information, see the `bind_address` description in Section 5.1.8, “Server System Variables”.

Note that if the broker allocates dynamic addresses, the address provided for your system might change the next time you connect to the broker. If so, any accounts you create that name the original address become invalid. To bind to a specific address but avoid this change-of-address problem, you might be able to arrange with the broker for a static IPv6 address.

The following example shows how to use Freenet6 as the broker and the `gogoc` IPv6 client package on Gentoo Linux.

1. Create an account at Freenet6 by visiting this URL and signing up:

   ```
   http://gogonet.gogo6.com
   ```

2. After creating the account, go to this URL, sign in, and create a user ID and password for the IPv6 broker:

   ```
   http://gogonet.gogo6.com/page/freenet6-registration
   ```

3. As `root`, install `gogoc`:

   ```
   $> emerge gogoc
   ```

4. Edit `/etc/gogoc/gogoc.conf` to set the `userid` and `password` values. For example:

   ```
   userid=gogouser
   passwd=gogopass
   ```

5. Start `gogoc`:

   ```
   $> /etc/init.d/gogoc start
   ```

   To start `gogoc` each time your system boots, execute this command:

   ```
   $> rc-update add gogoc default
   ```

6. Use `ping6` to try to ping a host:

   ```
   $> ping6 ipv6.google.com
   ```

7. To see your IPv6 address:

   ```
   $> ifconfig tun
   ```

5.1.14 Network Namespace Support

A network namespace is a logical copy of the network stack from the host system. Network namespaces are useful for setting up containers or virtual environments. Each namespace has its own IP addresses, network interfaces, routing tables, and so forth. The default or global namespace is the one in which the host system physical interfaces exist.

Namespace-specific address spaces can lead to problems when MySQL connections cross namespaces. For example, the network address space for a MySQL instance running in a container or virtual network may differ from the address space of the host machine. This can produce phenomena such as a client connection from an address in one namespace appearing to the MySQL server to be coming from a different address, even for client and server running on the same machine. Suppose that both processes run on a host with IP address `203.0.113.10` but use different namespaces. A connection may produce a result like this:

```
$> mysql --user=admin --host=203.0.113.10 --protocol=tcp

mysql> SELECT USER();
+--------------------+
| USER()             |
+--------------------+
| admin@198.51.100.2 |
+--------------------+
```

In this case, the expected `USER()` value is `admin@203.0.113.10`. Such behavior can make it difficult to assign account permissions properly if the address from which an connection originates is not what it appears.

To address this issue, MySQL enables specifying the network namespace to use for TCP/IP connections, so that both endpoints of connections use an agreed-upon common address space.

MySQL 8.0.22 and higher supports network namespaces on platforms that implement them. Support within MySQL applies to:

- The MySQL server, `mysqld`.
- X Plugin.
- The `mysql` client and the `mysqlxtest` test suite client. (Other clients are not supported. They must be invoked from within the network namespace of the server to which they are to connect.)
- Regular replication.
- Group Replication, only when using the MySQL communication stack to establish group communication connections (from MySQL 8.0.27).

The following sections describe how to use network namespaces in MySQL:

- Host System Prerequisites
- MySQL Configuration
- Network Namespace Monitoring

Host System Prerequisites

Prior to using network namespace support in MySQL, these host system prerequisites must be satisfied:

- The host operating system must support network namespaces. (For example, Linux.)
- Any network namespace to be used by MySQL must first be created on the host system.
- Host name resolution must be configured by the system administrator to support network namespaces.

Note

A known limitation is that, within MySQL, host name resolution does not work for names specified in network namespace-specific host files. For example,

> if the address for a host name in the `red` namespace is specified in the `/etc/netns/red/hosts` file, binding to the name fails on both the server and client sides. The workaround is to use the IP address rather than the host name.

- The system administrator must enable the `CAP_SYS_ADMIN` operating system privilege for the MySQL binaries that support network namespaces (`mysqld`, `mysql`, `mysqlxtest`).

> **Important**
>
> Enabling `CAP_SYS_ADMIN` is a security sensitive operation because it enables a process to perform other privileged actions in addition to setting namespaces. For a description of its effects, see https://man7.org/linux/man-pages/man7/capabilities.7.html.
>
> Because `CAP_SYS_ADMIN` must be enabled explicitly by the system administrator, MySQL binaries by default do not have network namespace support enabled. The system administrator should evaluate the security implications of running MySQL processes with `CAP_SYS_ADMIN` before enabling it.

The instructions in the following example set up network namespaces named `red` and `blue`. The names you choose may differ, as may the network addresses and interfaces on your host system.

Invoke the commands shown here either as the `root` operating system user or by prefixing each command with `sudo`. For example, to invoke the `ip` or `setcap` command if you are not `root`, use `sudo ip` or `sudo setcap`.

To configure network namespaces, use the `ip` command. For some operations, the `ip` command must execute within a particular namespace (which must already exist). In such cases, begin the command like this:

```
ip netns exec namespace_name
```

For example, this command executes within the `red` namespace to bring up the loopback interface:

```
ip netns exec red ip link set lo up
```

To add namespaces named `red` and `blue`, each with its own virtual Ethernet device used as a link between namespaces and its own loopback interface:

```
ip netns add red
ip link add veth-red type veth peer name vpeer-red
ip link set vpeer-red netns red
ip addr add 192.0.2.1/24 dev veth-red
ip link set veth-red up
ip netns exec red ip addr add 192.0.2.2/24 dev vpeer-red
ip netns exec red ip link set vpeer-red up
ip netns exec red ip link set lo up

ip netns add blue
ip link add veth-blue type veth peer name vpeer-blue
ip link set vpeer-blue netns blue
ip addr add 198.51.100.1/24 dev veth-blue
ip link set veth-blue up
ip netns exec blue ip addr add 198.51.100.2/24 dev vpeer-blue
ip netns exec blue ip link set vpeer-blue up
ip netns exec blue ip link set lo up

# if you want to enable inter-subnet routing...
sysctl net.ipv4.ip_forward=1
ip netns exec red ip route add default via 192.0.2.1
ip netns exec blue ip route add default via 198.51.100.1
```

A diagram of the links between namespaces looks like this:

```
red             global          blue

192.0.2.2   <=>  192.0.2.1
(vpeer-red)      (veth-red)

                 198.51.100.1 <=> 198.51.100.2
                 (veth-blue)      (vpeer-blue)
```

To check which namespaces and links exist:

```
ip netns list
ip link list
```

To see the routing tables for the global and named namespaces:

```
ip route show
ip netns exec red ip route show
ip netns exec blue ip route show
```

To remove the `red` and `blue` links and namespaces:

```
ip link del veth-red
ip link del veth-blue

ip netns del red
ip netns del blue

sysctl net.ipv4.ip_forward=0
```

So that the MySQL binaries that include network namespace support can actually use namespaces, you must grant them the `CAP_SYS_ADMIN` capability. The following `setcap` commands assume that you have changed location to the directory containing your MySQL binaries (adjust the pathname for your system as necessary):

```
cd /usr/local/mysql/bin
```

To grant `CAP_SYS_ADMIN` capability to the appropriate binaries:

```
setcap cap_sys_admin+ep ./mysqld
setcap cap_sys_admin+ep ./mysql
setcap cap_sys_admin+ep ./mysqlxtest
```

To check `CAP_SYS_ADMIN` capability:

```
$> getcap ./mysqld ./mysql ./mysqlxtest
./mysqld = cap_sys_admin+ep
./mysql = cap_sys_admin+ep
./mysqlxtest = cap_sys_admin+ep
```

To remove `CAP_SYS_ADMIN` capability:

```
setcap -r ./mysqld
setcap -r ./mysql
setcap -r ./mysqlxtest
```

Important

If you reinstall binaries to which you have previously applied `setcap`, you must use `setcap` again. For example, if you perform an in-place MySQL upgrade, failure to grant the `CAP_SYS_ADMIN` capability again results in namespace-related failures. The server fails with this error for attempts to bind to an address with a named namespace:

```
[ERROR] [MY-013408] [Server] setns() failed with error 'Operation not permitted'
```

A client invoked with the `--network-namespace` option fails like this:

```
ERROR: Network namespace error: Operation not permitted
```

MySQL Configuration

Assuming that the preceding host system prerequisites have been satisfied, MySQL enables configuring the server-side namespace for the listening (inbound) side of connections and the client-side namespace for the outbound side of connections.

On the server side, the `bind_address`, `admin_address`, and `mysqlx_bind_address` system variables have extended syntax for specifying the network namespace to use for a given IP address or host name on which to listen for incoming connections. To specify a namespace for an address, add a slash and the namespace name. For example, a server `my.cnf` file might contain these lines:

```
[mysqld]
bind_address = 127.0.1.1,192.0.2.2/red,198.51.100.2/blue
admin_address = 102.0.2.2/red
mysqlx_bind_address = 102.0.2.2/red
```

These rules apply:

- A network namespace can be specified for an IP address or a host name.
- A network namespace cannot be specified for a wildcard IP address.
- For a given address, the network namespace is optional. If given, it must be specified as a */ns* suffix immediately following the address.
- An address with no */ns* suffix uses the host system global namespace. The global namespace is therefore the default.
- An address with a */ns* suffix uses the namespace named *ns*.
- The host system must support network namespaces and each named namespace must previously have been set up. Naming a nonexistent namespace produces an error.
- `bind_address` and (as of MySQL 8.0.21) `mysqlx_bind_address` accept a list of multiple comma-separated addresses, the variable value can specify addresses in the global namespace, in named namespaces, or a mix.

If an error occurs during server startup for attempts to use a namespace, the server does not start. If errors occur for X Plugin during plugin initialization such that it is unable to bind to any address, the plugin fails its initialization sequence and the server does not load it.

On the client side, a network namespace can be specified in these contexts:

- For the `mysql` client and the `mysqlxtest` test suite client, use the `--network-namespace` option. For example:

  ```
  mysql --host=192.0.2.2 --network-namespace=red
  ```

  If the `--network-namespace` option is omitted, the connection uses the default (global) namespace.

- For replication connections from replica servers to source servers, use the `CHANGE REPLICATION SOURCE TO` statement (from MySQL 8.0.23) or `CHANGE MASTER TO` statement (before MySQL 8.0.23) and specify the `NETWORK_NAMESPACE` option. For example:

  ```
  CHANGE REPLICATION SOURCE TO
    SOURCE_HOST = '192.0.2.2',
    NETWORK_NAMESPACE = 'red';
  ```

  If the `NETWORK_NAMESPACE` option is omitted, replication connections use the default (global) namespace.

The following example sets up a MySQL server that listens for connections in the global, `red`, and `blue` namespaces, and shows how to configure accounts that connect from the `red` and `blue`

namespaces. It is assumed that the `red` and `blue` namespaces have already been created as shown in Host System Prerequisites.

1. Configure the server to listen on addresses in multiple namespaces. Put these lines in the server `my.cnf` file and start the server:

```
[mysqld]
bind_address = 127.0.1.1,192.0.2.2/red,198.51.100.2/blue
```

   The value tells the server to listen on the loopback address `127.0.0.1` in the global namespace, the address `192.0.2.2` in the `red` namespace, and the address `198.51.100.2` in the `blue` namespace.

2. Connect to the server in the global namespace and create accounts that have permission to connect from an address in the address space of each named namespace:

```
$> mysql -u root -h 127.0.0.1 -p
Enter password: root_password

mysql> CREATE USER 'red_user'@'192.0.2.2'
       IDENTIFIED BY 'red_user_password';
mysql> CREATE USER 'blue_user'@'198.51.100.2'
       IDENTIFIED BY 'blue_user_password';
```

3. Verify that you can connect to the server in each named namespace:

```
$> mysql -u red_user -h 192.0.2.2 --network-namespace=red -p
Enter password: red_user_password

mysql> SELECT USER();
+--------------------+
| USER()             |
+--------------------+
| red_user@192.0.2.2 |
+--------------------+
```

```
$> mysql -u blue_user -h 198.51.100.2 --network-namespace=blue -p
Enter password: blue_user_password

mysql> SELECT USER();
+------------------------+
| USER()                 |
+------------------------+
| blue_user@198.51.100.2 |
+------------------------+
```

Note

You might see different results from `USER()`, which can return a value that includes a host name rather than an IP address if your DNS is configured to be able to resolve the address to the corresponding host name and the server is not run with the `skip_name_resolve` system variable enabled.

   You might also try invoking `mysql` without the `--network-namespace` option to see whether the connection attempt succeeds, and, if so, how the `USER()` value is affected.

Network Namespace Monitoring

For replication monitoring purposes, these information sources have a column that displays the applicable network namespace for connections:

- The Performance Schema `replication_connection_configuration` table. See Section 27.12.11.1, “The replication_connection_configuration Table”.
- The replica server connection metadata repository. See Section 17.2.4.2, “Replication Metadata Repositories”.

- The `SHOW REPLICA STATUS` (or before MySQL 8.0.22, `SHOW SLAVE STATUS`) statement. See Section 13.7.7.35, “SHOW REPLICA STATUS Statement”.

5.1.15 MySQL Server Time Zone Support

This section describes the time zone settings maintained by MySQL, how to load the system tables required for named time support, how to stay current with time zone changes, and how to enable leap-second support.

Beginning with MySQL 8.0.19, time zone offsets are also supported for inserted datetime values; see Section 11.2.2, “The DATE, DATETIME, and TIMESTAMP Types”, for more information.

For information about time zone settings in replication setups, see Section 17.5.1.14, “Replication and System Functions” and Section 17.5.1.33, “Replication and Time Zones”.

- Time Zone Variables
- Populating the Time Zone Tables
- Staying Current with Time Zone Changes
- Time Zone Leap Second Support

Time Zone Variables

MySQL Server maintains several time zone settings:

- The server system time zone. When the server starts, it attempts to determine the time zone of the host machine and uses it to set the `system_time_zone` system variable.

  To explicitly specify the system time zone for MySQL Server at startup, set the `TZ` environment variable before you start `mysqld`. If you start the server using `mysqld_safe`, its `--timezone` option provides another way to set the system time zone. The permissible values for `TZ` and `--timezone` are system dependent. Consult your operating system documentation to see what values are acceptable.

- The server current time zone. The global `time_zone` system variable indicates the time zone the server currently is operating in. The initial `time_zone` value is `'SYSTEM'`, which indicates that the server time zone is the same as the system time zone.

  Note

  If set to `SYSTEM`, every MySQL function call that requires a time zone calculation makes a system library call to determine the current system time zone. This call may be protected by a global mutex, resulting in contention.

  The initial global server time zone value can be specified explicitly at startup with the `--default-time-zone` option on the command line, or you can use the following line in an option file:

  ```
  default-time-zone='timezone'
  ```

  If you have the `SYSTEM_VARIABLES_ADMIN` privilege (or the deprecated `SUPER` privilege), you can set the global server time zone value at runtime with this statement:

  ```
  SET GLOBAL time_zone = timezone;
  ```

- Per-session time zones. Each client that connects has its own session time zone setting, given by the session `time_zone` variable. Initially, the session variable takes its value from the global `time_zone` variable, but the client can change its own time zone with this statement:

  ```
  SET time_zone = timezone;
  ```

The session time zone setting affects display and storage of time values that are zone-sensitive. This includes the values displayed by functions such as `NOW()` or `CURTIME()`, and values stored in and retrieved from `TIMESTAMP` columns. Values for `TIMESTAMP` columns are converted from the session time zone to UTC for storage, and from UTC to the session time zone for retrieval.

The session time zone setting does not affect values displayed by functions such as `UTC_TIMESTAMP()` or values in `DATE`, `TIME`, or `DATETIME` columns. Nor are values in those data types stored in UTC; the time zone applies for them only when converting from `TIMESTAMP` values. If you want locale-specific arithmetic for `DATE`, `TIME`, or `DATETIME` values, convert them to UTC, perform the arithmetic, and then convert back.

The current global and session time zone values can be retrieved like this:

```
SELECT @@GLOBAL.time_zone, @@SESSION.time_zone;
```

`timezone` values can be given in several formats, none of which are case-sensitive:

- As the value `'SYSTEM'`, indicating that the server time zone is the same as the system time zone.
- As a string indicating an offset from UTC of the form *`[H]H:MM`*, prefixed with a `+` or `-`, such as `'+10:00'`, `'-6:00'`, or `'+05:30'`. A leading zero can optionally be used for hours values less than 10; MySQL prepends a leading zero when storing and retrieving the value in such cases. MySQL converts `'-00:00'` or `'-0:00'` to `'+00:00'`.

  Prior to MySQL 8.0.19, this value had to be in the range `'-12:59'` to `'+13:00'`, inclusive; beginning with MySQL 8.0.19, the permitted range is `'-13:59'` to `'+14:00'`, inclusive.
- As a named time zone, such as `'Europe/Helsinki'`, `'US/Eastern'`, `'MET'`, or `'UTC'`.

Note

Named time zones can be used only if the time zone information tables in the `mysql` database have been created and populated. Otherwise, use of a named time zone results in an error:

```
mysql> SET time_zone = 'UTC';
ERROR 1298 (HY000): Unknown or incorrect time zone: 'UTC'
```

Populating the Time Zone Tables

Several tables in the `mysql` system schema exist to store time zone information (see Section 5.3, “The mysql System Schema”). The MySQL installation procedure creates the time zone tables, but does not load them. To do so manually, use the following instructions.

Note

Loading the time zone information is not necessarily a one-time operation because the information changes occasionally. When such changes occur, applications that use the old rules become out of date and you may find it necessary to reload the time zone tables to keep the information used by your MySQL server current. See Staying Current with Time Zone Changes.

If your system has its own *zoneinfo* database (the set of files describing time zones), use the `mysql_tzinfo_to_sql` program to load the time zone tables. Examples of such systems are Linux, macOS, FreeBSD, and Solaris. One likely location for these files is the `/usr/share/zoneinfo` directory. If your system has no zoneinfo database, you can use a downloadable package, as described later in this section.

To load the time zone tables from the command line, pass the zoneinfo directory path name to `mysql_tzinfo_to_sql` and send the output into the `mysql` program. For example:

```
mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root -p mysql
```

The `mysql` command shown here assumes that you connect to the server using an account such as `root` that has privileges for modifying tables in the `mysql` system schema. Adjust the connection parameters as required.

`mysql_tzinfo_to_sql` reads your system's time zone files and generates SQL statements from them. `mysql` processes those statements to load the time zone tables.

`mysql_tzinfo_to_sql` also can be used to load a single time zone file or generate leap second information:

- To load a single time zone file *`tz_file`* that corresponds to a time zone name *`tz_name`*, invoke `mysql_tzinfo_to_sql` like this:

  ```
  mysql_tzinfo_to_sql tz_file tz_name | mysql -u root -p mysql
  ```

  With this approach, you must execute a separate command to load the time zone file for each named zone that the server needs to know about.

- If your time zone must account for leap seconds, initialize leap second information like this, where *`tz_file`* is the name of your time zone file:

  ```
  mysql_tzinfo_to_sql --leap tz_file | mysql -u root -p mysql
  ```

After running `mysql_tzinfo_to_sql`, restart the server so that it does not continue to use any previously cached time zone data.

If your system has no zoneinfo database (for example, Windows), you can use a package containing SQL statements that is available for download at the MySQL Developer Zone:

```
https://dev.mysql.com/downloads/timezones.html
```

Warning

Do *not* use a downloadable time zone package if your system has a zoneinfo database. Use the `mysql_tzinfo_to_sql` utility instead. Otherwise, you may cause a difference in datetime handling between MySQL and other applications on your system.

To use an SQL-statement time zone package that you have downloaded, unpack it, then load the unpacked file contents into the time zone tables:

```
mysql -u root -p mysql < file_name
```

Then restart the server.

Warning

Do *not* use a downloadable time zone package that contains `MyISAM` tables. That is intended for older MySQL versions. MySQL now uses `InnoDB` for the time zone tables. Trying to replace them with `MyISAM` tables causes problems.

Staying Current with Time Zone Changes

When time zone rules change, applications that use the old rules become out of date. To stay current, it is necessary to make sure that your system uses current time zone information is used. For MySQL, there are multiple factors to consider in staying current:

- The operating system time affects the value that the MySQL server uses for times if its time zone is set to `SYSTEM`. Make sure that your operating system is using the latest time zone information. For most operating systems, the latest update or service pack prepares your system for the time changes. Check the website for your operating system vendor for an update that addresses the time changes.

- If you replace the system's `/etc/localtime` time zone file with a version that uses rules differing from those in effect at `mysqld` startup, restart `mysqld` so that it uses the updated rules. Otherwise, `mysqld` might not notice when the system changes its time.
- If you use named time zones with MySQL, make sure that the time zone tables in the `mysql` database are up to date:
  - If your system has its own zoneinfo database, reload the MySQL time zone tables whenever the zoneinfo database is updated.
  - For systems that do not have their own zoneinfo database, check the MySQL Developer Zone for updates. When a new update is available, download it and use it to replace the content of your current time zone tables.

  For instructions for both methods, see Populating the Time Zone Tables. `mysqld` caches time zone information that it looks up, so after updating the time zone tables, restart `mysqld` to make sure that it does not continue to serve outdated time zone data.

If you are uncertain whether named time zones are available, for use either as the server's time zone setting or by clients that set their own time zone, check whether your time zone tables are empty. The following query determines whether the table that contains time zone names has any rows:

```
mysql> SELECT COUNT(*) FROM mysql.time_zone_name;
+----------+
| COUNT(*) |
+----------+
|        0 |
+----------+
```

A count of zero indicates that the table is empty. In this case, no applications currently are using named time zones, and you need not update the tables (unless you want to enable named time zone support). A count greater than zero indicates that the table is not empty and that its contents are available to be used for named time zone support. In this case, be sure to reload your time zone tables so that applications that use named time zones can obtain correct query results.

To check whether your MySQL installation is updated properly for a change in Daylight Saving Time rules, use a test like the one following. The example uses values that are appropriate for the 2007 DST 1-hour change that occurs in the United States on March 11 at 2 a.m.

The test uses this query:

```
SELECT
  CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central') AS time1,
  CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central') AS time2;
```

The two time values indicate the times at which the DST change occurs, and the use of named time zones requires that the time zone tables be used. The desired result is that both queries return the same result (the input time, converted to the equivalent value in the 'US/Central' time zone).

Before updating the time zone tables, you see an incorrect result like this:

```
+---------------------+---------------------+
| time1               | time2               |
+---------------------+---------------------+
| 2007-03-11 01:00:00 | 2007-03-11 02:00:00 |
+---------------------+---------------------+
```

After updating the tables, you should see the correct result:

```
+---------------------+---------------------+
| time1               | time2               |
+---------------------+---------------------+
| 2007-03-11 01:00:00 | 2007-03-11 01:00:00 |
+---------------------+---------------------+
```

Time Zone Leap Second Support

Leap second values are returned with a time part that ends with `:59:59`. This means that a function such as `NOW()` can return the same value for two or three consecutive seconds during the leap second. It remains true that literal temporal values having a time part that ends with `:59:60` or `:59:61` are considered invalid.

If it is necessary to search for `TIMESTAMP` values one second before the leap second, anomalous results may be obtained if you use a comparison with `'YYYY-MM-DD hh:mm:ss'` values. The following example demonstrates this. It changes the session time zone to UTC so there is no difference between internal `TIMESTAMP` values (which are in UTC) and displayed values (which have time zone correction applied).

```
mysql> CREATE TABLE t1 (
         a INT,
         ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
         PRIMARY KEY (ts)
       );
Query OK, 0 rows affected (0.01 sec)

mysql> -- change to UTC
mysql> SET time_zone = '+00:00';
Query OK, 0 rows affected (0.00 sec)

mysql> -- Simulate NOW() = '2008-12-31 23:59:59'
mysql> SET timestamp = 1230767999;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t1 (a) VALUES (1);
Query OK, 1 row affected (0.00 sec)

mysql> -- Simulate NOW() = '2008-12-31 23:59:60'
mysql> SET timestamp = 1230768000;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t1 (a) VALUES (2);
Query OK, 1 row affected (0.00 sec)

mysql> -- values differ internally but display the same
mysql> SELECT a, ts, UNIX_TIMESTAMP(ts) FROM t1;
+------+---------------------+--------------------+
| a    | ts                  | UNIX_TIMESTAMP(ts) |
+------+---------------------+--------------------+
|    1 | 2008-12-31 23:59:59 |         1230767999 |
|    2 | 2008-12-31 23:59:59 |         1230768000 |
+------+---------------------+--------------------+
2 rows in set (0.00 sec)

mysql> -- only the non-leap value matches
mysql> SELECT * FROM t1 WHERE ts = '2008-12-31 23:59:59';
+------+---------------------+
| a    | ts                  |
+------+---------------------+
|    1 | 2008-12-31 23:59:59 |
+------+---------------------+
1 row in set (0.00 sec)

mysql> -- the leap value with seconds=60 is invalid
mysql> SELECT * FROM t1 WHERE ts = '2008-12-31 23:59:60';
Empty set, 2 warnings (0.00 sec)
```

To work around this, you can use a comparison based on the UTC value actually stored in the column, which has the leap second correction applied:

```
mysql> -- selecting using UNIX_TIMESTAMP value return leap value
mysql> SELECT * FROM t1 WHERE UNIX_TIMESTAMP(ts) = 1230768000;
+------+---------------------+
| a    | ts                  |
+------+---------------------+
|    2 | 2008-12-31 23:59:59 |
+------+---------------------+
```

```
1 row in set (0.00 sec)
```

5.1.16 Resource Groups

MySQL supports creation and management of resource groups, and permits assigning threads running within the server to particular groups so that threads execute according to the resources available to the group. Group attributes enable control over its resources, to enable or restrict resource consumption by threads in the group. DBAs can modify these attributes as appropriate for different workloads.

Currently, CPU time is a manageable resource, represented by the concept of “virtual CPU” as a term that includes CPU cores, hyperthreads, hardware threads, and so forth. The server determines at startup how many virtual CPUs are available, and database administrators with appropriate privileges can associate these CPUs with resource groups and assign threads to groups.

For example, to manage execution of batch jobs that need not execute with high priority, a DBA can create a `Batch` resource group, and adjust its priority up or down depending on how busy the server is. (Perhaps batch jobs assigned to the group should run at lower priority during the day and at higher priority during the night.) The DBA can also adjust the set of CPUs available to the group. Groups can be enabled or disabled to control whether threads are assignable to them.

The following sections describe aspects of resource group use in MySQL:

- Resource Group Elements
- Resource Group Attributes
- Resource Group Management
- Resource Group Replication
- Resource Group Restrictions

Important

On some platforms or MySQL server configurations, resource groups are unavailable or have limitations. In particular, Linux systems might require a manual step for some installation methods. For details, see Resource Group Restrictions.

Resource Group Elements

These capabilities provide the SQL interface for resource group management in MySQL:

- SQL statements enable creating, altering, and dropping resource groups, and enable assigning threads to resource groups. An optimizer hint enables assigning individual statements to resource groups.
- Resource group privileges provide control over which users can perform resource group operations.
- The Information Schema `RESOURCE_GROUPS` table exposes information about resource group definitions and the Performance Schema `threads` table shows the resource group assignment for each thread.
- Status variables provide execution counts for each management SQL statement.

Resource Group Attributes

Resource groups have attributes that define the group. All attributes can be set at group creation time. Some attributes are fixed at creation time; others can be modified any time thereafter.

These attributes are defined at resource group creation time and cannot be modified:

- Each group has a name. Resource group names are identifiers like table and column names, and need not be quoted in SQL statements unless they contain special characters or are reserved words. Group names are not case-sensitive and may be up to 64 characters long.
- Each group has a type, which is either `SYSTEM` or `USER`. The resource group type affects the range of priority values assignable to the group, as described later. This attribute together with the differences in permitted priorities enables system threads to be identified so as to protect them from contention for CPU resources against user threads.

  System and user threads correspond to background and foreground threads as listed in the Performance Schema `threads` table.

These attributes are defined at resource group creation time and can be modified any time thereafter:

- The CPU affinity is the set of virtual CPUs the resource group can use. An affinity can be any nonempty subset of the available CPUs. If a group has no affinity, it can use all available CPUs.
- The thread priority is the execution priority for threads assigned to the resource group. Priority values range from -20 (highest priority) to 19 (lowest priority). The default priority is 0, for both system and user groups.

  System groups are permitted a higher priority than user groups, ensuring that user threads never have a higher priority than system threads:

  - For system resource groups, the permitted priority range is -20 to 0.
  - For user resource groups, the permitted priority range is 0 to 19.
- Each group can be enabled or disabled, affording administrators control over thread assignment. Threads can be assigned only to enabled groups.

Resource Group Management

By default, there is one system group and one user group, named `SYS_default` and `USR_default`, respectively. These default groups cannot be dropped and their attributes cannot be modified. Each default group has no CPU affinity and priority 0.

Newly created system and user threads are assigned to the `SYS_default` and `USR_default` groups, respectively.

For user-defined resource groups, all attributes are assigned at group creation time. After a group has been created, its attributes can be modified, with the exception of the name and type attributes.

To create and manage user-defined resource groups, use these SQL statements:

- `CREATE RESOURCE GROUP` creates a new group. See Section 13.7.2.2, “CREATE RESOURCE GROUP Statement”.
- `ALTER RESOURCE GROUP` modifies an existing group. See Section 13.7.2.1, “ALTER RESOURCE GROUP Statement”.
- `DROP RESOURCE GROUP` drops an existing group. See Section 13.7.2.3, “DROP RESOURCE GROUP Statement”.

Those statements require the `RESOURCE_GROUP_ADMIN` privilege.

To manage resource group assignments, use these capabilities:

- `SET RESOURCE GROUP` assigns threads to a group. See Section 13.7.2.4, “SET RESOURCE GROUP Statement”.
- The `RESOURCE_GROUP` optimizer hint assigns individual statements to a group. See Section 8.9.3, “Optimizer Hints”.

Those operations require the RESOURCE_GROUP_ADMIN or RESOURCE_GROUP_USER privilege.

Resource group definitions are stored in the resource_groups data dictionary table so that groups persist across server restarts. Because resource_groups is part of the data dictionary, it is not directly accessible by users. Resource group information is available using the Information Schema RESOURCE_GROUPS table, which is implemented as a view on the data dictionary table. See Section 26.3.26, “The INFORMATION_SCHEMA RESOURCE_GROUPS Table”.

Initially, the RESOURCE_GROUPS table has these rows describing the default groups:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.RESOURCE_GROUPS\G
*************************** 1. row ***************************
   RESOURCE_GROUP_NAME: USR_default
   RESOURCE_GROUP_TYPE: USER
RESOURCE_GROUP_ENABLED: 1
              VCPU_IDS: 0-3
       THREAD_PRIORITY: 0
*************************** 2. row ***************************
   RESOURCE_GROUP_NAME: SYS_default
   RESOURCE_GROUP_TYPE: SYSTEM
RESOURCE_GROUP_ENABLED: 1
              VCPU_IDS: 0-3
       THREAD_PRIORITY: 0
```

The THREAD_PRIORITY values are 0, indicating the default priority. The VCPU_IDS values show a range comprising all available CPUs. For the default groups, the displayed value varies depending on the system on which the MySQL server runs.

Earlier discussion mentioned a scenario involving a resource group named Batch to manage execution of batch jobs that need not execute with high priority. To create such a group, use a statement similar to this:

```
CREATE RESOURCE GROUP Batch
  TYPE = USER
  VCPU = 2-3            -- assumes a system with at least 4 CPUs
  THREAD_PRIORITY = 10;
```

To verify that the resource group was created as expected, check the RESOURCE_GROUPS table:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.RESOURCE_GROUPS
       WHERE RESOURCE_GROUP_NAME = 'Batch'\G
*************************** 1. row ***************************
   RESOURCE_GROUP_NAME: Batch
   RESOURCE_GROUP_TYPE: USER
RESOURCE_GROUP_ENABLED: 1
              VCPU_IDS: 2-3
       THREAD_PRIORITY: 10
```

If the THREAD_PRIORITY value is 0 rather than 10, check whether your platform or system configuration limits the resource group capability; see Resource Group Restrictions.

To assign a thread to the Batch group, do this:

```
SET RESOURCE GROUP Batch FOR thread_id;
```

Thereafter, statements in the named thread execute with Batch group resources.

If a session's own current thread should be in the Batch group, execute this statement within the session:

```
SET RESOURCE GROUP Batch;
```

Thereafter, statements in the session execute with Batch group resources.

To execute a single statement using the Batch group, use the RESOURCE_GROUP optimizer hint:

```
INSERT /*+ RESOURCE_GROUP(Batch) */ INTO t2 VALUES(2);
```

Threads assigned to the `Batch` group execute with its resources, which can be modified as desired:

- For times when the system is highly loaded, decrease the number of CPUs assigned to the group, lower its priority, or (as shown) both:

```
ALTER RESOURCE GROUP Batch
  VCPU = 3
  THREAD_PRIORITY = 19;
```

- For times when the system is lightly loaded, increase the number of CPUs assigned to the group, raise its priority, or (as shown) both:

```
ALTER RESOURCE GROUP Batch
  VCPU = 0-3
  THREAD_PRIORITY = 0;
```

Resource Group Replication

Resource group management is local to the server on which it occurs. Resource group SQL statements and modifications to the `resource_groups` data dictionary table are not written to the binary log and are not replicated.

Resource Group Restrictions

On some platforms or MySQL server configurations, resource groups are unavailable or have limitations:

- Resource groups are unavailable if the thread pool plugin is installed.
- Resource groups are unavailable on macOS, which provides no API for binding CPUs to a thread.
- On FreeBSD and Solaris, resource group thread priorities are ignored. (Effectively, all threads run at priority 0.) Attempts to change priorities result in a warning:

```
mysql> ALTER RESOURCE GROUP abc THREAD_PRIORITY = 10;
Query OK, 0 rows affected, 1 warning (0.18 sec)

mysql> SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------+
| Level   | Code | Message                                                         |
+---------+------+-----------------------------------------------------------------+
| Warning | 4560 | Attribute thread_priority is ignored (using default value). |
+---------+------+-----------------------------------------------------------------+
```

- On Linux, resource groups thread priorities are ignored unless the `CAP_SYS_NICE` capability is set. Granting `CAP_SYS_NICE` capability to a process enables a range of privileges; consult http://man7.org/linux/man-pages/man7/capabilities.7.html for the full list. Please be careful when enabling this capability.

  On Linux platforms using systemd and kernel support for Ambient Capabilities (Linux 4.3 or newer), the recommended way to enable `CAP_SYS_NICE` capability is to modify the MySQL service file and leave the `mysqld` binary unmodified. To adjust the service file for MySQL, use this procedure:

  1. Run the appropriate command for your platform:

     - Oracle Linux, Red Hat, and Fedora systems:

       ```
       $> sudo systemctl edit mysqld
       ```

     - SUSE, Ubuntu, and Debian systems:

       ```
       $> sudo systemctl edit mysql
       ```

  2. Using an editor, add the following text to the service file:

     ```
     [Service]
     ```

```
AmbientCapabilities=CAP_SYS_NICE
```

3. Restart the MySQL service.

If you cannot enable the `CAP_SYS_NICE` capability as just described, it can be set manually using the `setcap` command, specifying the path name to the `mysqld` executable (this requires `sudo` access). You can check the capabilities using `getcap`. For example:

```
$> sudo setcap cap_sys_nice+ep /path/to/mysqld
$> getcap /path/to/mysqld
/path/to/mysqld = cap_sys_nice+ep
```

As a safety measure, restrict execution of the `mysqld` binary to the `root` user and users with `mysql` group membership:

```
$> sudo chown root:mysql /path/to/mysqld
$> sudo chmod 0750 /path/to/mysqld
```

Important

If manual use of `setcap` is required, it must be performed after each reinstall.

- On Windows, threads run at one of five thread priority levels. The resource group thread priority range of -20 to 19 maps onto those levels as indicated in the following table.

Table 5.6 Resource Group Thread Priority on Windows

Priority Range	Windows Priority Level
-20 to -10	`THREAD_PRIORITY_HIGHEST`
-9 to -1	`THREAD_PRIORITY_ABOVE_NORMAL`
0	`THREAD_PRIORITY_NORMAL`
1 to 10	`THREAD_PRIORITY_BELOW_NORMAL`
11 to 19	`THREAD_PRIORITY_LOWEST`

5.1.17 Server-Side Help Support

MySQL Server supports a `HELP` statement that returns information from the MySQL Reference Manual (see Section 13.8.3, “HELP Statement”). This information is stored in several tables in the `mysql` schema (see Section 5.3, “The mysql System Schema”). Proper operation of the `HELP` statement requires that these help tables be initialized.

For a new installation of MySQL using a binary or source distribution on Unix, help-table content initialization occurs when you initialize the data directory (see Section 2.9.1, “Initializing the Data Directory”). For an RPM distribution on Linux or binary distribution on Windows, content initialization occurs as part of the MySQL installation process.

For a MySQL upgrade using a binary distribution, help-table content is upgraded automatically by the server as of MySQL 8.0.16. Prior to MySQL 8.0.16, the content is not upgraded automatically, but you can upgrade it manually. Locate the `fill_help_tables.sql` file in the `share` or `share/mysql` directory. Change location into that directory and process the file with the `mysql` client as follows:

```
mysql -u root -p mysql < fill_help_tables.sql
```

The command shown here assumes that you connect to the server using an account such as `root` that has privileges for modifying tables in the `mysql` schema. Adjust the connection parameters as required.

Prior to MySQL 8.0.16, if you are working with Git and a MySQL development source tree, the source tree contains only a “stub” version of `fill_help_tables.sql`. To obtain a non-stub copy, use one from a source or binary distribution.

Note

Each MySQL series has its own series-specific reference manual, so help-table content is series specific as well. This has implications for replication because help-table content should match the MySQL series. If you load MySQL 8.0 help content into a MySQL 8.0 replication server, it does not make sense to replicate that content to a replica server from a different MySQL series and for which that content is not appropriate. For this reason, as you upgrade individual servers in a replication scenario, you should upgrade each server's help tables, using the instructions given earlier. (Manual help-content upgrade is necessary only for replication servers from versions lower than 8.0.16. As mentioned in the preceding instructions, content upgrades occur automatically as of MySQL 8.0.16.)

5.1.18 Server Tracking of Client Session State

The MySQL server implements several session state trackers. A client can enable these trackers to receive notification of changes to its session state.

- Uses for Session State Trackers
- Available Session State Trackers
- C API Session State Tracker Support
- Test Suite Session State Tracker Support

Uses for Session State Trackers

Session state trackers have uses such as these:

- To facilitate session migration.
- To facilitate transaction switching.

The tracker mechanism provides a means for MySQL connectors and client applications to determine whether any session context is available to permit session migration from one server to another. (To change sessions in a load-balanced environment, it is necessary to detect whether there is session state to take into consideration when deciding whether a switch can be made.)

The tracker mechanism permits applications to know when transactions can be moved from one session to another. Transaction state tracking enables this, which is useful for applications that may wish to move transactions from a busy server to one that is less loaded. For example, a load-balancing connector managing a client connection pool could move transactions between available sessions in the pool.

However, session switching cannot be done at arbitrary times. If a session is in the middle of a transaction for which reads or writes have been done, switching to a different session implies a transaction rollback on the original session. A session switch must be done only when a transaction does not yet have any reads or writes performed within it.

Examples of when transactions might reasonably be switched:

- Immediately after `START TRANSACTION`
- After `COMMIT AND CHAIN`

In addition to knowing transaction state, it is useful to know transaction characteristics, so as to use the same characteristics if the transaction is moved to a different session. The following characteristics are relevant for this purpose:

```
READ ONLY
READ WRITE
ISOLATION LEVEL
WITH CONSISTENT SNAPSHOT
```

Available Session State Trackers

To support the session-tracking activities, notification is available for these types of client session state information:

- Changes to these attributes of client session state:
  - The default schema (database).
  - Session-specific values for system variables.
  - User-defined variables.
  - Temporary tables.
  - Prepared statements.

  The `session_track_state_change` system variable controls this tracker.
- Changes to the default schema name. The `session_track_schema` system variable controls this tracker.
- Changes to the session values of system variables. The `session_track_system_variables` system variable controls this tracker. The `SENSITIVE_VARIABLES_OBSERVER` privilege is required to track changes to the values of sensitive system variables.
- Available GTIDs. The `session_track_gtids` system variable controls this tracker.
- Information about transaction state and characteristics. The `session_track_transaction_info` system variable controls this tracker.

For descriptions of the tracker-related system variables, see Section 5.1.8, “Server System Variables”. Those system variables permit control over which change notifications occur, but do not provide a way to access notification information. Notification occurs in the MySQL client/server protocol, which includes tracker information in OK packets so that session state changes can be detected.

C API Session State Tracker Support

To enable client applications to extract state-change information from OK packets returned by the server, the MySQL C API provides a pair of functions:

- `mysql_session_track_get_first()` fetches the first part of the state-change information received from the server. See mysql_session_track_get_first().
- `mysql_session_track_get_next()` fetches any remaining state-change information received from the server. Following a successful call to `mysql_session_track_get_first()`, call this function repeatedly as long as it returns success. See mysql_session_track_get_next().

Test Suite Session State Tracker Support

The `mysqltest` program has `disable_session_track_info` and `enable_session_track_info` commands that control whether session tracker notifications occur. You can use these commands to see from the command line what notifications SQL statements produce. Suppose that a file `testscript` contains the following `mysqltest` script:

```
DROP TABLE IF EXISTS test.t1;
CREATE TABLE test.t1 (i INT, f FLOAT);
```

```
--enable_session_track_info
SET @@SESSION.session_track_schema=ON;
SET @@SESSION.session_track_system_variables='*';
SET @@SESSION.session_track_state_change=ON;
USE information_schema;
SET NAMES 'utf8mb4';
SET @@SESSION.session_track_transaction_info='CHARACTERISTICS';
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET TRANSACTION READ WRITE;
START TRANSACTION;
SELECT 1;
INSERT INTO test.t1 () VALUES();
INSERT INTO test.t1 () VALUES(1, RAND());
COMMIT;
```

Run the script as follows to see the information provided by the enabled trackers. For a description of the `Tracker:` information displayed by `mysqltest` for the various trackers, see mysql_session_track_get_first().

```
$> mysqltest < testscript
DROP TABLE IF EXISTS test.t1;
CREATE TABLE test.t1 (i INT, f FLOAT);
SET @@SESSION.session_track_schema=ON;
SET @@SESSION.session_track_system_variables='*';
-- Tracker : SESSION_TRACK_SYSTEM_VARIABLES
-- session_track_system_variables
-- *

SET @@SESSION.session_track_state_change=ON;
-- Tracker : SESSION_TRACK_SYSTEM_VARIABLES
-- session_track_state_change
-- ON

USE information_schema;
-- Tracker : SESSION_TRACK_SCHEMA
-- information_schema

-- Tracker : SESSION_TRACK_STATE_CHANGE
-- 1

SET NAMES 'utf8mb4';
-- Tracker : SESSION_TRACK_SYSTEM_VARIABLES
-- character_set_client
-- utf8mb4
-- character_set_connection
-- utf8mb4
-- character_set_results
-- utf8mb4

-- Tracker : SESSION_TRACK_STATE_CHANGE
-- 1

SET @@SESSION.session_track_transaction_info='CHARACTERISTICS';
-- Tracker : SESSION_TRACK_SYSTEM_VARIABLES
-- session_track_transaction_info
-- CHARACTERISTICS

-- Tracker : SESSION_TRACK_STATE_CHANGE
-- 1

-- Tracker : SESSION_TRACK_TRANSACTION_CHARACTERISTICS
--

-- Tracker : SESSION_TRACK_TRANSACTION_STATE
-- ________

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
-- Tracker : SESSION_TRACK_TRANSACTION_CHARACTERISTICS
-- SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

SET TRANSACTION READ WRITE;
-- Tracker : SESSION_TRACK_TRANSACTION_CHARACTERISTICS
```

```
-- SET TRANSACTION ISOLATION LEVEL SERIALIZABLE; SET TRANSACTION READ WRITE;

START TRANSACTION;
-- Tracker : SESSION_TRACK_TRANSACTION_CHARACTERISTICS
-- SET TRANSACTION ISOLATION LEVEL SERIALIZABLE; START TRANSACTION READ WRITE;

-- Tracker : SESSION_TRACK_TRANSACTION_STATE
-- T_______

SELECT 1;
1
1
-- Tracker : SESSION_TRACK_TRANSACTION_STATE
-- T_____S_

INSERT INTO test.t1 () VALUES();
-- Tracker : SESSION_TRACK_TRANSACTION_STATE
-- T___W_S_

INSERT INTO test.t1 () VALUES(1, RAND());
-- Tracker : SESSION_TRACK_TRANSACTION_STATE
-- T___WsS_

COMMIT;
-- Tracker : SESSION_TRACK_TRANSACTION_CHARACTERISTICS
--

-- Tracker : SESSION_TRACK_TRANSACTION_STATE
-- ________

ok
```

Preceding the `START TRANSACTION` statement, two `SET TRANSACTION` statements execute that set the isolation level and access mode characteristics for the next transaction. The `SESSION_TRACK_TRANSACTION_CHARACTERISTICS` value indicates those next-transaction values that have been set.

Following the `COMMIT` statement that ends the transaction, the `SESSION_TRACK_TRANSACTION_CHARACTERISTICS` value is reported as empty. This indicates that the next-transaction characteristics that were set preceding the start of the transaction have been reset, and that the session defaults apply. To track changes to those session defaults, track the session values of the `transaction_isolation` and `transaction_read_only` system variables.

To see information about GTIDs, enable the `SESSION_TRACK_GTIDS` tracker using the `session_track_gtids` system system variable.

5.1.19 The Server Shutdown Process

The server shutdown process takes place as follows:

1. The shutdown process is initiated.

   This can occur initiated several ways. For example, a user with the `SHUTDOWN` privilege can execute a `mysqladmin shutdown` command. `mysqladmin` can be used on any platform supported by MySQL. Other operating system-specific shutdown initiation methods are possible as well: The server shuts down on Unix when it receives a `SIGTERM` signal. A server running as a service on Windows shuts down when the services manager tells it to.

2. The server creates a shutdown thread if necessary.

   Depending on how shutdown was initiated, the server might create a thread to handle the shutdown process. If shutdown was requested by a client, a shutdown thread is created. If shutdown is the result of receiving a `SIGTERM` signal, the signal thread might handle shutdown itself, or it might create a separate thread to do so. If the server tries to create a shutdown thread and cannot (for example, if memory is exhausted), it issues a diagnostic message that appears in the error log:

```
Error: Can't create thread to kill server
```

3. The server stops accepting new connections.

   To prevent new activity from being initiated during shutdown, the server stops accepting new client connections by closing the handlers for the network interfaces to which it normally listens for connections: the TCP/IP port, the Unix socket file, the Windows named pipe, and shared memory on Windows.

4. The server terminates current activity.

   For each thread associated with a client connection, the server breaks the connection to the client and marks the thread as killed. Threads die when they notice that they are so marked. Threads for idle connections die quickly. Threads that currently are processing statements check their state periodically and take longer to die. For additional information about thread termination, see Section 13.7.8.4, “KILL Statement”, in particular for the instructions about killed `REPAIR TABLE` or `OPTIMIZE TABLE` operations on `MyISAM` tables.

   For threads that have an open transaction, the transaction is rolled back. If a thread is updating a nontransactional table, an operation such as a multiple-row `UPDATE` or `INSERT` may leave the table partially updated because the operation can terminate before completion.

   If the server is a replication source server, it treats threads associated with currently connected replicas like other client threads. That is, each one is marked as killed and exits when it next checks its state.

   If the server is a replica server, it stops the replication I/O and SQL threads, if they are active, before marking client threads as killed. The SQL thread is permitted to finish its current statement (to avoid causing replication problems), and then stops. If the SQL thread is in the middle of a transaction at this point, the server waits until the current replication event group (if any) has finished executing, or until the user issues a `KILL QUERY` or `KILL CONNECTION` statement. See also Section 13.4.2.11, “STOP SLAVE Statement”. Since nontransactional statements cannot be rolled back, in order to guarantee crash-safe replication, only transactional tables should be used.

   Note

   To guarantee crash safety on the replica, you must run the replica with `--relay-log-recovery` enabled.

   See also Section 17.2.4, “Relay Log and Replication Metadata Repositories”).

5. The server shuts down or closes storage engines.

   At this stage, the server flushes the table cache and closes all open tables.

   Each storage engine performs any actions necessary for tables that it manages. `InnoDB` flushes its buffer pool to disk (unless `innodb_fast_shutdown` is 2), writes the current LSN to the tablespace, and terminates its own internal threads. `MyISAM` flushes any pending index writes for a table.

6. The server exits.

To provide information to management processes, the server returns one of the exit codes described in the following list. The phrase in parentheses indicates the action taken by systemd in response to the code, for platforms on which systemd is used to manage the server.

- 0 = successful termination (no restart done)
- 1 = unsuccessful termination (no restart done)
- 2 = unsuccessful termination (restart done)

5.2 The MySQL Data Directory

Information managed by the MySQL server is stored under a directory known as the data directory. The following list briefly describes the items typically found in the data directory, with cross references for additional information:

- Data directory subdirectories. Each subdirectory of the data directory is a database directory and corresponds to a database managed by the server. All MySQL installations have certain standard databases:
  - The `mysql` directory corresponds to the `mysql` system schema, which contains information required by the MySQL server as it runs. This database contains data dictionary tables and system tables. See Section 5.3, “The mysql System Schema”.
  - The `performance_schema` directory corresponds to the Performance Schema, which provides information used to inspect the internal execution of the server at runtime. See Chapter 27, *MySQL Performance Schema*.
  - The `sys` directory corresponds to the `sys` schema, which provides a set of objects to help interpret Performance Schema information more easily. See Chapter 28, *MySQL sys Schema*.
  - The `ndbinfo` directory corresponds to the `ndbinfo` database that stores information specific to NDB Cluster (present only for installations built to include NDB Cluster). See Section 23.6.16, “ndbinfo: The NDB Cluster Information Database”.

  Other subdirectories correspond to databases created by users or applications.

Note

`INFORMATION_SCHEMA` is a standard database, but its implementation uses no corresponding database directory.

- Log files written by the server. See Section 5.4, “MySQL Server Logs”.
- `InnoDB` tablespace and log files. See Chapter 15, *The InnoDB Storage Engine*.
- Default/autogenerated SSL and RSA certificate and key files. See Section 6.3.3, “Creating SSL and RSA Certificates and Keys”.
- The server process ID file (while the server is running).
- The `mysqld-auto.cnf` file that stores persisted global system variable settings. See Section 13.7.6.1, “SET Syntax for Variable Assignment”.

Some items in the preceding list can be relocated elsewhere by reconfiguring the server. In addition, the `--datadir` option enables the location of the data directory itself to be changed. For a given MySQL installation, check the server configuration to determine whether items have been moved.

5.3 The mysql System Schema

The `mysql` schema is the system schema. It contains tables that store information required by the MySQL server as it runs. A broad categorization is that the `mysql` schema contains data dictionary tables that store database object metadata, and system tables used for other operational purposes. The following discussion further subdivides the set of system tables into smaller categories.

- Data Dictionary Tables
- Grant System Tables
- Object Information System Tables

- Log System Tables
- Server-Side Help System Tables
- Time Zone System Tables
- Replication System Tables
- Optimizer System Tables
- Miscellaneous System Tables

The remainder of this section enumerates the tables in each category, with cross references for additional information. Data dictionary tables and system tables use the `InnoDB` storage engine unless otherwise indicated.

`mysql` system tables and data dictionary tables reside in a single `InnoDB` tablespace file named `mysql.ibd` in the MySQL data directory. Previously, these tables were created in individual tablespace files in the `mysql` database directory.

Data-at-rest encryption can be enabled for the `mysql` system schema tablespace. For more information, see Section 15.13, “InnoDB Data-at-Rest Encryption”.

Data Dictionary Tables

These tables comprise the data dictionary, which contains metadata about database objects. For additional information, see Chapter 14, *MySQL Data Dictionary*.

Important

The data dictionary is new in MySQL 8.0. A data dictionary-enabled server entails some general operational differences compared to previous MySQL releases. For details, see Section 14.7, “Data Dictionary Usage Differences”. Also, for upgrades to MySQL 8.0 from MySQL 5.7, the upgrade procedure differs somewhat from previous MySQL releases and requires that you verify the upgrade readiness of your installation by checking specific prerequisites. For more information, see Section 2.10, “Upgrading MySQL”, particularly Section 2.10.5, “Preparing Your Installation for Upgrade”.

- `catalogs`: Catalog information.
- `character_sets`: Information about available character sets.
- `check_constraints`: Information about `CHECK` constraints defined on tables. See Section 13.1.20.6, “CHECK Constraints”.
- `collations`: Information about collations for each character set.
- `column_statistics`: Histogram statistics for column values. See Section 8.9.6, “Optimizer Statistics”.
- `column_type_elements`: Information about types used by columns.
- `columns`: Information about columns in tables.
- `dd_properties`: A table that identifies data dictionary properties, such as its version. The server uses this to determine whether the data dictionary must be upgraded to a newer version.
- `events`: Information about Event Scheduler events. See Section 25.4, “Using the Event Scheduler”. If the server is started with the `--skip-grant-tables` option, the event scheduler is disabled and events registered in the table do not run. See Section 25.4.2, “Event Scheduler Configuration”.
- `foreign_keys`, `foreign_key_column_usage`: Information about foreign keys.

- `index_column_usage`: Information about columns used by indexes.
- `index_partitions`: Information about partitions used by indexes.
- `index_stats`: Used to store dynamic index statistics generated when `ANALYZE TABLE` is executed.
- `indexes`: Information about table indexes.
- `innodb_ddl_log`: Stores DDL logs for crash-safe DDL operations.
- `parameter_type_elements`: Information about stored procedure and function parameters, and about return values for stored functions.
- `parameters`: Information about stored procedures and functions. See Section 25.2, “Using Stored Routines”.
- `resource_groups`: Information about resource groups. See Section 5.1.16, “Resource Groups”.
- `routines`: Information about stored procedures and functions. See Section 25.2, “Using Stored Routines”.
- `schemata`: Information about schemata. In MySQL, a schema is a database, so this table provides information about databases.
- `st_spatial_reference_systems`: Information about available spatial reference systems for spatial data.
- `table_partition_values`: Information about values used by table partitions.
- `table_partitions`: Information about partitions used by tables.
- `table_stats`: Information about dynamic table statistics generated when `ANALYZE TABLE` is executed.
- `tables`: Information about tables in databases.
- `tablespace_files`: Information about files used by tablespaces.
- `tablespaces`: Information about active tablespaces.
- `triggers`: Information about triggers.
- `view_routine_usage`: Information about dependencies between views and stored functions used by them.
- `view_table_usage`: Used to track dependencies between views and their underlying tables.

Data dictionary tables are invisible. They cannot be read with `SELECT`, do not appear in the output of `SHOW TABLES`, are not listed in the `INFORMATION_SCHEMA.TABLES` table, and so forth. However, in most cases there are corresponding `INFORMATION_SCHEMA` tables that can be queried. Conceptually, the `INFORMATION_SCHEMA` provides a view through which MySQL exposes data dictionary metadata. For example, you cannot select from the `mysql.schemata` table directly:

```
mysql> SELECT * FROM mysql.schemata;
ERROR 3554 (HY000): Access to data dictionary table 'mysql.schemata' is rejected.
```

Instead, select that information from the corresponding `INFORMATION_SCHEMA` table:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.SCHEMATA\G
*************************** 1. row ***************************
              CATALOG_NAME: def
               SCHEMA_NAME: mysql
DEFAULT_CHARACTER_SET_NAME: utf8mb4
```

```
    DEFAULT_COLLATION_NAME: utf8mb4_0900_ai_ci
                  SQL_PATH: NULL
        DEFAULT_ENCRYPTION: NO
*************************** 2. row ***************************
              CATALOG_NAME: def
               SCHEMA_NAME: information_schema
DEFAULT_CHARACTER_SET_NAME: utf8mb3
    DEFAULT_COLLATION_NAME: utf8mb3_general_ci
                  SQL_PATH: NULL
        DEFAULT_ENCRYPTION: NO
*************************** 3. row ***************************
              CATALOG_NAME: def
               SCHEMA_NAME: performance_schema
DEFAULT_CHARACTER_SET_NAME: utf8mb4
    DEFAULT_COLLATION_NAME: utf8mb4_0900_ai_ci
                  SQL_PATH: NULL
        DEFAULT_ENCRYPTION: NO
...
```

There is no Information Schema table that corresponds exactly to `mysql.indexes`, but `INFORMATION_SCHEMA.STATISTICS` contains much of the same information.

As of yet, there are no `INFORMATION_SCHEMA` tables that correspond exactly to `mysql.foreign_keys`, `mysql.foreign_key_column_usage`. The standard SQL way to obtain foreign key information is by using the `INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS` and `KEY_COLUMN_USAGE` tables; these tables are now implemented as views on the `foreign_keys`, `foreign_key_column_usage`, and other data dictionary tables.

Some system tables from before MySQL 8.0 have been replaced by data dictionary tables and are no longer present in the `mysql` system schema:

- The `events` data dictionary table supersedes the `event` table from before MySQL 8.0.
- The `parameters` and `routines` data dictionary tables together supersede the `proc` table from before MySQL 8.0.

Grant System Tables

These system tables contain grant information about user accounts and the privileges held by them. For additional information about the structure, contents, and purpose of the these tables, see Section 6.2.3, “Grant Tables”.

As of MySQL 8.0, the grant tables are `InnoDB` (transactional) tables. Previously, these were `MyISAM` (nontransactional) tables. The change of grant-table storage engine underlies an accompanying change in MySQL 8.0 to the behavior of account-management statements such as `CREATE USER` and `GRANT`. Previously, an account-management statement that named multiple users could succeed for some users and fail for others. The statements are now transactional and either succeed for all named users or roll back and have no effect if any error occurs.

Note

If MySQL is upgraded from an older version but the grant tables have not been upgraded from `MyISAM` to `InnoDB`, the server considers them read only and account-management statements produce an error. For upgrade instructions, see Section 2.10, “Upgrading MySQL”.

- `user`: User accounts, global privileges, and other nonprivilege columns.
- `global_grants`: Assignments of dynamic global privileges to users; see Static Versus Dynamic Privileges.
- `db`: Database-level privileges.
- `tables_priv`: Table-level privileges.

- `columns_priv`: Column-level privileges.
- `procs_priv`: Stored procedure and function privileges.
- `proxies_priv`: Proxy-user privileges.
- `default_roles`: This table lists default roles to be activated after a user connects and authenticates, or executes `SET ROLE DEFAULT`.
- `role_edges`: This table lists edges for role subgraphs.

  A given `user` table row might refer to a user account or a role. The server can distinguish whether a row represents a user account, a role, or both by consulting the `role_edges` table for information about relations between authentication IDs.
- `password_history`: Information about password changes.

Object Information System Tables

These system tables contain information about components, loadable functions, and server-side plugins:

- `component`: The registry for server components installed using `INSTALL COMPONENT`. Any components listed in this table are installed by a loader service during the server startup sequence. See Section 5.5.1, “Installing and Uninstalling Components”.
- `func`: The registry for loadable functions installed using `CREATE FUNCTION`. During the normal startup sequence, the server loads functions registered in this table. If the server is started with the `--skip-grant-tables` option, functions registered in the table are not loaded and are unavailable. See Section 5.7.1, “Installing and Uninstalling Loadable Functions”.

  Note

  Like the `mysql.func` system table, the Performance Schema `user_defined_functions` table lists loadable functions installed using `CREATE FUNCTION`. Unlike the `mysql.func` table, the `user_defined_functions` table also lists functions installed automatically by server components or plugins. This difference makes `user_defined_functions` preferable to `mysql.func` for checking which functions are installed. See Section 27.12.21.9, “The user_defined_functions Table”.
- `plugin`: The registry for server-side plugins installed using `INSTALL PLUGIN`. During the normal startup sequence, the server loads plugins registered in this table. If the server is started with the `--skip-grant-tables` option, plugins registered in the table are not loaded and are unavailable. See Section 5.6.1, “Installing and Uninstalling Plugins”.

Log System Tables

The server uses these system tables for logging:

- `general_log`: The general query log table.
- `slow_log`: The slow query log table.

Log tables use the `CSV` storage engine.

For more information, see Section 5.4, “MySQL Server Logs”.

Server-Side Help System Tables

These system tables contain server-side help information:

- `help_category`: Information about help categories.
- `help_keyword`: Keywords associated with help topics.
- `help_relation`: Mappings between help keywords and topics.
- `help_topic`: Help topic contents.

For more information, see Section 5.1.17, “Server-Side Help Support”.

Time Zone System Tables

These system tables contain time zone information:

- `time_zone`: Time zone IDs and whether they use leap seconds.
- `time_zone_leap_second`: When leap seconds occur.
- `time_zone_name`: Mappings between time zone IDs and names.
- `time_zone_transition`, `time_zone_transition_type`: Time zone descriptions.

For more information, see Section 5.1.15, “MySQL Server Time Zone Support”.

Replication System Tables

The server uses these system tables to support replication:

- `gtid_executed`: Table for storing GTID values. See mysql.gtid_executed Table.
- `ndb_binlog_index`: Binary log information for NDB Cluster replication. This table is created only if the server is built with `NDBCLUSTER` support. See Section 23.7.4, “NDB Cluster Replication Schema and Tables”.
- `slave_master_info`, `slave_relay_log_info`, `slave_worker_info`: Used to store replication information on replica servers. See Section 17.2.4, “Relay Log and Replication Metadata Repositories”.

All of the tables just listed use the `InnoDB` storage engine.

Optimizer System Tables

These system tables are for use by the optimizer:

- `innodb_index_stats`, `innodb_table_stats`: Used for `InnoDB` persistent optimizer statistics. See Section 15.8.10.1, “Configuring Persistent Optimizer Statistics Parameters”.
- `server_cost`, `engine_cost`: The optimizer cost model uses tables that contain cost estimate information about operations that occur during query execution. `server_cost` contains optimizer cost estimates for general server operations. `engine_cost` contains estimates for operations specific to particular storage engines. See Section 8.9.5, “The Optimizer Cost Model”.

Miscellaneous System Tables

Other system tables do not fit the preceding categories:

- `audit_log_filter`, `audit_log_user`: If MySQL Enterprise Audit is installed, these tables provide persistent storage of audit log filter definitions and user accounts. See Audit Log Tables.

- `firewall_group_allowlist`, `firewall_groups`, `firewall_memebership`, `firewall_users`, `firewall_whitelist`: If MySQL Enterprise Firewall is installed, these tables provide persistent storage for information used by the firewall. See Section 6.4.7, “MySQL Enterprise Firewall”.

- `servers`: Used by the `FEDERATED` storage engine. See Section 16.8.2.2, “Creating a FEDERATED Table Using CREATE SERVER”.

- `innodb_dynamic_metadata`: Used by the `InnoDB` storage engine to store fast-changing table metadata such as auto-increment counter values and index tree corruption flags. Replaces the data dictionary buffer table that resided in the `InnoDB` system tablespace.

5.4 MySQL Server Logs

MySQL Server has several logs that can help you find out what activity is taking place.

Log Type	Information Written to Log
Error log	Problems encountered starting, running, or stopping `mysqld`
General query log	Established client connections and statements received from clients
Binary log	Statements that change data (also used for replication)
Relay log	Data changes received from a replication source server
Slow query log	Queries that took more than `long_query_time` seconds to execute
DDL log (metadata log)	Metadata operations performed by DDL statements

By default, no logs are enabled, except the error log on Windows. (The DDL log is always created when required, and has no user-configurable options; see The DDL Log.) The following log-specific sections provide information about the server options that enable logging.

By default, the server writes files for all enabled logs in the data directory. You can force the server to close and reopen the log files (or in some cases switch to a new log file) by flushing the logs. Log flushing occurs when you issue a `FLUSH LOGS` statement; execute `mysqladmin` with a `flush-logs` or `refresh` argument; or execute `mysqldump` with a `--flush-logs` or `--master-data` option. See Section 13.7.8.3, “FLUSH Statement”, Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”, and Section 4.5.4, “mysqldump — A Database Backup Program”. In addition, the binary log is flushed when its size reaches the value of the `max_binlog_size` system variable.

You can control the general query and slow query logs during runtime. You can enable or disable logging, or change the log file name. You can tell the server to write general query and slow query entries to log tables, log files, or both. For details, see Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”, Section 5.4.3, “The General Query Log”, and Section 5.4.5, “The Slow Query Log”.

The relay log is used only on replicas, to hold data changes from the replication source server that must also be made on the replica. For discussion of relay log contents and configuration, see Section 17.2.4.1, “The Relay Log”.

For information about log maintenance operations such as expiration of old log files, see Section 5.4.6, “Server Log Maintenance”.

For information about keeping logs secure, see Section 6.1.2.3, “Passwords and Logging”.

5.4.1 Selecting General Query Log and Slow Query Log Output Destinations

MySQL Server provides flexible control over the destination of output written to the general query log and the slow query log, if those logs are enabled. Possible destinations for log entries are log files or the `general_log` and `slow_log` tables in the `mysql` system database. File output, table output, or both can be selected.

- Log Control at Server Startup
- Log Control at Runtime
- Log Table Benefits and Characteristics

Log Control at Server Startup

The `log_output` system variable specifies the destination for log output. Setting this variable does not in itself enable the logs; they must be enabled separately.

- If `log_output` is not specified at startup, the default logging destination is `FILE`.
- If `log_output` is specified at startup, its value is a list one or more comma-separated words chosen from `TABLE` (log to tables), `FILE` (log to files), or `NONE` (do not log to tables or files). `NONE`, if present, takes precedence over any other specifiers.

The `general_log` system variable controls logging to the general query log for the selected log destinations. If specified at server startup, `general_log` takes an optional argument of 1 or 0 to enable or disable the log. To specify a file name other than the default for file logging, set the `general_log_file` variable. Similarly, the `slow_query_log` variable controls logging to the slow query log for the selected destinations and setting `slow_query_log_file` specifies a file name for file logging. If either log is enabled, the server opens the corresponding log file and writes startup messages to it. However, further logging of queries to the file does not occur unless the `FILE` log destination is selected.

Examples:

- To write general query log entries to the log table and the log file, use `--log_output=TABLE,FILE` to select both log destinations and `--general_log` to enable the general query log.
- To write general and slow query log entries only to the log tables, use `--log_output=TABLE` to select tables as the log destination and `--general_log` and `--slow_query_log` to enable both logs.
- To write slow query log entries only to the log file, use `--log_output=FILE` to select files as the log destination and `--slow_query_log` to enable the slow query log. In this case, because the default log destination is `FILE`, you could omit the `log_output` setting.

Log Control at Runtime

The system variables associated with log tables and files enable runtime control over logging:

- The `log_output` variable indicates the current logging destination. It can be modified at runtime to change the destination.
- The `general_log` and `slow_query_log` variables indicate whether the general query log and slow query log are enabled (`ON`) or disabled (`OFF`). You can set these variables at runtime to control whether the logs are enabled.
- The `general_log_file` and `slow_query_log_file` variables indicate the names of the general query log and slow query log files. You can set these variables at server startup or at runtime to change the names of the log files.

- To disable or enable general query logging for the current session, set the session `sql_log_off` variable to `ON` or `OFF`. (This assumes that the general query log itself is enabled.)

Log Table Benefits and Characteristics

The use of tables for log output offers the following benefits:

- Log entries have a standard format. To display the current structure of the log tables, use these statements:

```
SHOW CREATE TABLE mysql.general_log;
SHOW CREATE TABLE mysql.slow_log;
```

- Log contents are accessible through SQL statements. This enables the use of queries that select only those log entries that satisfy specific criteria. For example, to select log contents associated with a particular client (which can be useful for identifying problematic queries from that client), it is easier to do this using a log table than a log file.

- Logs are accessible remotely through any client that can connect to the server and issue queries (if the client has the appropriate log table privileges). It is not necessary to log in to the server host and directly access the file system.

The log table implementation has the following characteristics:

- In general, the primary purpose of log tables is to provide an interface for users to observe the runtime execution of the server, not to interfere with its runtime execution.

- `CREATE TABLE`, `ALTER TABLE`, and `DROP TABLE` are valid operations on a log table. For `ALTER TABLE` and `DROP TABLE`, the log table cannot be in use and must be disabled, as described later.

- By default, the log tables use the `CSV` storage engine that writes data in comma-separated values format. For users who have access to the `.CSV` files that contain log table data, the files are easy to import into other programs such as spreadsheets that can process CSV input.

  The log tables can be altered to use the `MyISAM` storage engine. You cannot use `ALTER TABLE` to alter a log table that is in use. The log must be disabled first. No engines other than `CSV` or `MyISAM` are legal for the log tables.

  Log Tables and “Too many open files” Errors. If you select `TABLE` as a log destination and the log tables use the `CSV` storage engine, you may find that disabling and enabling the general query log or slow query log repeatedly at runtime results in a number of open file descriptors for the `.CSV` file, possibly resulting in a “Too many open files” error. To work around this issue, execute `FLUSH TABLES` or ensure that the value of `open_files_limit` is greater than the value of `table_open_cache_instances`.

- To disable logging so that you can alter (or drop) a log table, you can use the following strategy. The example uses the general query log; the procedure for the slow query log is similar but uses the `slow_log` table and `slow_query_log` system variable.

```
SET @old_log_state = @@GLOBAL.general_log;
SET GLOBAL general_log = 'OFF';
ALTER TABLE mysql.general_log ENGINE = MyISAM;
SET GLOBAL general_log = @old_log_state;
```

- `TRUNCATE TABLE` is a valid operation on a log table. It can be used to expire log entries.

- `RENAME TABLE` is a valid operation on a log table. You can atomically rename a log table (to perform log rotation, for example) using the following strategy:

```
USE mysql;
DROP TABLE IF EXISTS general_log2;
CREATE TABLE general_log2 LIKE general_log;
RENAME TABLE general_log TO general_log_backup, general_log2 TO general_log;
```

- `CHECK TABLE` is a valid operation on a log table.
- `LOCK TABLES` cannot be used on a log table.
- `INSERT`, `DELETE`, and `UPDATE` cannot be used on a log table. These operations are permitted only internally to the server itself.
- `FLUSH TABLES WITH READ LOCK` and the state of the `read_only` system variable have no effect on log tables. The server can always write to the log tables.
- Entries written to the log tables are not written to the binary log and thus are not replicated to replicas.
- To flush the log tables or log files, use `FLUSH TABLES` or `FLUSH LOGS`, respectively.
- Partitioning of log tables is not permitted.
- A `mysqldump` dump includes statements to recreate those tables so that they are not missing after reloading the dump file. Log table contents are not dumped.

5.4.2 The Error Log

This section discusses how to configure the MySQL server for logging of diagnostic messages to the error log. For information about selecting the error message character set and language, see Section 10.6, “Error Message Character Set”, and Section 10.12, “Setting the Error Message Language”.

The error log contains a record of `mysqld` startup and shutdown times. It also contains diagnostic messages such as errors, warnings, and notes that occur during server startup and shutdown, and while the server is running. For example, if `mysqld` notices that a table needs to be automatically checked or repaired, it writes a message to the error log.

Depending on error log configuration, error messages may also populate the Performance Schema `error_log` table, to provide an SQL interface to the log and enable its contents to be queried. See Section 27.12.21.1, “The error_log Table”.

On some operating systems, the error log contains a stack trace if `mysqld` exits abnormally. The trace can be used to determine where `mysqld` exited. See Section 5.9, “Debugging MySQL”.

If used to start `mysqld`, `mysqld_safe` may write messages to the error log. For example, when `mysqld_safe` notices abnormal `mysqld` exits, it restarts `mysqld` and writes a `mysqld restarted` message to the error log.

The following sections discuss aspects of configuring error logging.

5.4.2.1 Error Log Configuration

In MySQL 8.0, error logging uses the MySQL component architecture described at Section 5.5, “MySQL Components”. The error log subsystem consists of components that perform log event filtering and writing, as well as a system variable that configures which components to load and enable to achieve the desired logging result.

This section discusses how to load and enable components for error logging. For instructions specific to log filters, see Section 5.4.2.4, “Types of Error Log Filtering”. For instructions specific to the JSON and system log sinks, see Section 5.4.2.7, “Error Logging in JSON Format”, and Section 5.4.2.8, “Error Logging to the System Log”. For additional details about all available log components, see Section 5.5.3, “Error Log Components”.

Component-based error logging offers these features:

- Log events that can be filtered by filter components to affect the information available for writing.

- Log events that are output by sink (writer) components. Multiple sink components can be enabled, to write error log output to multiple destinations.
- Built-in filter and sink components that implement the default error log format.
- A loadable sink that enables logging in JSON format.
- A loadable sink that enables logging to the system log.
- System variables that control which log components to load and enable and how each component operates.

Error log configuration is described under the following topics in this section:

- The Default Error Log Configuration
- Error Log Configuration Methods
- Implicit Error Log Configuration
- Explicit Error Log Configuration
- Changing the Error Log Configuration Method
- Troubleshooting Configuration Issues
- Configuring Multiple Log Sinks
- Log Sink Performance Schema Support

The Default Error Log Configuration

The `log_error_services` system variable controls which loadable log components to load (as of MySQL 8.0.30) and which log components to enable for error logging. By default, `log_error_services` has this value:

```
mysql> SELECT @@GLOBAL.log_error_services;
+---------------------------------------+
| @@GLOBAL.log_error_services           |
+---------------------------------------+
| log_filter_internal; log_sink_internal |
+---------------------------------------+
```

That value indicates that log events first pass through the `log_filter_internal` filter component, then through the `log_sink_internal` sink component, both of which are built-in components. A filter modifies log events seen by components named later in the `log_error_services` value. A sink is a destination for log events. Typically, a sink processes log events into log messages that have a particular format and writes these messages to its associated output, such as a file or the system log.

The combination of `log_filter_internal` and `log_sink_internal` implements the default error log filtering and output behavior. The action of these components is affected by other server options and system variables:

- The output destination is determined by the `--log-error` option (and, on Windows, `--pid-file` and `--console`). These determine whether to write error messages to the console or a file and, if to a file, the error log file name. See Section 5.4.2.2, “Default Error Log Destination Configuration”.
- The `log_error_verbosity` and `log_error_suppression_list` system variables affect which types of log events `log_filter_internal` permits or suppresses. See Section 5.4.2.5, “Priority-Based Error Log Filtering (log_filter_internal)”.

When configuring `log_error_services`, be aware of the following characteristics:

- A list of log components may be delimited by semicolon or (as of MySQL 8.0.12) comma, optionally followed by space. A given setting cannot use both semicolon and comma separators. Component order is significant because the server executes components in the order listed.
- The final component in the `log_error_services` value cannot be a filter. This is an error because any changes it has on events would have no effect on output:

```
mysql> SET GLOBAL log_error_services = 'log_filter_internal';
ERROR 1231 (42000): Variable 'log_error_services' can't be set to the value
of 'log_filter_internal'
```

  To correct the problem, include a sink at the end of the value:

```
mysql> SET GLOBAL log_error_services = 'log_filter_internal; log_sink_internal';
```

- The order of components named in `log_error_services` is significant, particularly with respect to the relative order of filters and sinks. Consider this `log_error_services` value:

```
log_filter_internal; log_sink_1; log_sink_2
```

  In this case, log events pass to the built-in filter, then to the first sink, then to the second sink. Both sinks receive the filtered log events.

  Compare that to this `log_error_services` value:

```
log_sink_1; log_filter_internal; log_sink_2
```

  In this case, log events pass to the first sink, then to the built-in filter, then to the second sink. The first sink receives unfiltered events. The second sink receives filtered events. You might configure error logging this way if you want one log that contains messages for all log events, and another log that contains messages only for a subset of log events.

Error Log Configuration Methods

Error log configuration involves loading and enabling error log components as necessary and performing component-specific configuration.

There are two error log configuration methods, *implicit* and *explicit*. It is recommended that one configuration method is selected and used exclusively. Using both methods can result in warnings at startup. For more information, see Troubleshooting Configuration Issues.

- *Implicit Error Log Configuration* (introduced in MySQL 8.0.30)

  This configuration method loads and enables the log components defined by the `log_error_services` variable. Loadable components that are not already loaded are loaded implicitly at startup before the `InnoDB` storage engine is fully available. This configuration method has the following advantages:

  - Log components are loaded early in the startup sequence, before the `InnoDB` storage engine, making logged information available sooner.
  - It avoids loss of buffered log information should a failure occur during startup.
  - Installing error log components using `INSTALL COMPONENT` is not required, simplifying error log configuration.

  To use this method, see Implicit Error Log Configuration.

- *Explicit Error Log Configuration*

Note

This configuration method is supported for backward compatibility. The implicit configuration method, introduced in MySQL 8.0.30, is recommended.

This configuration method requires loading error log components using `INSTALL COMPONENT` and then configuring `log_error_services` to enable the log components. `INSTALL COMPONENT` adds the component to the `mysql.component` table (an `InnoDB` table), and the components to load at startup are read from this table, which is only accessible after `InnoDB` is initialized.

Logged information is buffered during the startup sequence while the `InnoDB` storage engine is initialized, which is sometimes prolonged by operations such as recovery and data dictionary upgrade that occur during the `InnoDB` startup sequence.

To use this method, see Explicit Error Log Configuration.

Implicit Error Log Configuration

This procedure describes how to load and enable error logging components implicitly using `log_error_services`. For a discussion of error log configuration methods, see Error Log Configuration Methods.

To load and enable error logging components implicitly:

1. List the error log components in the `log_error_services` value.

   To load and enable the error log components at server startup, set `log_error_services` in an option file. The following example configures the use of the JSON log sink (`log_sink_json`) in addition to the built-in log filter and sink (`log_filter_internal`, `log_sink_internal`).

   ```
   [mysqld]
   log_error_services='log_filter_internal; log_sink_internal; log_sink_json'
   ```

   Note

   To use the JSON log sink (`log_sink_syseventlog`) instead of the default sink (`log_sink_internal`), you would replace `log_sink_internal` with `log_sink_json`.

   To load and enable the component immediately and for subsequent restarts, set `log_error_services` using `SET PERSIST`:

   ```
   SET PERSIST log_error_services = 'log_filter_internal; log_sink_internal; log_sink_json';
   ```

2. If the error log component exposes any system variables that must be set for component initialization to succeed, assign those variables appropriate values. You can set these variables in an option file or using `SET PERSIST`.

   Important

   When implementing an implicit configuration, set `log_error_services` first to load a component and expose its system variables, and then set component system variables afterward. This configuration order is required regardless of whether variable assignment is performed on the command-line, in an option file, or using `SET PERSIST`.

To disable a log component, remove it from the `log_error_services` value. Also remove any associated component variables settings that you have defined.

Note

Loading a log component implicitly using `log_error_services` has no effect on the `mysql.component` table. It does not add the component to the `mysql.component` table, nor does it remove a component previously installed using `INSTALL COMPONENT` from the `mysql.component` table.

Explicit Error Log Configuration

This procedure describes how to load and enable error logging components explicitly by loading components using `INSTALL COMPONENT` and then enabling using `log_error_services`. For a discussion of error log configuration methods, see Error Log Configuration Methods.

To load and enable error logging components explicitly:

1. Load the component using `INSTALL COMPONENT` (unless it is built in or already loaded). For example, to load the JSON log sink, issue the following statement:

   ```
   INSTALL COMPONENT 'file://component_log_sink_json';
   ```

   Loading a component using `INSTALL COMPONENT` registers it in the `mysql.component` system table so that the server loads it automatically for subsequent startups, after `InnoDB` is initialized.

   The URN to use when loading a log component with `INSTALL COMPONENT` is the component name prefixed with `file://component_`. For example, for the `log_sink_json` component, the corresponding URN is `file://component_log_sink_json`. For error log component URNs, see Section 5.5.3, “Error Log Components”.

2. If the error log component exposes any system variables that must be set for component initialization to succeed, assign those variables appropriate values. You can set these variables in an option file or using `SET PERSIST`.

3. Enable the component by listing it in the `log_error_services` value.

   Important

   From MySQL 8.0.30, when loading log components explicitly using `INSTALL COMPONENT`, do not persist or set `log_error_services` in an option file, which loads log components implicitly at startup. Instead, enable log components at runtime using a `SET GLOBAL` statement.

   The following example configures the use of the JSON log sink (`log_sink_json`) in addition to the built-in log filter and sink (`log_filter_internal`, `log_sink_internal`).

   ```
   SET GLOBAL log_error_services = 'log_filter_internal; log_sink_internal; log_sink_json';
   ```

   Note

   To use the JSON log sink (`log_sink_syseventlog`) instead of the default sink (`log_sink_internal`), you would replace `log_sink_internal` with `log_sink_json`.

To disable a log component, remove it from the `log_error_services` value. Then, if the component is loadable and you also want to unload it, use `UNINSTALL COMPONENT`. Also remove any associated component variables settings that you have defined.

Attempts to use `UNINSTALL COMPONENT` to unload a loadable component that is still named in the `log_error_services` value produce an error.

Changing the Error Log Configuration Method

If you have previously loaded error log components explicitly using `INSTALL COMPONENT` and want to switch to an implicit configuration, as described in Implicit Error Log Configuration, the following steps are recommended:

1. Set `log_error_services` back to its default configuration.

   ```
   SET GLOBAL log_error_services = 'log_filter_internal,log_sink_internal';
   ```

2. Use `UNINSTALL COMPONENT` to uninstall any loadable logging components that you installed previously. For example, if you installed the JSON log sink previously, uninstall it as shown:

   ```
   UNINSTALL COMPONENT 'file://component_log_sink_json';
   ```

3. Remove any component variable settings for the uninstalled component. For example, if component variables were set in an option file, remove the settings from the option file. If component variables were set using `SET PERSIST`, use `RESET PERSIST` to clear the settings.

4. Follow the steps in Implicit Error Log Configuration to reimplement your configuration.

If you need to revert from an implicit configuration to an explicit configuration, perform the following steps:

1. Set `log_error_services` back to its default configuration to unload implicitly loaded log components.

   ```
   SET GLOBAL log_error_services = 'log_filter_internal,log_sink_internal';
   ```

2. Remove any component variable settings associated with the uninstalled components. For example, if component variables were set in an option file, remove the settings from the option file. If component variables were set using `SET PERSIST`, use `RESET PERSIST` to clear the settings.

3. Restart the server to uninstall the log components that were implicitly loaded.

4. Follow the steps in Explicit Error Log Configuration to reimplement your configuration.

Troubleshooting Configuration Issues

From MySQL 8.0.30, log components listed in the `log_error_services` value at startup are loaded implicitly early in the MySQL Server startup sequence. If the log component was loaded previously using `INSTALL COMPONENT`, the server attempts to load the component again later in the startup sequence, which produces the following warning:

```
Cannot load component from specified URN: 'file://component_component_name'
```

You can check for this warning in the error log or by querying the Performance Schema `error_log` table using the following query:

```
SELECT error_code, data
  FROM performance_schema.error_log
 WHERE data LIKE "%'file://component_%"
   AND error_code="MY-013129" AND data LIKE "%MY-003529%";
```

To prevent this warning, follow the instructions in Changing the Error Log Configuration Method to adjust your error log configuration. Either an implicit or explicit error log configuration should be used, but not both.

A similar error occurs when attempting to explicitly load a component that was implicitly loaded at startup. For example, if `log_error_services` lists the JSON log sink component, that component is implicitly loaded at startup. Attempting to explicitly load the same component later returns this error:

```
mysql> INSTALL COMPONENT 'file://component_log_sink_json';
ERROR 3529 (HY000): Cannot load component from specified URN: 'file://component_log_sink_json'.
```

Configuring Multiple Log Sinks

It is possible to configure multiple log sinks, which enables sending output to multiple destinations. To enable the JSON log sink in addition to (rather than instead of) the default sink, set the `log_error_services` value like this:

```
SET GLOBAL log_error_services = 'log_filter_internal; log_sink_internal; log_sink_json';
```

To revert to using only the default sink and unload the system log sink, execute these statements:

```
SET GLOBAL log_error_services = 'log_filter_internal; log_sink_internal;
```

```
UNINSTALL COMPONENT 'file://component_log_sink_json';
```

Log Sink Performance Schema Support

If enabled log components include a sink that provides Performance Schema support, events written to the error log are also written to the Performance Schema `error_log` table. This enables examining error log contents using SQL queries. Currently, the traditional-format `log_sink_internal` and JSON-format `log_sink_json` sinks support this capability. See Section 27.12.21.1, “The error_log Table”.

5.4.2.2 Default Error Log Destination Configuration

This section describes which server options configure the default error log destination, which can be the console or a named file. It also indicates which log sink components base their own output destination on the default destination.

In this discussion, “console” means `stderr`, the standard error output. This is your terminal or console window unless the standard error output has been redirected to a different destination.

The server interprets options that determine the default error log destination somewhat differently for Windows and Unix systems. Be sure to configure the destination using the information appropriate to your platform. After the server interprets the default error log destination options, it sets the `log_error` system variable to indicate the default destination, which affects where several log sink components write error messages. The following sections address these topics.

- Default Error Log Destination on Windows
- Default Error Log Destination on Unix and Unix-Like Systems
- How the Default Error Log Destination Affects Log Sinks

Default Error Log Destination on Windows

On Windows, `mysqld` uses the `--log-error`, `--pid-file`, and `--console` options to determine whether the default error log destination is the console or a file, and, if a file, the file name:

- If `--console` is given, the default destination is the console. (`--console` takes precedence over `--log-error` if both are given, and the following items regarding `--log-error` do not apply.)
- If `--log-error` is not given, or is given without naming a file, the default destination is a file named *`host_name`*`.err` in the data directory, unless the `--pid-file` option is specified. In that case, the file name is the PID file base name with a suffix of `.err` in the data directory.
- If `--log-error` is given to name a file, the default destination is that file (with an `.err` suffix added if the name has no suffix). The file location is under the data directory unless an absolute path name is given to specify a different location.

If the default error log destination is the console, the server sets the `log_error` system variable to `stderr`. Otherwise, the default destination is a file and the server sets `log_error` to the file name.

Default Error Log Destination on Unix and Unix-Like Systems

On Unix and Unix-like systems, `mysqld` uses the `--log-error` option to determine whether the default error log destination is the console or a file, and, if a file, the file name:

- If `--log-error` is not given, the default destination is the console.
- If `--log-error` is given without naming a file, the default destination is a file named *`host_name`*`.err` in the data directory.
- If `--log-error` is given to name a file, the default destination is that file (with an `.err` suffix added if the name has no suffix). The file location is under the data directory unless an absolute path name is given to specify a different location.

- If `--log-error` is given in an option file in a `[mysqld]`, `[server]`, or `[mysqld_safe]` section, on systems that use `mysqld_safe` to start the server, `mysqld_safe` finds and uses the option, and passes it to `mysqld`.

Note

It is common for Yum or APT package installations to configure an error log file location under `/var/log` with an option like `log-error=/var/log/mysqld.log` in a server configuration file. Removing the path name from the option causes the *`host_name`*`.err` file in the data directory to be used.

If the default error log destination is the console, the server sets the `log_error` system variable to `stderr`. Otherwise, the default destination is a file and the server sets `log_error` to the file name.

How the Default Error Log Destination Affects Log Sinks

After the server interprets the error log destination configuration options, it sets the `log_error` system variable to indicate the default error log destination. Log sink components may base their own output destination on the `log_error` value, or determine their destination independently of `log_error`

If `log_error` is `stderr`, the default error log destination is the console, and log sinks that base their output destination on the default destination also write to the console:

- `log_sink_internal`, `log_sink_json`, `log_sink_test`: These sinks write to the console. This is true even for sinks such as `log_sink_json` that can be enabled multiple times; all instances write to the console.
- `log_sink_syseventlog`: This sink writes to the system log, regardless of the `log_error` value.

If `log_error` is not `stderr`, the default error log destination is a file and `log_error` indicates the file name. Log sinks that base their output destination on the default destination base output file naming on that file name. (A sink might use exactly that name, or it might use some variant thereof.) Suppose that the `log_error` value *`file_name`*. Then log sinks use the name like this:

- `log_sink_internal`, `log_sink_test`: These sinks write to *`file_name`*.
- `log_sink_json`: Successive instances of this sink named in the `log_error_services` value write to files named *`file_name`* plus a numbered *`.NN`*`.json` suffix: *`file_name`*`.00.json`, *`file_name`*`.01.json`, and so forth.
- `log_sink_syseventlog`: This sink writes to the system log, regardless of the `log_error` value.

5.4.2.3 Error Event Fields

Error events intended for the error log contain a set of fields, each of which consists of a key/value pair. An event field may be classified as core, optional, or user-defined:

- A core field is set up automatically for error events. However, its presence in the event during event processing is not guaranteed because a core field, like any type of field, may be unset by a log filter. If this happens, the field cannot be found by subsequent processing within that filter and by components that execute after the filter (such as log sinks).
- An optional field is normally absent but may be present for certain event types. When present, an optional field provides additional event information as appropriate and available.
- A user-defined field is any field with a name that is not already defined as a core or optional field. A user-defined field does not exist until created by a log filter.

As implied by the preceding description, any given field may be absent during event processing, either because it was not present in the first place, or was discarded by a filter. For log sinks, the effect of field absence is sink specific. For example, a sink might omit the field from the log message, indicate

that the field is missing, or substitute a default. When in doubt, test: use a filter that unsets the field, then check what the log sink does with it.

The following sections describe the core and optional error event fields. For individual log filter components, there may be additional filter-specific considerations for these fields, or filters may add user-defined fields not listed here. For details, see the documentation for specific filters.

- Core Error Event Fields
- Optional Error Event Fields

Core Error Event Fields

These error event fields are core fields:

- `time`

  The event timestamp, with microsecond precision.

- `msg`

  The event message string.

- `prio`

  The event priority, to indicate a system, error, warning, or note/information event. This field corresponds to severity in `syslog`. The following table shows the possible priority levels.

Event Type	Numeric Priority
System event	0
Error event	1
Warning event	2
Note/information event	3

  The `prio` value is numeric. Related to it, an error event may also include an optional `label` field representing the priority as a string. For example, an event with a `prio` value of 2 may have a `label` value of `'Warning'`.

  Filter components may include or drop error events based on priority, except that system events are mandatory and cannot be dropped.

  In general, message priorities are determined as follows:

  Is the situation or event actionable?

  - Yes: Is the situation or event ignorable?
    - Yes: Priority is warning.
    - No: Priority is error.
  - No: Is the situation or event mandatory?
    - Yes: Priority is system.
    - No: Priority is note/information.

- `err_code`

  The event error code, as a number (for example, `1022`).

- `err_symbol`

  The event error symbol, as a string (for example, `'ER_DUP_KEY'`).

- `SQL_state`

  The event SQLSTATE value, as a string (for example, `'23000'`).

- `subsystem`

  The subsystem in which the event occurred. Possible values are `InnoDB` (the `InnoDB` storage engine), `Repl` (the replication subsystem), `Server` (otherwise).

Optional Error Event Fields

Optional error event fields fall into the following categories:

- Additional information about the error, such as the error signaled by the operating system or the error label:
  - `OS_errno`

    The operating system error number.

  - `OS_errmsg`

    The operating system error message.

  - `label`

    The label corresponding to the `prio` value, as a string.

- Identification of the client for which the event occurred:
  - `user`

    The client user.

  - `host`

    The client host.

  - `thread`

    The ID of the thread within `mysqld` responsible for producing the error event. This ID indicates which part of the server produced the event, and is consistent with general query log and slow query log messages, which include the connection thread ID.

  - `query_id`

    The query ID.

- Debugging information:
  - `source_file`

    The source file in which the event occurred, without any leading path.

  - `source_line`

    The line within the source file at which the event occurred.

  - `function`

The function in which the event occurred.

- `component`

  The component or plugin in which the event occurred.

5.4.2.4 Types of Error Log Filtering

Error log configuration normally includes one log filter component and one or more log sink components. For error log filtering, MySQL offers a choice of components:

- `log_filter_internal`: This filter component provides error log filtering based on log event priority and error code, in combination with the `log_error_verbosity` and `log_error_suppression_list` system variables. `log_filter_internal` is built in and enabled by default. See Section 5.4.2.5, “Priority-Based Error Log Filtering (log_filter_internal)”.
- `log_filter_dragnet`: This filter component provides error log filtering based on user-supplied rules, in combination with the `dragnet.log_error_filter_rules` system variable. See Section 5.4.2.6, “Rule-Based Error Log Filtering (log_filter_dragnet)”.

5.4.2.5 Priority-Based Error Log Filtering (log_filter_internal)

The `log_filter_internal` log filter component implements a simple form of log filtering based on error event priority and error code. To affect how `log_filter_internal` permits or suppresses error, warning, and information events intended for the error log, set the `log_error_verbosity` and `log_error_suppression_list` system variables.

`log_filter_internal` is built in and enabled by default. If this filter is disabled, `log_error_verbosity` and `log_error_suppression_list` have no effect, so filtering must be performed using another filter service instead where desired (for example, with individual filter rules when using `log_filter_dragnet`). For information about filter configuration, see Section 5.4.2.1, “Error Log Configuration”.

- Verbosity Filtering
- Suppression-List Filtering
- Verbosity and Suppression-List Interaction

Verbosity Filtering

Events intended for the error log have a priority of `ERROR`, `WARNING`, or `INFORMATION`. The `log_error_verbosity` system variable controls verbosity based on which priorities to permit for messages written to the log, as shown in the following table.

log_error_verbosity Value	Permitted Message Priorities
1	`ERROR`
2	`ERROR`, `WARNING`
3	`ERROR`, `WARNING`, `INFORMATION`

If `log_error_verbosity` is 2 or greater, the server logs messages about statements that are unsafe for statement-based logging. If the value is 3, the server logs aborted connections and access-denied errors for new connection attempts. See Section B.3.2.9, “Communication Errors and Aborted Connections”.

If you use replication, a `log_error_verbosity` value of 2 or greater is recommended, to obtain more information about what is happening, such as messages about network failures and reconnections.

If log_error_verbosity is 2 or greater on a replica, the replica prints messages to the error log to provide information about its status, such as the binary log and relay log coordinates where it starts its job, when it is switching to another relay log, when it reconnects after a disconnect, and so forth.

There is also a message priority of SYSTEM that is not subject to verbosity filtering. System messages about non-error situations are printed to the error log regardless of the log_error_verbosity value. These messages include startup and shutdown messages, and some significant changes to settings.

In the MySQL error log, system messages are labeled as “System”. Other log sinks might or might not follow the same convention, and in the resulting logs, system messages might be assigned the label used for the information priority level, such as “Note” or “Information”. If you apply any additional filtering or redirection for logging based on the labeling of messages, system messages do not override your filter, but are handled by it in the same way as other messages.

Suppression-List Filtering

The log_error_suppression_list system variable applies to events intended for the error log and specifies which events to suppress when they occur with a priority of WARNING or INFORMATION. For example, if a particular type of warning is considered undesirable “noise” in the error log because it occurs frequently but is not of interest, it can be suppressed. log_error_suppression_list does not suppress messages with a priority of ERROR or SYSTEM.

The log_error_suppression_list value may be the empty string for no suppression, or a list of one or more comma-separated values indicating the error codes to suppress. Error codes may be specified in symbolic or numeric form. A numeric code may be specified with or without the MY- prefix. Leading zeros in the numeric part are not significant. Examples of permitted code formats:

```
ER_SERVER_SHUTDOWN_COMPLETE
MY-000031
000031
MY-31
31
```

For readability and portability, symbolic values are preferable to numeric values.

Although codes to be suppressed can be expressed in symbolic or numeric form, the numeric value of each code must be in a permitted range:

- 1 to 999: Global error codes that are used by the server as well as by clients.
- 10000 and higher: Server error codes intended to be written to the error log (not sent to clients).

In addition, each error code specified must actually be used by MySQL. Attempts to specify a code not within a permitted range or within a permitted range but not used by MySQL produce an error and the log_error_suppression_list value remains unchanged.

For information about error code ranges and the error symbols and numbers defined within each range, see Section B.1, “Error Message Sources and Elements”, and MySQL 8.0 Error Message Reference.

The server can generate messages for a given error code at differing priorities, so suppression of a message associated with an error code listed in log_error_suppression_list depends on its priority. Suppose that the variable has a value of 'ER_PARSER_TRACE,MY-010001,10002'. Then log_error_suppression_list has these effects on messages for those codes:

- Messages generated with a priority of WARNING or INFORMATION are suppressed.
- Messages generated with a priority of ERROR or SYSTEM are not suppressed.

Verbosity and Suppression-List Interaction

The effect of log_error_verbosity combines with that of log_error_suppression_list. Consider a server started with these settings:

```
[mysqld]
```

```
log_error_verbosity=2     # error and warning messages only
log_error_suppression_list='ER_PARSER_TRACE,MY-010001,10002'
```

In this case, `log_error_verbosity` permits messages with `ERROR` or `WARNING` priority and discards messages with `INFORMATION` priority. Of the nondiscarded messages, `log_error_suppression_list` discards messages with `WARNING` priority and any of the named error codes.

Note

The `log_error_verbosity` value of 2 shown in the example is also its default value, so the effect of this variable on `INFORMATION` messages is as just described by default, without an explicit setting. You must set `log_error_verbosity` to 3 if you want `log_error_suppression_list` to affect messages with `INFORMATION` priority.

Consider a server started with this setting:

```
[mysqld]
log_error_verbosity=1     # error messages only
```

In this case, `log_error_verbosity` permits messages with `ERROR` priority and discards messages with `WARNING` or `INFORMATION` priority. Setting `log_error_suppression_list` has no effect because all error codes it might suppress are already discarded due to the `log_error_verbosity` setting.

5.4.2.6 Rule-Based Error Log Filtering (log_filter_dragnet)

The `log_filter_dragnet` log filter component enables log filtering based on user-defined rules.

To enable the `log_filter_dragnet` filter, first load the filter component, then modify the `log_error_services` value. The following example enables `log_filter_dragnet` in combination with the built-in log sink:

```
INSTALL COMPONENT 'file://component_log_filter_dragnet';
SET GLOBAL log_error_services = 'log_filter_dragnet; log_sink_internal';
```

To set `log_error_services` to take effect at server startup, use the instructions at Section 5.4.2.1, “Error Log Configuration”. Those instructions apply to other error-logging system variables as well.

With `log_filter_dragnet` enabled, define its filter rules by setting the `dragnet.log_error_filter_rules` system variable. A rule set consists of zero or more rules, where each rule is an `IF` statement terminated by a period (`.`) character. If the variable value is empty (zero rules), no filtering occurs.

Example 1. This rule set drops information events, and, for other events, removes the `source_line` field:

```
SET GLOBAL dragnet.log_error_filter_rules =
  'IF prio>=INFORMATION THEN drop. IF EXISTS source_line THEN unset source_line.';
```

The effect is similar to the filtering performed by the `log_sink_internal` filter with a setting of `log_error_verbosity=2`.

For readability, you might find it preferable to list the rules on separate lines. For example:

```
SET GLOBAL dragnet.log_error_filter_rules = '
  IF prio>=INFORMATION THEN drop.
  IF EXISTS source_line THEN unset source_line.
';
```

Example 2: This rule limits information events to no more than one per 60 seconds:

```
SET GLOBAL dragnet.log_error_filter_rules =
  'IF prio>=INFORMATION THEN throttle 1/60.';
```

Once you have the filtering configuration set up as you desire, consider assigning `dragnet.log_error_filter_rules` using `SET PERSIST` rather than `SET GLOBAL` to make the setting persist across server restarts. Alternatively, add the setting to the server option file.

To stop using the filtering language, first remove it from the set of error logging components. Usually this means using a different filter component rather than no filter component. For example:

```
SET GLOBAL log_error_services = 'log_filter_internal; log_sink_internal';
```

Again, consider using `SET PERSIST` rather than `SET GLOBAL` to make the setting persist across server restarts.

Then uninstall the filter `log_filter_dragnet` component:

```
UNINSTALL COMPONENT 'file://component_log_filter_dragnet';
```

The following sections describe aspects of `log_filter_dragnet` operation in more detail:

- Grammar for log_filter_dragnet Rule Language
- Actions for log_filter_dragnet Rules
- Field References in log_filter_dragnet Rules

Grammar for log_filter_dragnet Rule Language

The following grammar defines the language for `log_filter_dragnet` filter rules. Each rule is an `IF` statement terminated by a period (`.`) character. The language is not case-sensitive.

```
rule:
    IF condition THEN action
    [ELSEIF condition THEN action] ...
    [ELSE action]
    .

condition: {
    field comparator value
  | [NOT] EXISTS field
  | condition {AND | OR}  condition
}

action: {
    drop
  | throttle {count | count / window_size}
  | set field [:= | =] value
  | unset [field]
}

field: {
    core_field
  | optional_field
  | user_defined_field
}

core_field: {
    time
  | msg
  | prio
  | err_code
  | err_symbol
  | SQL_state
  | subsystem
}

optional_field: {
    OS_errno
  | OS_errmsg
  | label
  | user
  | host
```

```
  | thread
  | query_id
  | source_file
  | source_line
  | function
  | component
}

user_defined_field:
    sequence of characters in [a-zA-Z0-9_] class

comparator: {== | != | <> | >= | => | <= | =< | < | >}

value: {
    string_literal
  | integer_literal
  | float_literal
  | error_symbol
  | priority
}

count: integer_literal
window_size: integer_literal

string_literal:
    sequence of characters quoted as '...' or "..."

integer_literal:
    sequence of characters in [0-9] class

float_literal:
    integer_literal[.integer_literal]

error_symbol:
    valid MySQL error symbol such as ER_ACCESS_DENIED_ERROR or ER_STARTUP

priority: {
    ERROR
  | WARNING
  | INFORMATION
}
```

Simple conditions compare a field to a value or test field existence. To construct more complex conditions, use the `AND` and `OR` operators. Both operators have the same precedence and evaluate left to right.

To escape a character within a string, precede it by a backslash (`\`). A backslash is required to include backslash itself or the string-quoting character, optional for other characters.

For convenience, `log_filter_dragnet` supports symbolic names for comparisons to certain fields. For readability and portability, symbolic values are preferable (where applicable) to numeric values.

- Event priority values 1, 2, and 3 can be specified as `ERROR`, `WARNING`, and `INFORMATION`. Priority symbols are recognized only in comparisons with the `prio` field. These comparisons are equivalent:

  ```
  IF prio == INFORMATION THEN ...
  IF prio == 3 THEN ...
  ```

- Error codes can be specified in numeric form or as the corresponding error symbol. For example, `ER_STARTUP` is the symbolic name for error `1408`, so these comparisons are equivalent:

  ```
  IF err_code == ER_STARTUP THEN ...
  IF err_code == 1408 THEN ...
  ```

  Error symbols are recognized only in comparisons with the `err_code` field and user-defined fields.

  To find the error symbol corresponding to a given error code number, use one of these methods:

  - Check the list of server errors at Server Error Message Reference.

- Use the `perror` command. Given an error number argument, `perror` displays information about the error, including its symbol.

Suppose that a rule set with error numbers looks like this:

```
IF err_code == 10927 OR err_code == 10914 THEN drop.
IF err_code == 1131 THEN drop.
```

Using `perror`, determine the error symbols:

```
$> perror 10927 10914 1131
MySQL error code MY-010927 (ER_ACCESS_DENIED_FOR_USER_ACCOUNT_LOCKED):
Access denied for user '%-.48s'@'%-.64s'. Account is locked.
MySQL error code MY-010914 (ER_ABORTING_USER_CONNECTION):
Aborted connection %u to db: '%-.192s' user: '%-.48s' host:
'%-.64s' (%-.64s).
MySQL error code MY-001131 (ER_PASSWORD_ANONYMOUS_USER):
You are using MySQL as an anonymous user and anonymous users
are not allowed to change passwords
```

Substituting error symbols for numbers, the rule set becomes:

```
IF err_code == ER_ACCESS_DENIED_FOR_USER_ACCOUNT_LOCKED
  OR err_code == ER_ABORTING_USER_CONNECTION THEN drop.
IF err_code == ER_PASSWORD_ANONYMOUS_USER THEN drop.
```

Symbolic names can be specified as quoted strings for comparison with string fields, but in such cases the names are strings that have no special meaning and `log_filter_dragnet` does not resolve them to the corresponding numeric value. Also, typos may go undetected, whereas an error occurs immediately on `SET` for attempts to use an unquoted symbol unknown to the server.

Actions for log_filter_dragnet Rules

`log_filter_dragnet` supports these actions in filter rules:

- `drop`: Drop the current log event (do not log it).
- `throttle`: Apply rate limiting to reduce log verbosity for events matching particular conditions. The argument indicates a rate, in the form *`count`* or *`count`*/*`window_size`*. The *`count`* value indicates the permitted number of event occurrences to log per time window. The *`window_size`* value is the time window in seconds; if omitted, the default window is 60 seconds. Both values must be integer literals.

  This rule throttles plugin-shutdown messages to 5 occurrences per 60 seconds:

  ```
  IF err_code == ER_PLUGIN_SHUTTING_DOWN_PLUGIN THEN throttle 5.
  ```

  This rule throttles errors and warnings to 1000 occurrences per hour and information messages to 100 occurrences per hour:

  ```
  IF prio <= INFORMATION THEN throttle 1000/3600 ELSE throttle 100/3600.
  ```

- `set`: Assign a value to a field (and cause the field to exist if it did not already). In subsequent rules, `EXISTS` tests against the field name are true, and the new value can be tested by comparison conditions.
- `unset`: Discard a field. In subsequent rules, `EXISTS` tests against the field name are false, and comparisons of the field against any value are false.

  In the special case that the condition refers to exactly one field name, the field name following `unset` is optional and `unset` discards the named field. These rules are equivalent:

  ```
  IF myfield == 2 THEN unset myfield.
  IF myfield == 2 THEN unset.
  ```

Field References in log_filter_dragnet Rules

`log_filter_dragnet` rules support references to core, optional, and user-defined fields in error events.

- Core Field References
- Optional Field References
- User-Defined Field References

Core Field References

The `log_filter_dragnet` grammar at Grammar for log_filter_dragnet Rule Language names the core fields that filter rules recognize. For general descriptions of these fields, see Section 5.4.2.3, “Error Event Fields”, with which you are assumed to be familiar. The following remarks provide additional information only as it pertains specifically to core field references as used within `log_filter_dragnet` rules.

- `prio`

  The event priority, to indicate an error, warning, or note/information event. In comparisons, each priority can be specified as a symbolic priority name or an integer literal. Priority symbols are recognized only in comparisons with the `prio` field. These comparisons are equivalent:

```
IF prio == INFORMATION THEN ...
IF prio == 3 THEN ...
```

  The following table shows the permitted priority levels.

Event Type	Priority Symbol	Numeric Priority
Error event	`ERROR`	1
Warning event	`WARNING`	2
Note/information event	`INFORMATION`	3

  There is also a message priority of `SYSTEM`, but system messages cannot be filtered and are always written to the error log.

  Priority values follow the principle that higher priorities have lower values, and vice versa. Priority values begin at 1 for the most severe events (errors) and increase for events with decreasing priority. For example, to discard events with priority lower than warnings, test for priority values higher than `WARNING`:

```
IF prio > WARNING THEN drop.
```

  The following examples show the `log_filter_dragnet` rules to achieve an effect similar to each `log_error_verbosity` value permitted by the `log_filter_internal` filter:

  - Errors only (`log_error_verbosity=1`):

```
IF prio > ERROR THEN drop.
```

  - Errors and warnings (`log_error_verbosity=2`):

```
IF prio > WARNING THEN drop.
```

  - Errors, warnings, and notes (`log_error_verbosity=3`):

```
IF prio > INFORMATION THEN drop.
```

    This rule can actually be omitted because there are no `prio` values greater than `INFORMATION`, so effectively it drops nothing.

- `err_code`

  The numeric event error code. In comparisons, the value to test can be specified as a symbolic error name or an integer literal. Error symbols are recognized only in comparisons with the `err_code` field and user-defined fields. These comparisons are equivalent:

  ```
  IF err_code == ER_ACCESS_DENIED_ERROR THEN ...
  IF err_code == 1045 THEN ...
  ```

- `err_symbol`

  The event error symbol, as a string (for example, `'ER_DUP_KEY'`). `err_symbol` values are intended more for identifying particular lines in log output than for use in filter rule comparisons because `log_filter_dragnet` does not resolve comparison values specified as strings to the equivalent numeric error code. (For that to occur, an error must be specified using its unquoted symbol.)

Optional Field References

The `log_filter_dragnet` grammar at Grammar for log_filter_dragnet Rule Language names the optional fields that filter rules recognize. For general descriptions of these fields, see Section 5.4.2.3, “Error Event Fields”, with which you are assumed to be familiar. The following remarks provide additional information only as it pertains specifically to optional field references as used within `log_filter_dragnet` rules.

- `label`

  The label corresponding to the `prio` value, as a string. Filter rules can change the label for log sinks that support custom labels. `label` values are intended more for identifying particular lines in log output than for use in filter rule comparisons because `log_filter_dragnet` does not resolve comparison values specified as strings to the equivalent numeric priority.

- `source_file`

  The source file in which the event occurred, without any leading path. For example, to test for the `sql/gis/distance.cc` file, write the comparison like this:

  ```
  IF source_file == "distance.cc" THEN ...
  ```

User-Defined Field References

Any field name in a `log_filter_dragnet` filter rule not recognized as a core or optional field name is taken to refer to a user-defined field.

5.4.2.7 Error Logging in JSON Format

This section describes how to configure error logging using the built-in filter, `log_filter_internal`, and the JSON sink, `log_sink_json`, to take effect immediately and for subsequent server startups. For general information about configuring error logging, see Section 5.4.2.1, “Error Log Configuration”.

To enable the JSON sink, first load the sink component, then modify the `log_error_services` value:

```
INSTALL COMPONENT 'file://component_log_sink_json';
SET PERSIST log_error_services = 'log_filter_internal; log_sink_json';
```

To set `log_error_services` to take effect at server startup, use the instructions at Section 5.4.2.1, “Error Log Configuration”. Those instructions apply to other error-logging system variables as well.

It is permitted to name `log_sink_json` multiple times in the `log_error_services` value. For example, to write unfiltered events with one instance and filtered events with another instance, you could set `log_error_services` like this:

```
SET PERSIST log_error_services = 'log_sink_json; log_filter_internal; log_sink_json';
```

The JSON sink determines its output destination based on the default error log destination, which is given by the `log_error` system variable. If `log_error` names a file, the JSON sink bases output file naming on that file name, plus a numbered *`.NN`*`.json` suffix, with *`NN`* starting at 00. For example, if `log_error` is *`file_name`*, successive instances of `log_sink_json` named in the `log_error_services` value write to *`file_name`*`.00.json`, *`file_name`*`.01.json`, and so forth.

If `log_error` is `stderr`, the JSON sink writes to the console. If `log_sink_json` is named multiple times in the `log_error_services` value, they all write to the console, which is likely not useful.

5.4.2.8 Error Logging to the System Log

It is possible to have `mysqld` write the error log to the system log (the Event Log on Windows, and `syslog` on Unix and Unix-like systems).

This section describes how to configure error logging using the built-in filter, `log_filter_internal`, and the system log sink, `log_sink_syseventlog`, to take effect immediately and for subsequent server startups. For general information about configuring error logging, see Section 5.4.2.1, “Error Log Configuration”.

To enable the system log sink, first load the sink component, then modify the `log_error_services` value:

```
INSTALL COMPONENT 'file://component_log_sink_syseventlog';
SET PERSIST log_error_services = 'log_filter_internal; log_sink_syseventlog';
```

To set `log_error_services` to take effect at server startup, use the instructions at Section 5.4.2.1, “Error Log Configuration”. Those instructions apply to other error-logging system variables as well.

Note

For MySQL 8.0 configuration, you must enable error logging to the system log explicitly. This differs from MySQL 5.7 and earlier, for which error logging to the system log is enabled by default on Windows, and on all platforms requires no component loading.

Error logging to the system log may require additional system configuration. Consult the system log documentation for your platform.

On Windows, error messages written to the Event Log within the Application log have these characteristics:

- Entries marked as `Error`, `Warning`, and `Note` are written to the Event Log, but not messages such as information statements from individual storage engines.
- Event Log entries have a source of `MySQL` (or `MySQL-`*`tag`* if `syseventlog.tag` is defined as *`tag`*).

On Unix and Unix-like systems, logging to the system log uses `syslog`. The following system variables affect `syslog` messages:

- `syseventlog.facility`: The default facility for `syslog` messages is `daemon`. Set this variable to specify a different facility.
- `syseventlog.include_pid`: Whether to include the server process ID in each line of `syslog` output.
- `syseventlog.tag`: This variable defines a tag to add to the server identifier (`mysqld`) in `syslog` messages. If defined, the tag is appended to the identifier with a leading hyphen.

Note

Prior to MySQL 8.0.13, use the `log_syslog_facility`, `log_syslog_include_pid`, and `log_syslog_tag` system variables rather than the `syseventlog.`*`xxx`* variables.

MySQL uses the custom label “System” for important system messages about non-error situations, such as startup, shutdown, and some significant changes to settings. In logs that do not support custom labels, including the Event Log on Windows, and `syslog` on Unix and Unix-like systems, system messages are assigned the label used for the information priority level. However, these messages are printed to the log even if the MySQL `log_error_verbosity` setting normally excludes messages at the information level.

When a log sink must fall back to a label of “Information” instead of “System” in this way, and the log event is further processed outside of the MySQL server (for example, filtered or forwarded by a `syslog` configuration), these events may by default be processed by the secondary application as being of “Information” priority rather than “System” priority.

5.4.2.9 Error Log Output Format

Each error log sink (writer) component has a characteristic output format it uses to write messages to its destination, but other factors may influence the content of the messages:

- The information available to the log sink. If a log filter component executed prior to execution of the sink component removes a log event field, that field is not available for writing. For information about log filtering, see Section 5.4.2.4, “Types of Error Log Filtering”.
- The information relevant to the log sink. Not every sink writes all fields available in error events.
- System variables may affect log sinks. See System Variables That Affect Error Log Format.

For names and descriptions of the fields in error events, see Section 5.4.2.3, “Error Event Fields”. For all log sinks, the thread ID included in error log messages is that of the thread within `mysqld` responsible for writing the message. This ID indicates which part of the server produced the message, and is consistent with general query log and slow query log messages, which include the connection thread ID.

log_sink_internal Output Format

The internal log sink produces traditional error log output. For example:

```
2020-08-06T14:25:02.835618Z 0 [Note] [MY-012487] [InnoDB] DDL log recovery : begin
2020-08-06T14:25:02.936146Z 0 [Warning] [MY-010068] [Server] CA certificate /var/mysql/sslinfo/cacert.p
2020-08-06T14:25:02.963127Z 0 [Note] [MY-010253] [Server] IPv6 is available.
2020-08-06T14:25:03.109022Z 5 [Note] [MY-010051] [Server] Event Scheduler: scheduler thread started wit
```

Traditional-format messages have these fields:

```
time thread [label] [err_code] [subsystem] msg
```

The `[` and `]` square bracket characters are literal characters in the message format. They do not indicate that fields are optional.

The `label` value corresponds to the string form of the `prio` error event priority field.

The `[err_code]` and `[subsystem]` fields were added in MySQL 8.0. They are missing from logs generated by older servers. Log parsers can treat these fields as parts of the message text that is present only for logs written by servers recent enough to include them. Parsers must treat the `err_code` part of `[err_code]` indicators as a string value, not a number, because values such as `MY-012487` and `MY-010051` contain nonnumeric characters.

log_sink_json Output Format

The JSON-format log sink produces messages as JSON objects that contain key-value pairs. For example:

```
{
  "prio": 3,
  "err_code": 10051,
  "source_line": 561,
  "source_file": "event_scheduler.cc",
  "function": "run",
  "msg": "Event Scheduler: scheduler thread started with id 5",
  "time": "2020-08-06T14:25:03.109022Z",
  "ts": 1596724012005,
  "thread": 5,
  "err_symbol": "ER_SCHEDULER_STARTED",
  "SQL_state": "HY000",
  "subsystem": "Server",
  "buffered": 1596723903109022,
  "label": "Note"
}
```

The message shown is reformatted for readability. Events written to the error log appear one message per line.

The `ts` (timestamp) key was added in MySQL 8.0.20 and is unique to the JSON-format log sink. The value is an integer indicating milliseconds since the epoch (`'1970-01-01 00:00:00'` UTC).

The `ts` and `buffered` values are Unix timestamp values and can be converted using `FROM_UNIXTIME()` and an appropriate divisor:

```
mysql> SET time_zone = '+00:00';
mysql> SELECT FROM_UNIXTIME(1596724012005/1000.0);
+------------------------------------+
| FROM_UNIXTIME(1596724012005/1000.0) |
+------------------------------------+
| 2020-08-06 14:26:52.0050           |
+------------------------------------+
mysql> SELECT FROM_UNIXTIME(1596723903109022/1000000.0);
+-----------------------------------------+
| FROM_UNIXTIME(1596723903109022/1000000.0) |
+-----------------------------------------+
| 2020-08-06 14:25:03.1090                  |
+-----------------------------------------+
```

log_sink_syseventlog Output Format

The system log sink produces output that conforms to the system log format used on the local platform.

Early-Startup Logging Output Format

The server generates some error log messages before startup options have been processed, and thus before it knows error log settings such as the `log_error_verbosity` and `log_timestamps` system variable values, and before it knows which log components are to be used. The server handles error log messages that are generated early in the startup process as follows:

- Prior to MySQL 8.0.14, the server generates messages with the default timestamp, format, and verbosity level, and buffers them. After the startup options are processed and the error log configuration is known, the server flushes the buffered messages. Because these early messages use the default log configuration, they may differ from what is specified by the startup options. Also, the early messages are not flushed to log sinks other than the default. For example, logging to the JSON sink does not include these early messages because they are not in JSON format.
- As of MySQL 8.0.14, the server buffers log events rather than formatted log messages. This enables it to retroactively apply configuration settings to those events after the settings are known, with the result that flushed messages use the configured settings, not the defaults. Also, messages are flushed to all configured sinks, not just the default sink.

If a fatal error occurs before log configuration is known and the server must exit, the server formats buffered messages using the logging defaults so they are not lost. If no fatal error occurs but startup is excessively slow prior to processing startup options, the server periodically formats and flushes buffered messages using the logging defaults so as not to appear unresponsive. Although this behavior is similar to pre-8.0.14 behavior in that the defaults are used, it is preferable to losing messages when exceptional conditions occur.

System Variables That Affect Error Log Format

The `log_timestamps` system variable controls the time zone of timestamps in messages written to the error log (as well as to general query log and slow query log files). The server applies `log_timestamps` to error events before they reach any log sink; it thus affects error message output from all sinks.

Permitted `log_timestamps` values are `UTC` (the default) and `SYSTEM` (the local system time zone). Timestamps are written using ISO 8601 / RFC 3339 format: *`YYYY-MM-DDThh:mm:ss.uuuuuu`* plus a tail value of `Z` signifying Zulu time (UTC) or `±hh:mm` (an offset that indicates the local system time zone adjustment relative to UTC). For example:

```
2020-08-07T15:02:00.832521Z           (UTC)
2020-08-07T10:02:00.832521-05:00      (SYSTEM)
```

5.4.2.10 Error Log File Flushing and Renaming

If you flush the error log using a `FLUSH ERROR LOGS` or `FLUSH LOGS` statement, or a `mysqladmin flush-logs` command, the server closes and reopens any error log file to which it is writing. To rename an error log file, do so manually before flushing. Flushing the logs then opens a new file with the original file name. For example, assuming a log file name of *`host_name`*`.err`, use the following commands to rename the file and create a new one:

```
mv host_name.err host_name.err-old
mysqladmin flush-logs error
mv host_name.err-old backup-directory
```

On Windows, use `rename` rather than `mv`.

If the location of the error log file is not writable by the server, the log-flushing operation fails to create a new log file. For example, on Linux, the server might write the error log to the `/var/log/mysqld.log` file, where the `/var/log` directory is owned by `root` and is not writable by `mysqld`. For information about handling this case, see Section 5.4.6, “Server Log Maintenance”.

If the server is not writing to a named error log file, no error log file renaming occurs when the error log is flushed.

5.4.3 The General Query Log

The general query log is a general record of what `mysqld` is doing. The server writes information to this log when clients connect or disconnect, and it logs each SQL statement received from clients. The general query log can be very useful when you suspect an error in a client and want to know exactly what the client sent to `mysqld`.

Each line that shows when a client connects also includes `using` *`connection_type`* to indicate the protocol used to establish the connection. *`connection_type`* is one of `TCP/IP` (TCP/IP connection established without SSL), `SSL/TLS` (TCP/IP connection established with SSL), `Socket` (Unix socket file connection), `Named Pipe` (Windows named pipe connection), or `Shared Memory` (Windows shared memory connection).

`mysqld` writes statements to the query log in the order that it receives them, which might differ from the order in which they are executed. This logging order is in contrast with that of the binary log, for which statements are written after they are executed but before any locks are released. In addition, the query

log may contain statements that only select data while such statements are never written to the binary log.

When using statement-based binary logging on a replication source server, statements received by its replicas are written to the query log of each replica. Statements are written to the query log of the source if a client reads events with the `mysqlbinlog` utility and passes them to the server.

However, when using row-based binary logging, updates are sent as row changes rather than SQL statements, and thus these statements are never written to the query log when `binlog_format` is `ROW`. A given update also might not be written to the query log when this variable is set to `MIXED`, depending on the statement used. See Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”, for more information.

By default, the general query log is disabled. To specify the initial general query log state explicitly, use `--general_log[={0|1}]`. With no argument or an argument of 1, `--general_log` enables the log. With an argument of 0, this option disables the log. To specify a log file name, use `--general_log_file=`*`file_name`*. To specify the log destination, use the `log_output` system variable (as described in Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”).

Note

If you specify the `TABLE` log destination, see Log Tables and “Too many open files” Errors.

If you specify no name for the general query log file, the default name is *`host_name`*`.log`. The server creates the file in the data directory unless an absolute path name is given to specify a different directory.

To disable or enable the general query log or change the log file name at runtime, use the global `general_log` and `general_log_file` system variables. Set `general_log` to 0 (or `OFF`) to disable the log or to 1 (or `ON`) to enable it. Set `general_log_file` to specify the name of the log file. If a log file already is open, it is closed and the new file is opened.

When the general query log is enabled, the server writes output to any destinations specified by the `log_output` system variable. If you enable the log, the server opens the log file and writes startup messages to it. However, further logging of queries to the file does not occur unless the `FILE` log destination is selected. If the destination is `NONE`, the server writes no queries even if the general log is enabled. Setting the log file name has no effect on logging if the log destination value does not contain `FILE`.

Server restarts and log flushing do not cause a new general query log file to be generated (although flushing closes and reopens it). To rename the file and create a new one, use the following commands:

```
$> mv host_name.log host_name-old.log
$> mysqladmin flush-logs general
$> mv host_name-old.log backup-directory
```

On Windows, use `rename` rather than `mv`.

You can also rename the general query log file at runtime by disabling the log:

```
SET GLOBAL general_log = 'OFF';
```

With the log disabled, rename the log file externally (for example, from the command line). Then enable the log again:

```
SET GLOBAL general_log = 'ON';
```

This method works on any platform and does not require a server restart.

To disable or enable general query logging for the current session, set the session `sql_log_off` variable to `ON` or `OFF`. (This assumes that the general query log itself is enabled.)

Passwords in statements written to the general query log are rewritten by the server not to occur literally in plain text. Password rewriting can be suppressed for the general query log by starting the server with the `--log-raw` option. This option may be useful for diagnostic purposes, to see the exact text of statements as received by the server, but for security reasons is not recommended for production use. See also Section 6.1.2.3, “Passwords and Logging”.

An implication of password rewriting is that statements that cannot be parsed (due, for example, to syntax errors) are not written to the general query log because they cannot be known to be password free. Use cases that require logging of all statements including those with errors should use the `--log-raw` option, bearing in mind that this also bypasses password rewriting.

Password rewriting occurs only when plain text passwords are expected. For statements with syntax that expect a password hash value, no rewriting occurs. If a plain text password is supplied erroneously for such syntax, the password is logged as given, without rewriting.

The `log_timestamps` system variable controls the time zone of timestamps in messages written to the general query log file (as well as to the slow query log file and the error log). It does not affect the time zone of general query log and slow query log messages written to log tables, but rows retrieved from those tables can be converted from the local system time zone to any desired time zone with `CONVERT_TZ()` or by setting the session `time_zone` system variable.

5.4.4 The Binary Log

The binary log contains “events” that describe database changes such as table creation operations or changes to table data. It also contains events for statements that potentially could have made changes (for example, a `DELETE` which matched no rows), unless row-based logging is used. The binary log also contains information about how long each statement took that updated data. The binary log has two important purposes:

- For replication, the binary log on a replication source server provides a record of the data changes to be sent to replicas. The source sends the information contained in its binary log to its replicas, which reproduce those transactions to make the same data changes that were made on the source. See Section 17.2, “Replication Implementation”.
- Certain data recovery operations require use of the binary log. After a backup has been restored, the events in the binary log that were recorded after the backup was made are re-executed. These events bring databases up to date from the point of the backup. See Section 7.5, “Point-in-Time (Incremental) Recovery”.

The binary log is not used for statements such as `SELECT` or `SHOW` that do not modify data. To log all statements (for example, to identify a problem query), use the general query log. See Section 5.4.3, “The General Query Log”.

Running a server with binary logging enabled makes performance slightly slower. However, the benefits of the binary log in enabling you to set up replication and for restore operations generally outweigh this minor performance decrement.

The binary log is resilient to unexpected halts. Only complete events or transactions are logged or read back.

Passwords in statements written to the binary log are rewritten by the server not to occur literally in plain text. See also Section 6.1.2.3, “Passwords and Logging”.

From MySQL 8.0.14, binary log files and relay log files can be encrypted, helping to protect these files and the potentially sensitive data contained in them from being misused by outside attackers, and also from unauthorized viewing by users of the operating system where they are stored. You enable encryption on a MySQL server by setting the `binlog_encryption` system variable to `ON`. For more information, see Section 17.3.2, “Encrypting Binary Log Files and Relay Log Files”.

The following discussion describes some of the server options and variables that affect the operation of binary logging. For a complete list, see Section 17.1.6.4, “Binary Logging Options and Variables”.

Binary logging is enabled by default (the `log_bin` system variable is set to ON). The exception is if you use `mysqld` to initialize the data directory manually by invoking it with the `--initialize` or `--initialize-insecure` option, when binary logging is disabled by default, but can be enabled by specifying the `--log-bin` option.

To disable binary logging, you can specify the `--skip-log-bin` or `--disable-log-bin` option at startup. If either of these options is specified and `--log-bin` is also specified, the option specified later takes precedence.

The `--log-slave-updates` and `--slave-preserve-commit-order` options require binary logging. If you disable binary logging, either omit these options, or specify `--log-slave-updates=OFF` and `--skip-slave-preserve-commit-order`. MySQL disables these options by default when `--skip-log-bin` or `--disable-log-bin` is specified. If you specify `--log-slave-updates` or `--slave-preserve-commit-order` together with `--skip-log-bin` or `--disable-log-bin`, a warning or error message is issued.

The `--log-bin[=base_name]` option is used to specify the base name for binary log files. If you do not supply the `--log-bin` option, MySQL uses `binlog` as the default base name for the binary log files. For compatibility with earlier releases, if you supply the `--log-bin` option with no string or with an empty string, the base name defaults to *`host_name`*`-bin`, using the name of the host machine. It is recommended that you specify a base name, so that if the host name changes, you can easily continue to use the same binary log file names (see Section B.3.7, “Known Issues in MySQL”). If you supply an extension in the log name (for example, `--log-bin=`*`base_name.extension`*), the extension is silently removed and ignored.

`mysqld` appends a numeric extension to the binary log base name to generate binary log file names. The number increases each time the server creates a new log file, thus creating an ordered series of files. The server creates a new file in the series each time any of the following events occurs:

- The server is started or restarted
- The server flushes the logs.
- The size of the current log file reaches `max_binlog_size`.

A binary log file may become larger than `max_binlog_size` if you are using large transactions because a transaction is written to the file in one piece, never split between files.

To keep track of which binary log files have been used, `mysqld` also creates a binary log index file that contains the names of the binary log files. By default, this has the same base name as the binary log file, with the extension `'.index'`. You can change the name of the binary log index file with the `--log-bin-index[=`*`file_name`*`]` option. You should not manually edit this file while `mysqld` is running; doing so would confuse `mysqld`.

The term “binary log file” generally denotes an individual numbered file containing database events. The term “binary log” collectively denotes the set of numbered binary log files plus the index file.

The default location for binary log files and the binary log index file is the data directory. You can use the `--log-bin` option to specify an alternative location, by adding a leading absolute path name to the base name to specify a different directory. When the server reads an entry from the binary log index file, which tracks the binary log files that have been used, it checks whether the entry contains a relative path. If it does, the relative part of the path is replaced with the absolute path set using the `--log-bin` option. An absolute path recorded in the binary log index file remains unchanged; in such a case, the index file must be edited manually to enable a new path or paths to be used. The binary log file base name and any specified path are available as the `log_bin_basename` system variable.

In MySQL 5.7, a server ID had to be specified when binary logging was enabled, or the server would not start. In MySQL 8.0, the `server_id` system variable is set to 1 by default. The server can be started with this default ID when binary logging is enabled, but an informational message is issued if you do not specify a server ID explicitly using the `server_id` system variable. For servers that are used in a replication topology, you must specify a unique nonzero server ID for each server.

A client that has privileges sufficient to set restricted session system variables (see Section 5.1.9.1, “System Variable Privileges”) can disable binary logging of its own statements by using a `SET sql_log_bin=OFF` statement.

By default, the server logs the length of the event as well as the event itself and uses this to verify that the event was written correctly. You can also cause the server to write checksums for the events by setting the `binlog_checksum` system variable. When reading back from the binary log, the source uses the event length by default, but can be made to use checksums if available by enabling the system variable `source_verify_checksum` (from MySQL 8.0.26) or `master_verify_checksum` (before MySQL 8.0.26). The replication I/O (receiver) thread on the replica also verifies events received from the source. You can cause the replication SQL (applier) thread to use checksums if available when reading from the relay log by enabling the system variable `replica_sql_verify_checksum` (from MySQL 8.0.26) or `slave_sql_verify_checksum` (before MySQL 8.0.26).

The format of the events recorded in the binary log is dependent on the binary logging format. Three format types are supported: row-based logging, statement-based logging and mixed-base logging. The binary logging format used depends on the MySQL version. For general descriptions of the logging formats, see Section 5.4.4.1, “Binary Logging Formats”. For detailed information about the format of the binary log, see MySQL Internals: The Binary Log.

The server evaluates the `--binlog-do-db` and `--binlog-ignore-db` options in the same way as it does the `--replicate-do-db` and `--replicate-ignore-db` options. For information about how this is done, see Section 17.2.5.1, “Evaluation of Database-Level Replication and Binary Logging Options”.

A replica is started with the system variable `log_replica_updates` (from MySQL 8.0.26) or `log_slave_updates` (before MySQL 8.0.26) enabled by default, meaning that the replica writes to its own binary log any data modifications that are received from the source. The binary log must be enabled for this setting to work (see Section 17.1.6.3, “Replica Server Options and Variables”). This setting enables the replica to act as a source to other replicas.

You can delete all binary log files with the `RESET MASTER` statement, or a subset of them with `PURGE BINARY LOGS`. See Section 13.7.8.6, “RESET Statement”, and Section 13.4.1.1, “PURGE BINARY LOGS Statement”.

If you are using replication, you should not delete old binary log files on the source until you are sure that no replica still needs to use them. For example, if your replicas never run more than three days behind, once a day you can execute `mysqladmin flush-logs binary` on the source and then remove any logs that are more than three days old. You can remove the files manually, but it is preferable to use `PURGE BINARY LOGS`, which also safely updates the binary log index file for you (and which can take a date argument). See Section 13.4.1.1, “PURGE BINARY LOGS Statement”.

You can display the contents of binary log files with the `mysqlbinlog` utility. This can be useful when you want to reprocess statements in the log for a recovery operation. For example, you can update a MySQL server from the binary log as follows:

```
$> mysqlbinlog log_file | mysql -h server_name
```

`mysqlbinlog` also can be used to display the contents of the relay log file on a replica, because they are written using the same format as binary log files. For more information on the `mysqlbinlog` utility and how to use it, see Section 4.6.9, “mysqlbinlog — Utility for Processing Binary Log Files”. For more information about the binary log and recovery operations, see Section 7.5, “Point-in-Time (Incremental) Recovery”.

Binary logging is done immediately after a statement or transaction completes but before any locks are released or any commit is done. This ensures that the log is logged in commit order.

Updates to nontransactional tables are stored in the binary log immediately after execution.

Within an uncommitted transaction, all updates (`UPDATE`, `DELETE`, or `INSERT`) that change transactional tables such as `InnoDB` tables are cached until a `COMMIT` statement is received by the

server. At that point, `mysqld` writes the entire transaction to the binary log before the `COMMIT` is executed.

Modifications to nontransactional tables cannot be rolled back. If a transaction that is rolled back includes modifications to nontransactional tables, the entire transaction is logged with a `ROLLBACK` statement at the end to ensure that the modifications to those tables are replicated.

When a thread that handles the transaction starts, it allocates a buffer of `binlog_cache_size` to buffer statements. If a statement is bigger than this, the thread opens a temporary file to store the transaction. The temporary file is deleted when the thread ends. From MySQL 8.0.17, if binary log encryption is active on the server, the temporary file is encrypted.

The `Binlog_cache_use` status variable shows the number of transactions that used this buffer (and possibly a temporary file) for storing statements. The `Binlog_cache_disk_use` status variable shows how many of those transactions actually had to use a temporary file. These two variables can be used for tuning `binlog_cache_size` to a large enough value that avoids the use of temporary files.

The `max_binlog_cache_size` system variable (default 4GB, which is also the maximum) can be used to restrict the total size used to cache a multiple-statement transaction. If a transaction is larger than this many bytes, it fails and rolls back. The minimum value is 4096.

If you are using the binary log and row based logging, concurrent inserts are converted to normal inserts for `CREATE ... SELECT` or `INSERT ... SELECT` statements. This is done to ensure that you can re-create an exact copy of your tables by applying the log during a backup operation. If you are using statement-based logging, the original statement is written to the log.

The binary log format has some known limitations that can affect recovery from backups. See Section 17.5.1, “Replication Features and Issues”.

Binary logging for stored programs is done as described in Section 25.7, “Stored Program Binary Logging”.

Note that the binary log format differs in MySQL 8.0 from previous versions of MySQL, due to enhancements in replication. See Section 17.5.2, “Replication Compatibility Between MySQL Versions”.

If the server is unable to write to the binary log, flush binary log files, or synchronize the binary log to disk, the binary log on the replication source server can become inconsistent and replicas can lose synchronization with the source. The `binlog_error_action` system variable controls the action taken if an error of this type is encountered with the binary log.

- The default setting, `ABORT_SERVER`, makes the server halt binary logging and shut down. At this point, you can identify and correct the cause of the error. On restart, recovery proceeds as in the case of an unexpected server halt (see Section 17.4.2, “Handling an Unexpected Halt of a Replica”).
- The setting `IGNORE_ERROR` provides backward compatibility with older versions of MySQL. With this setting, the server continues the ongoing transaction and logs the error, then halts binary logging, but continues to perform updates. At this point, you can identify and correct the cause of the error. To resume binary logging, `log_bin` must be enabled again, which requires a server restart. Only use this option if you require backward compatibility, and the binary log is non-essential on this MySQL server instance. For example, you might use the binary log only for intermittent auditing or debugging of the server, and not use it for replication from the server or rely on it for point-in-time restore operations.

By default, the binary log is synchronized to disk at each write (`sync_binlog=1`). If `sync_binlog` was not enabled, and the operating system or machine (not only the MySQL server) crashed, there is a chance that the last statements of the binary log could be lost. To prevent this, enable the `sync_binlog` system variable to synchronize the binary log to disk after every *`N`* commit groups. See Section 5.1.8, “Server System Variables”. The safest value for `sync_binlog` is 1 (the default), but this is also the slowest.

In earlier MySQL releases, there was a chance of inconsistency between the table content and binary log content if a crash occurred, even with `sync_binlog` set to 1. For example, if you are using `InnoDB` tables and the MySQL server processes a `COMMIT` statement, it writes many prepared transactions to the binary log in sequence, synchronizes the binary log, and then commits the transaction into `InnoDB`. If the server unexpectedly exited between those two operations, the transaction would be rolled back by `InnoDB` at restart but still exist in the binary log. Such an issue was resolved in previous releases by enabling `InnoDB` support for two-phase commit in XA transactions. In MySQL 8.0, `InnoDB` support for two-phase commit in XA transactions is always enabled.

`InnoDB` support for two-phase commit in XA transactions ensures that the binary log and `InnoDB` data files are synchronized. However, the MySQL server should also be configured to synchronize the binary log and the `InnoDB` logs to disk before committing the transaction. The `InnoDB` logs are synchronized by default, and `sync_binlog=1` ensures the binary log is synchronized. The effect of implicit `InnoDB` support for two-phase commit in XA transactions and `sync_binlog=1` is that at restart after a crash, after doing a rollback of transactions, the MySQL server scans the latest binary log file to collect transaction *`xid`* values and calculate the last valid position in the binary log file. The MySQL server then tells `InnoDB` to complete any prepared transactions that were successfully written to the to the binary log, and truncates the binary log to the last valid position. This ensures that the binary log reflects the exact data of `InnoDB` tables, and therefore the replica remains in synchrony with the source because it does not receive a statement which has been rolled back.

If the MySQL server discovers at crash recovery that the binary log is shorter than it should have been, it lacks at least one successfully committed `InnoDB` transaction. This should not happen if `sync_binlog=1` and the disk/file system do an actual sync when they are requested to (some do not), so the server prints an error message `The binary log file_name is shorter than its expected size`. In this case, this binary log is not correct and replication should be restarted from a fresh snapshot of the source's data.

The session values of the following system variables are written to the binary log and honored by the replica when parsing the binary log:

- `sql_mode` (except that the `NO_DIR_IN_CREATE` mode is not replicated; see Section 17.5.1.39, “Replication and Variables”)
- `foreign_key_checks`
- `unique_checks`
- `character_set_client`
- `collation_connection`
- `collation_database`
- `collation_server`
- `sql_auto_is_null`

5.4.4.1 Binary Logging Formats

The server uses several logging formats to record information in the binary log:

- Replication capabilities in MySQL originally were based on propagation of SQL statements from source to replica. This is called *statement-based logging*. You can cause this format to be used by starting the server with `--binlog-format=STATEMENT`.
- In *row-based logging* (the default), the source writes events to the binary log that indicate how individual table rows are affected. You can cause the server to use row-based logging by starting it with `--binlog-format=ROW`.
- A third option is also available: *mixed logging*. With mixed logging, statement-based logging is used by default, but the logging mode switches automatically to row-based in certain cases as described

below. You can cause MySQL to use mixed logging explicitly by starting `mysqld` with the option `--binlog-format=MIXED`.

The logging format can also be set or limited by the storage engine being used. This helps to eliminate issues when replicating certain statements between a source and replica which are using different storage engines.

With statement-based replication, there may be issues with replicating nondeterministic statements. In deciding whether or not a given statement is safe for statement-based replication, MySQL determines whether it can guarantee that the statement can be replicated using statement-based logging. If MySQL cannot make this guarantee, it marks the statement as potentially unreliable and issues the warning, `Statement may not be safe to log in statement format`.

You can avoid these issues by using MySQL's row-based replication instead.

5.4.4.2 Setting The Binary Log Format

You can select the binary logging format explicitly by starting the MySQL server with `--binlog-format=`*`type`*. The supported values for *`type`* are:

- `STATEMENT` causes logging to be statement based.
- `ROW` causes logging to be row based. This is the default.
- `MIXED` causes logging to use mixed format.

Setting the binary logging format does not activate binary logging for the server. The setting only takes effect when binary logging is enabled on the server, which is the case when the `log_bin` system variable is set to `ON`. From MySQL 8.0, binary logging is enabled by default, and is only disabled if you specify the `--skip-log-bin` or `--disable-log-bin` option at startup.

The logging format also can be switched at runtime, although note that there are a number of situations in which you cannot do this, as discussed later in this section. Set the global value of the `binlog_format` system variable to specify the format for clients that connect subsequent to the change:

```
mysql> SET GLOBAL binlog_format = 'STATEMENT';
mysql> SET GLOBAL binlog_format = 'ROW';
mysql> SET GLOBAL binlog_format = 'MIXED';
```

An individual client can control the logging format for its own statements by setting the session value of `binlog_format`:

```
mysql> SET SESSION binlog_format = 'STATEMENT';
mysql> SET SESSION binlog_format = 'ROW';
mysql> SET SESSION binlog_format = 'MIXED';
```

Changing the global `binlog_format` value requires privileges sufficient to set global system variables. Changing the session `binlog_format` value requires privileges sufficient to set restricted session system variables. See Section 5.1.9.1, “System Variable Privileges”.

There are several reasons why a client might want to set binary logging on a per-session basis:

- A session that makes many small changes to the database might want to use row-based logging.
- A session that performs updates that match many rows in the `WHERE` clause might want to use statement-based logging because it is more efficient to log a few statements than many rows.
- Some statements require a lot of execution time on the source, but result in just a few rows being modified. It might therefore be beneficial to replicate them using row-based logging.

There are exceptions when you cannot switch the replication format at runtime:

- The replication format cannot be changed from within a stored function or a trigger.

- If the NDB storage engine is enabled.
- If a session has open temporary tables, the replication format cannot be changed for the session (SET @@SESSION.binlog_format).
- If any replication channel has open temporary tables, the replication format cannot be changed globally (SET @@GLOBAL.binlog_format or SET @@PERSIST.binlog_format).
- If any replication channel applier thread is currently running, the replication format cannot be changed globally (SET @@GLOBAL.binlog_format or SET @@PERSIST.binlog_format).

Trying to switch the replication format in any of these cases (or attempting to set the current replication format) results in an error. You can, however, use PERSIST_ONLY (SET @@PERSIST_ONLY.binlog_format) to change the replication format at any time, because this action does not modify the runtime global system variable value, and takes effect only after a server restart.

Switching the replication format at runtime is not recommended when any temporary tables exist, because temporary tables are logged only when using statement-based replication, whereas with row-based replication and mixed replication, they are not logged.

Switching the replication format while replication is ongoing can also cause issues. Each MySQL Server can set its own and only its own binary logging format (true whether binlog_format is set with global or session scope). This means that changing the logging format on a replication source server does not cause a replica to change its logging format to match. When using STATEMENT mode, the binlog_format system variable is not replicated. When using MIXED or ROW logging mode, it is replicated but is ignored by the replica.

A replica is not able to convert binary log entries received in ROW logging format to STATEMENT format for use in its own binary log. The replica must therefore use ROW or MIXED format if the source does. Changing the binary logging format on the source from STATEMENT to ROW or MIXED while replication is ongoing to a replica with STATEMENT format can cause replication to fail with errors such as Error executing row event: 'Cannot execute statement: impossible to write to binary log since statement is in row format and BINLOG_FORMAT = STATEMENT.' Changing the binary logging format on the replica to STATEMENT format when the source is still using MIXED or ROW format also causes the same type of replication failure. To change the format safely, you must stop replication and ensure that the same change is made on both the source and the replica.

If you are using InnoDB tables and the transaction isolation level is READ COMMITTED or READ UNCOMMITTED, only row-based logging can be used. It is *possible* to change the logging format to STATEMENT, but doing so at runtime leads very rapidly to errors because InnoDB can no longer perform inserts.

With the binary log format set to ROW, many changes are written to the binary log using the row-based format. Some changes, however, still use the statement-based format. Examples include all DDL (data definition language) statements such as CREATE TABLE, ALTER TABLE, or DROP TABLE.

When row-based binary logging is used, the binlog_row_event_max_size system variable and its corresponding startup option --binlog-row-event-max-size set a soft limit on the maximum size of row events. The default value is 8192 bytes, and the value can only be changed at server startup. Where possible, rows stored in the binary log are grouped into events with a size not exceeding the value of this setting. If an event cannot be split, the maximum size can be exceeded.

The --binlog-row-event-max-size option is available for servers that are capable of row-based replication. Rows are stored into the binary log in chunks having a size in bytes not exceeding the value of this option. The value must be a multiple of 256. The default value is 8192.

Warning

When using *statement-based logging* for replication, it is possible for the data on the source and replica to become different if a statement is designed in

> such a way that the data modification is *nondeterministic*; that is, it is left up to the query optimizer. In general, this is not a good practice even outside of replication. For a detailed explanation of this issue, see Section B.3.7, “Known Issues in MySQL”.

5.4.4.3 Mixed Binary Logging Format

When running in `MIXED` logging format, the server automatically switches from statement-based to row-based logging under the following conditions:

- When a function contains `UUID()`.
- When one or more tables with `AUTO_INCREMENT` columns are updated and a trigger or stored function is invoked. Like all other unsafe statements, this generates a warning if `binlog_format = STATEMENT`.

  For more information, see Section 17.5.1.1, “Replication and AUTO_INCREMENT”.
- When the body of a view requires row-based replication, the statement creating the view also uses it. For example, this occurs when the statement creating a view uses the `UUID()` function.
- When a call to a loadable function is involved.
- When `FOUND_ROWS()` or `ROW_COUNT()` is used. (Bug #12092, Bug #30244)
- When `USER()`, `CURRENT_USER()`, or `CURRENT_USER` is used. (Bug #28086)
- When one of the tables involved is a log table in the `mysql` database.
- When the `LOAD_FILE()` function is used. (Bug #39701)
- When a statement refers to one or more system variables. (Bug #31168)

  Exception. The following system variables, when used with session scope (only), do not cause the logging format to switch:

  - `auto_increment_increment`
  - `auto_increment_offset`
  - `character_set_client`
  - `character_set_connection`
  - `character_set_database`
  - `character_set_server`
  - `collation_connection`
  - `collation_database`
  - `collation_server`
  - `foreign_key_checks`
  - `identity`
  - `last_insert_id`
  - `lc_time_names`
  - `pseudo_thread_id`

- `sql_auto_is_null`
- `time_zone`
- `timestamp`
- `unique_checks`

For information about determining system variable scope, see Section 5.1.9, “Using System Variables”.

For information about how replication treats `sql_mode`, see Section 17.5.1.39, “Replication and Variables”.

In earlier releases, when mixed binary logging format was in use, if a statement was logged by row and the session that executed the statement had any temporary tables, all subsequent statements were treated as unsafe and logged in row-based format until all temporary tables in use by that session were dropped. As of MySQL 8.0, operations on temporary tables are not logged in mixed binary logging format, and the presence of temporary tables in the session has no impact on the logging mode used for each statement.

Note

A warning is generated if you try to execute a statement using statement-based logging that should be written using row-based logging. The warning is shown both in the client (in the output of `SHOW WARNINGS`) and through the `mysqld` error log. A warning is added to the `SHOW WARNINGS` table each time such a statement is executed. However, only the first statement that generated the warning for each client session is written to the error log to prevent flooding the log.

In addition to the decisions above, individual engines can also determine the logging format used when information in a table is updated. The logging capabilities of an individual engine can be defined as follows:

- If an engine supports row-based logging, the engine is said to be *row-logging capable*.
- If an engine supports statement-based logging, the engine is said to be *statement-logging capable*.

A given storage engine can support either or both logging formats. The following table lists the formats supported by each engine.

Storage Engine	Row Logging Supported	Statement Logging Supported
`ARCHIVE`	Yes	Yes
`BLACKHOLE`	Yes	Yes
`CSV`	Yes	Yes
`EXAMPLE`	Yes	No
`FEDERATED`	Yes	Yes
`HEAP`	Yes	Yes
`InnoDB`	Yes	Yes when the transaction isolation level is `REPEATABLE READ` or `SERIALIZABLE`; No otherwise.
`MyISAM`	Yes	Yes
`MERGE`	Yes	Yes
`NDB`	Yes	No

Whether a statement is to be logged and the logging mode to be used is determined according to the type of statement (safe, unsafe, or binary injected), the binary logging format (`STATEMENT`, `ROW`, or `MIXED`), and the logging capabilities of the storage engine (statement capable, row capable, both, or neither). (Binary injection refers to logging a change that must be logged using `ROW` format.)

Statements may be logged with or without a warning; failed statements are not logged, but generate errors in the log. This is shown in the following decision table. **Type**, **binlog_format**, **SLC**, and **RLC** columns outline the conditions, and **Error / Warning** and **Logged as** columns represent the corresponding actions. **SLC** stands for “statement-logging capable”, and **RLC** stands for “row-logging capable”.

Type	binlog_format	SLC	RLC	Error / Warning	Logged as
*	*	No	No	Error: Cannot execute statement: Binary logging is impossible since at least one engine is involved that is both row-incapable and statement-incapable.	-
Safe	STATEMENT	Yes	No	-	STATEMENT
Safe	MIXED	Yes	No	-	STATEMENT
Safe	ROW	Yes	No	Error: Cannot execute statement: Binary logging is impossible since BINLOG_FORMAT = ROW and at least one table uses a storage engine that is not capable of row-based logging.	-
Unsafe	STATEMENT	Yes	No	Warning: Unsafe statement binlogged in statement format, since BINLOG_FORMAT = STATEMENT	STATEMENT
Unsafe	MIXED	Yes	No	Error: Cannot execute statement:	-

Type	binlog_format	SLC	RLC	Error / Warning	Logged as
				Binary logging of an unsafe statement is impossible when the storage engine is limited to statement-based logging, even if BINLOG_FORMAT = MIXED.	
Unsafe	ROW	Yes	No	Error: Cannot execute statement: Binary logging is impossible since BINLOG_FORMAT = ROW and at least one table uses a storage engine that is not capable of row-based logging.	-
Row Injection	STATEMENT	Yes	No	Error: Cannot execute row injection: Binary logging is not possible since at least one table uses a storage engine that is not capable of row-based logging.	-
Row Injection	MIXED	Yes	No	Error: Cannot execute row injection: Binary logging is not possible since at least one table uses a storage engine that is not capable of row-based logging.	-

Type	binlog_format	SLC	RLC	Error / Warning	Logged as
Row Injection	ROW	Yes	No	Error: Cannot execute row injection: Binary logging is not possible since at least one table uses a storage engine that is not capable of row-based logging.	-
Safe	STATEMENT	No	Yes	Error: Cannot execute statement: Binary logging is impossible since BINLOG_FORMAT = STATEMENT and at least one table uses a storage engine that is not capable of statement-based logging.	-
Safe	MIXED	No	Yes	-	ROW
Safe	ROW	No	Yes	-	ROW
Unsafe	STATEMENT	No	Yes	Error: Cannot execute statement: Binary logging is impossible since BINLOG_FORMAT = STATEMENT and at least one table uses a storage engine that is not capable of statement-based logging.	-
Unsafe	MIXED	No	Yes	-	ROW
Unsafe	ROW	No	Yes	-	ROW
Row Injection	STATEMENT	No	Yes	Error: Cannot execute row	-

Type	binlog_format	SLC	RLC	Error / Warning	Logged as
				injection: Binary logging is not possible since BINLOG_FORMAT = STATEMENT.	
Row Injection	MIXED	No	Yes	-	ROW
Row Injection	ROW	No	Yes	-	ROW
Safe	STATEMENT	Yes	Yes	-	STATEMENT
Safe	MIXED	Yes	Yes	-	STATEMENT
Safe	ROW	Yes	Yes	-	ROW
Unsafe	STATEMENT	Yes	Yes	Warning: Unsafe statement binlogged in statement format since BINLOG_FORMAT = STATEMENT.	STATEMENT
Unsafe	MIXED	Yes	Yes	-	ROW
Unsafe	ROW	Yes	Yes	-	ROW
Row Injection	STATEMENT	Yes	Yes	Error: Cannot execute row injection: Binary logging is not possible because BINLOG_FORMAT = STATEMENT.	-
Row Injection	MIXED	Yes	Yes	-	ROW
Row Injection	ROW	Yes	Yes	-	ROW

When a warning is produced by the determination, a standard MySQL warning is produced (and is available using SHOW WARNINGS). The information is also written to the mysqld error log. Only one error for each error instance per client connection is logged to prevent flooding the log. The log message includes the SQL statement that was attempted.

If a replica has log_error_verbosity set to display warnings, the replica prints messages to the error log to provide information about its status, such as the binary log and relay log coordinates where it starts its job, when it is switching to another relay log, when it reconnects after a disconnect, statements that are unsafe for statement-based logging, and so forth.

5.4.4.4 Logging Format for Changes to mysql Database Tables

The contents of the grant tables in the mysql database can be modified directly (for example, with INSERT or DELETE) or indirectly (for example, with GRANT or CREATE USER). Statements that affect mysql database tables are written to the binary log using the following rules:

- Data manipulation statements that change data in `mysql` database tables directly are logged according to the setting of the `binlog_format` system variable. This pertains to statements such as `INSERT`, `UPDATE`, `DELETE`, `REPLACE`, `DO`, `LOAD DATA`, `SELECT`, and `TRUNCATE TABLE`.
- Statements that change the `mysql` database indirectly are logged as statements regardless of the value of `binlog_format`. This pertains to statements such as `GRANT`, `REVOKE`, `SET PASSWORD`, `RENAME USER`, `CREATE` (all forms except `CREATE TABLE ... SELECT`), `ALTER` (all forms), and `DROP` (all forms).

`CREATE TABLE ... SELECT` is a combination of data definition and data manipulation. The `CREATE TABLE` part is logged using statement format and the `SELECT` part is logged according to the value of `binlog_format`.

5.4.4.5 Binary Log Transaction Compression

Beginning with MySQL 8.0.20, you can enable binary log transaction compression on a MySQL server instance. When binary log transaction compression is enabled, transaction payloads are compressed using the zstd algorithm, and then written to the server's binary log file as a single event (a `Transaction_payload_event`).

Compressed transaction payloads remain in a compressed state while they are sent in the replication stream to replicas, other Group Replication group members, or clients such as `mysqlbinlog`. They are not decompressed by receiver threads, and are written to the relay log still in their compressed state. Binary log transaction compression therefore saves storage space both on the originator of the transaction and on the recipient (and for their backups), and saves network bandwidth when the transactions are sent between server instances.

Compressed transaction payloads are decompressed when the individual events contained in them need to be inspected. For example, the `Transaction_payload_event` is decompressed by an applier thread in order to apply the events it contains on the recipient. Decompression is also carried out during recovery, by `mysqlbinlog` when replaying transactions, and by the `SHOW BINLOG EVENTS` and `SHOW RELAYLOG EVENTS` statements.

You can enable binary log transaction compression on a MySQL server instance using the `binlog_transaction_compression` system variable, which defaults to `OFF`. You can also use the `binlog_transaction_compression_level_zstd` system variable to set the level for the zstd algorithm that is used for compression. This value determines the compression effort, from 1 (the lowest effort) to 22 (the highest effort). As the compression level increases, the compression ratio increases, which reduces the storage space and network bandwidth required for the transaction payload. However, the effort required for data compression also increases, taking time and CPU and memory resources on the originating server. Increases in the compression effort do not have a linear relationship to increases in the compression ratio.

Note

Prior to NDB 8.0.31: Binary log transaction compression can be enabled in NDB Cluster, but only when starting the server using the --binlog-transaction-compression option (and possibly --binlog-transaction-compression-level-zstd as well); changing the value of either or both of the system variables `binlog_transaction_compression` and `binlog_transaction_compression_level_zstd` at run time has no effect on the logging of `NDB` tables.

NDB 8.0.31 and later: You can enable binary logging of compressed transactions for tables using the `NDB` storage engine at run time using the `ndb_log_transaction_compression` system variable introduced in that release, and control the level of compression using `ndb_log_transaction_compression_level_zstd`. Starting `mysqld` with `--binlog-transaction-compression` on the command line or in a `my.cnf` file causes `ndb_log_transaction_compression`

to be enabled automatically and any setting for the `--ndb-log-transaction-compression` option to be ignored; to disable binary log transaction compression for the `NDB` storage engine *only*, set `ndb_log_transaction_compression=OFF` in a client session after starting `mysqld`.

The following types of event are excluded from binary log transaction compression, so are always written uncompressed to the binary log:

- Events relating to the GTID for the transaction (including anonymous GTID events).
- Other types of control event, such as view change events and heartbeat events.
- Incident events and the whole of any transactions that contain them.
- Non-transactional events and the whole of any transactions that contain them. A transaction involving a mix of non-transactional and transactional storage engines does not have its payload compressed.
- Events that are logged using statement-based binary logging. Binary log transaction compression is only applied for the row-based binary logging format.

Binary log encryption can be used on binary log files that contain compressed transactions.

Behaviors When Binary Log Transaction Compression is Enabled

Transactions with payloads that are compressed can be rolled back like any other transaction, and they can also be filtered out on a replica by the usual filtering options. Binary log transaction compression can be applied to XA transactions.

When binary log transaction compression is enabled, the `max_allowed_packet` and `replica_max_allowed_packet` or `slave_max_allowed_packet` limits for the server still apply, and are measured on the compressed size of the `Transaction_payload_event`, plus the bytes used for the event header.

Important

Compressed transaction payloads are sent as a single packet, rather than each event of the transaction being sent in an individual packet, as is the case when binary log transaction compression is not in use. If your replication topology handles large transactions, be aware that a large transaction which can be replicated successfully when binary log transaction compression is not in use, might stop replication due to its size when binary log transaction compression is in use.

For multithreaded workers, each transaction (including its GTID event and `Transaction_payload_event`) is assigned to a worker thread. The worker thread decompresses the transaction payload and applies the individual events in it one by one. If an error is found applying any event within the `Transaction_payload_event`, the complete transaction is reported to the co-ordinator as having failed. When `replica_parallel_type` or `slave_parallel_type` is set to `DATABASE`, all the databases affected by the transaction are mapped before the transaction is scheduled. The use of binary log transaction compression with the `DATABASE` policy can reduce parallelism compared to uncompressed transactions, which are mapped and scheduled for each event.

For semisynchronous replication (see Section 17.4.10, “Semisynchronous Replication”), the replica acknowledges the transaction when the complete `Transaction_payload_event` has been received.

When binary log checksums are enabled (which is the default), the replication source server does not write checksums for individual events in a compressed transaction payload. Instead, a checksum is

written for the complete `Transaction_payload_event`, and individual checksums are written for any events that were not compressed, such as events relating to GTIDs.

For the `SHOW BINLOG EVENTS` and `SHOW RELAYLOG EVENTS` statements, the `Transaction_payload_event` is first printed as a single unit, then it is unpacked and each event inside it is printed.

For operations that reference the end position of an event, such as `START REPLICA` (or before MySQL 8.0.22, `START SLAVE`) with the `UNTIL` clause, `SOURCE_POS_WAIT()` or `MASTER_POS_WAIT()`, and `sql_replica_skip_counter` or `sql_slave_skip_counter`, you must specify the end position of the compressed transaction payload (the `Transaction_payload_event`). When skipping events using `sql_replica_skip_counter` or `sql_slave_skip_counter`, a compressed transaction payload is counted as a single counter value, so all the events inside it are skipped as a unit.

Combining Compressed and Uncompressed Transaction Payloads

MySQL Server releases that support binary log transaction compression can handle a mix of compressed and uncompressed transaction payloads.

- The system variables relating to binary log transaction compression do not need to be set the same on all Group Replication group members, and are not replicated from sources to replicas in a replication topology. You can decide whether or not binary log transaction compression is appropriate for each MySQL Server instance that has a binary log.
- If transaction compression is enabled then disabled on a server, compression is not applied to future transactions originated on that server, but transaction payloads that have been compressed can still be handled and displayed.
- If transaction compression is specified for individual sessions by setting the session value of `binlog_transaction_compression`, the binary log can contain a mix of compressed and uncompressed transaction payloads.

When a source in a replication topology and its replica both have binary log transaction compression enabled, the replica receives compressed transaction payloads and writes them compressed to its relay log. It decompresses the transaction payloads to apply the transactions, and then compresses them again after applying for writing to its binary log. Any downstream replicas receive the compressed transaction payloads.

When a source in a replication topology has binary log transaction compression enabled but its replica does not, the replica receives compressed transaction payloads and writes them compressed to its relay log. It decompresses the transaction payloads to apply the transactions, and then writes them uncompressed to its own binary log, if it has one. Any downstream replicas receive the uncompressed transaction payloads.

When a source in a replication topology does not have binary log transaction compression enabled but its replica does, if the replica has a binary log, it compresses the transaction payloads after applying them, and writes the compressed transaction payloads to its binary log. Any downstream replicas receive the compressed transaction payloads.

When a MySQL server instance has no binary log, if it is at a release from MySQL 8.0.20, it can receive, handle, and display compressed transaction payloads regardless of its value for `binlog_transaction_compression`. Compressed transaction payloads received by such server instances are written in their compressed state to the relay log, so they benefit indirectly from compression that was carried out by other servers in the replication topology.

A replica at a release before MySQL 8.0.20 cannot replicate from a source with binary log transaction compression enabled. A replica at or above MySQL 8.0.20 can replicate from a source at an earlier release that does not support binary log transaction compression, and can carry out its own compression on transactions received from that source when writing them to its own binary log.

Monitoring Binary Log Transaction Compression

You can monitor the effects of binary log transaction compression using the Performance Schema table `binary_log_transaction_compression_stats`. The statistics include the data compression ratio for the monitored period, and you can also view the effect of compression on the last transaction on the server. You can reset the statistics by truncating the table. Statistics for binary logs and relay logs are split out so you can see the impact of compression for each log type. The MySQL server instance must have a binary log to produce these statistics.

The Performance Schema table `events_stages_current` shows when a transaction is in the stage of decompression or compression for its transaction payload, and displays its progress for this stage. Compression is carried out by the worker thread handling the transaction, just before the transaction is committed, provided that there are no events in the finalized capture cache that exclude the transaction from binary log transaction compression (for example, incident events). When decompression is required, it is carried out for one event from the payload at a time.

`mysqlbinlog` with the `--verbose` option includes comments stating the compressed size and the uncompressed size for compressed transaction payloads, and the compression algorithm that was used.

You can enable connection compression at the protocol level for replication connections, using the `SOURCE_COMPRESSION_ALGORITHMS` | `MASTER_COMPRESSION_ALGORITHMS` and `SOURCE_ZSTD_COMPRESSION_LEVEL` | `MASTER_ZSTD_COMPRESSION_LEVEL`options of the `CHANGE REPLICATION SOURCE TO` statement (from MySQL 8.0.23) or `CHANGE MASTER TO` statement (before MySQL 8.0.23), or the `replica_compressed_protocol` or `slave_compressed_protocol` system variable. If you enable binary log transaction compression in a system where connection compression is also enabled, the impact of connection compression is reduced, as there might be little opportunity to further compress the compressed transaction payloads. However, connection compression can still operate on uncompressed events and on message headers. Binary log transaction compression can be enabled in combination with connection compression if you need to save storage space as well as network bandwidth. For more information on connection compression for replication connections, see Section 4.2.8, “Connection Compression Control”.

For Group Replication, compression is enabled by default for messages that exceed the threshold set by the `group_replication_compression_threshold` system variable. You can also configure compression for messages sent for distributed recovery by the method of state transfer from a donor's binary log, using the `group_replication_recovery_compression_algorithms` and `group_replication_recovery_zstd_compression_level` system variables. If you enable binary log transaction compression in a system where these are configured, Group Replication's message compression can still operate on uncompressed events and on message headers, but its impact is reduced. For more information on message compression for Group Replication, see Section 18.7.4, “Message Compression”.

5.4.5 The Slow Query Log

The slow query log consists of SQL statements that take more than `long_query_time` seconds to execute and require at least `min_examined_row_limit` rows to be examined. The slow query log can be used to find queries that take a long time to execute and are therefore candidates for optimization. However, examining a long slow query log can be a time-consuming task. To make this easier, you can use the `mysqldumpslow` command to process a slow query log file and summarize its contents. See Section 4.6.10, “mysqldumpslow — Summarize Slow Query Log Files”.

The time to acquire the initial locks is not counted as execution time. `mysqld` writes a statement to the slow query log after it has been executed and after all locks have been released, so log order might differ from execution order.

- Slow Query Log Parameters
- Slow Query Log Contents

Slow Query Log Parameters

The minimum and default values of `long_query_time` are 0 and 10, respectively. The value can be specified to a resolution of microseconds.

By default, administrative statements are not logged, nor are queries that do not use indexes for lookups. This behavior can be changed using `log_slow_admin_statements` and `log_queries_not_using_indexes`, as described later.

By default, the slow query log is disabled. To specify the initial slow query log state explicitly, use `--slow_query_log[={0|1}]`. With no argument or an argument of 1, `--slow_query_log` enables the log. With an argument of 0, this option disables the log. To specify a log file name, use `--slow_query_log_file=file_name`. To specify the log destination, use the `log_output` system variable (as described in Section 5.4.1, "Selecting General Query Log and Slow Query Log Output Destinations").

Note

If you specify the `TABLE` log destination, see Log Tables and "Too many open files" Errors.

If you specify no name for the slow query log file, the default name is *`host_name`*`-slow.log`. The server creates the file in the data directory unless an absolute path name is given to specify a different directory.

To disable or enable the slow query log or change the log file name at runtime, use the global `slow_query_log` and `slow_query_log_file` system variables. Set `slow_query_log` to 0 to disable the log or to 1 to enable it. Set `slow_query_log_file` to specify the name of the log file. If a log file already is open, it is closed and the new file is opened.

The server writes less information to the slow query log if you use the `--log-short-format` option.

To include slow administrative statements in the slow query log, enable the `log_slow_admin_statements` system variable. Administrative statements include `ALTER TABLE`, `ANALYZE TABLE`, `CHECK TABLE`, `CREATE INDEX`, `DROP INDEX`, `OPTIMIZE TABLE`, and `REPAIR TABLE`.

To include queries that do not use indexes for row lookups in the statements written to the slow query log, enable the `log_queries_not_using_indexes` system variable. (Even with that variable enabled, the server does not log queries that would not benefit from the presence of an index due to the table having fewer than two rows.)

When queries that do not use an index are logged, the slow query log may grow quickly. It is possible to put a rate limit on these queries by setting the `log_throttle_queries_not_using_indexes` system variable. By default, this variable is 0, which means there is no limit. Positive values impose a per-minute limit on logging of queries that do not use indexes. The first such query opens a 60-second window within which the server logs queries up to the given limit, then suppresses additional queries. If there are suppressed queries when the window ends, the server logs a summary that indicates how many there were and the aggregate time spent in them. The next 60-second window begins when the server logs the next query that does not use indexes.

The server uses the controlling parameters in the following order to determine whether to write a query to the slow query log:

1. The query must either not be an administrative statement, or `log_slow_admin_statements` must be enabled.

2. The query must have taken at least `long_query_time` seconds, or `log_queries_not_using_indexes` must be enabled and the query used no indexes for row lookups.

3. The query must have examined at least `min_examined_row_limit` rows.

4. The query must not be suppressed according to the `log_throttle_queries_not_using_indexes` setting.

The `log_timestamps` system variable controls the time zone of timestamps in messages written to the slow query log file (as well as to the general query log file and the error log). It does not affect the time zone of general query log and slow query log messages written to log tables, but rows retrieved from those tables can be converted from the local system time zone to any desired time zone with `CONVERT_TZ()` or by setting the session `time_zone` system variable.

By default, a replica does not write replicated queries to the slow query log. To change this, enable the system variable `log_slow_replica_statements` (from MySQL 8.0.26) or `log_slow_slave_statements` (before MySQL 8.0.26). Note that if row-based replication is in use (`binlog_format=ROW`), these system variables have no effect. Queries are only added to the replica's slow query log when they are logged in statement format in the binary log, that is, when `binlog_format=STATEMENT` is set, or when `binlog_format=MIXED` is set and the statement is logged in statement format. Slow queries that are logged in row format when `binlog_format=MIXED` is set, or that are logged when `binlog_format=ROW` is set, are not added to the replica's slow query log, even if `log_slow_replica_statements` or `log_slow_slave_statements` is enabled.

Slow Query Log Contents

When the slow query log is enabled, the server writes output to any destinations specified by the `log_output` system variable. If you enable the log, the server opens the log file and writes startup messages to it. However, further logging of queries to the file does not occur unless the `FILE` log destination is selected. If the destination is `NONE`, the server writes no queries even if the slow query log is enabled. Setting the log file name has no effect on logging if `FILE` is not selected as an output destination.

If the slow query log is enabled and `FILE` is selected as an output destination, each statement written to the log is preceded by a line that begins with a `#` character and has these fields (with all fields on a single line):

- `Query_time: duration`

  The statement execution time in seconds.

- `Lock_time: duration`

  The time to acquire locks in seconds.

- `Rows_sent: N`

  The number of rows sent to the client.

- `Rows_examined:`

  The number of rows examined by the server layer (not counting any processing internal to storage engines).

Enabling the `log_slow_extra` system variable (available as of MySQL 8.0.14) causes the server to write the following extra fields to `FILE` output in addition to those just listed (`TABLE` output is unaffected). Some field descriptions refer to status variable names. Consult the status variable descriptions for more information. However, in the slow query log, the counters are per-statement values, not cumulative per-session values.

- `Thread_id: ID`

  The statement thread identifier.

- `Errno: error_number`

  The statement error number, or 0 if no error occurred.

- `Killed: N`

  If the statement was terminated, the error number indicating why, or 0 if the statement terminated normally.

- `Bytes_received: N`

  The `Bytes_received` value for the statement.

- `Bytes_sent: N`

  The `Bytes_sent` value for the statement.

- `Read_first: N`

  The `Handler_read_first` value for the statement.

- `Read_last: N`

  The `Handler_read_last` value for the statement.

- `Read_key: N`

  The `Handler_read_key` value for the statement.

- `Read_next: N`

  The `Handler_read_next` value for the statement.

- `Read_prev: N`

  The `Handler_read_prev` value for the statement.

- `Read_rnd: N`

  The `Handler_read_rnd` value for the statement.

- `Read_rnd_next: N`

  The `Handler_read_rnd_next` value for the statement.

- `Sort_merge_passes: N`

  The `Sort_merge_passes` value for the statement.

- `Sort_range_count: N`

  The `Sort_range` value for the statement.

- `Sort_rows: N`

  The `Sort_rows` value for the statement.

- `Sort_scan_count: N`

  The `Sort_scan` value for the statement.

- `Created_tmp_disk_tables: N`

  The `Created_tmp_disk_tables` value for the statement.

- `Created_tmp_tables: N`

  The `Created_tmp_tables` value for the statement.

- `Start: timestamp`

  The statement execution start time.

- `End: timestamp`

  The statement execution end time.

A given slow query log file may contain a mix of lines with and without the extra fields added by enabling `log_slow_extra`. Log file analyzers can determine whether a line contains the additional fields by the field count.

Each statement written to the slow query log file is preceded by a `SET` statement that includes a timestamp. As of MySQL 8.0.14, the timestamp indicates when the slow statement began executing. Prior to 8.0.14, the timestamp indicates when the slow statement was logged (which occurs after the statement finishes executing).

Passwords in statements written to the slow query log are rewritten by the server not to occur literally in plain text. See Section 6.1.2.3, “Passwords and Logging”.

From MySQL 8.0.29, statements that cannot be parsed (due, for example, to syntax errors) are not written to the slow query log.

5.4.6 Server Log Maintenance

As described in Section 5.4, “MySQL Server Logs”, MySQL Server can create several different log files to help you see what activity is taking place. However, you must clean up these files regularly to ensure that the logs do not take up too much disk space.

When using MySQL with logging enabled, you may want to back up and remove old log files from time to time and tell MySQL to start logging to new files. See Section 7.2, “Database Backup Methods”.

On a Linux (Red Hat) installation, you can use the `mysql-log-rotate` script for log maintenance. If you installed MySQL from an RPM distribution, this script should have been installed automatically. Be careful with this script if you are using the binary log for replication. You should not remove binary logs until you are certain that their contents have been processed by all replicas.

On other systems, you must install a short script yourself that you start from `cron` (or its equivalent) for handling log files.

Binary log files are automatically removed after the server's binary log expiration period. Removal of the files can take place at startup and when the binary log is flushed. The default binary log expiration period is 30 days. To specify an alternative expiration period, use the `binlog_expire_logs_seconds` system variable. If you are using replication, you should specify an expiration period that is no lower than the maximum amount of time your replicas might lag behind the source. To remove binary logs on demand, use the `PURGE BINARY LOGS` statement (see Section 13.4.1.1, “PURGE BINARY LOGS Statement”).

To force MySQL to start using new log files, flush the logs. Log flushing occurs when you execute a `FLUSH LOGS` statement or a `mysqladmin flush-logs`, `mysqladmin refresh`, `mysqldump --flush-logs`, or `mysqldump --master-data` command. See Section 13.7.8.3, “FLUSH Statement”, Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”, and Section 4.5.4, “mysqldump — A Database Backup Program”. In addition, the server flushes the binary log automatically when current binary log file size reaches the value of the `max_binlog_size` system variable.

`FLUSH LOGS` supports optional modifiers to enable selective flushing of individual logs (for example, `FLUSH BINARY LOGS`). See Section 13.7.8.3, “FLUSH Statement”.

A log-flushing operation has the following effects:

- If binary logging is enabled, the server closes the current binary log file and opens a new log file with the next sequence number.
- If general query logging or slow query logging to a log file is enabled, the server closes and reopens the log file.
- If the server was started with the `--log-error` option to cause the error log to be written to a file, the server closes and reopens the log file.

Execution of log-flushing statements or commands requires connecting to the server using an account that has the `RELOAD` privilege. On Unix and Unix-like systems, another way to flush the logs is to send a signal to the server, which can be done by `root` or the account that owns the server process. (See Section 4.10, “Unix Signal Handling in MySQL”.) Signals enable log flushing to be performed without having to connect to the server:

- A `SIGHUP` signal flushes all the logs. However, `SIGHUP` has additional effects other than log flushing that might be undesirable.
- As of MySQL 8.0.19, `SIGUSR1` causes the server to flush the error log, general query log, and slow query log. If you are interested in flushing only those logs, `SIGUSR1` can be used as a more “lightweight” signal that does not have the `SIGHUP` effects that are unrelated to logs.

As mentioned previously, flushing the binary log creates a new binary log file, whereas flushing the general query log, slow query log, or error log just closes and reopens the log file. For the latter logs, to cause a new log file to be created on Unix, rename the current log file first before flushing it. At flush time, the server opens the new log file with the original name. For example, if the general query log, slow query log, and error log files are named `mysql.log`, `mysql-slow.log`, and `err.log`, you can use a series of commands like this from the command line:

```
cd mysql-data-directory
mv mysql.log mysql.log.old
mv mysql-slow.log mysql-slow.log.old
mv err.log err.log.old
mysqladmin flush-logs
```

On Windows, use `rename` rather than `mv`.

At this point, you can make a backup of `mysql.log.old`, `mysql-slow.log.old`, and `err.log.old`, then remove them from disk.

To rename the general query log or slow query log at runtime, first connect to the server and disable the log:

```
SET GLOBAL general_log = 'OFF';
SET GLOBAL slow_query_log = 'OFF';
```

With the logs disabled, rename the log files externally (for example, from the command line). Then enable the logs again:

```
SET GLOBAL general_log = 'ON';
SET GLOBAL slow_query_log = 'ON';
```

This method works on any platform and does not require a server restart.

Note

For the server to recreate a given log file after you have renamed the file externally, the file location must be writable by the server. This may not always

be the case. For example, on Linux, the server might write the error log as `/var/log/mysqld.log`, where `/var/log` is owned by `root` and not writable by `mysqld`. In this case, log-flushing operations fail to create a new log file.

To handle this situation, you must manually create the new log file with the proper ownership after renaming the original log file. For example, execute these commands as `root`:

```
mv /var/log/mysqld.log /var/log/mysqld.log.old
install -omysql -gmysql -m0644 /dev/null /var/log/mysqld.log
```

5.5 MySQL Components

MySQL Server includes a component-based infrastructure for extending server capabilities. A component provides services that are available to the server and other components. (With respect to service use, the server is a component, equal to other components.) Components interact with each other only through the services they provide.

MySQL distributions include several components that implement server extensions:

- Components for configuring error logging. See Section 5.4.2, “The Error Log”, and Section 5.5.3, “Error Log Components”.
- A component for checking passwords. See Section 6.4.3, “The Password Validation Component”.
- Keyring components provide secure storage for sensitive information. See Section 6.4.4, “The MySQL Keyring”.
- A component that enables applications to add their own message events to the audit log. See Section 6.4.6, “The Audit Message Component”.
- A component that implements a loadable function for accessing query attributes. See Section 9.6, “Query Attributes”.

System and status variables implemented by a component are exposed when the component is installed and have names that begin with a component-specific prefix. For example, the `log_filter_dragnet` error log filter component implements a system variable named `log_error_filter_rules`, the full name of which is `dragnet.log_error_filter_rules`. To refer to this variable, use the full name.

The following sections describe how to install and uninstall components, and how to determine at runtime which components are installed and obtain information about them.

For information about the internal implementation of components, see the MySQL Server Doxygen documentation, available at https://dev.mysql.com/doc/index-other.html. For example, if you intend to write your own components, this information is important for understanding how components work.

5.5.1 Installing and Uninstalling Components

Components must be loaded into the server before they can be used. MySQL supports manual component loading at runtime and automatic loading during server startup.

While a component is loaded, information about it is available as described in Section 5.5.2, “Obtaining Component Information”.

The `INSTALL COMPONENT` and `UNINSTALL COMPONENT` SQL statements enable component loading and unloading. For example:

```
INSTALL COMPONENT 'file://component_validate_password';
UNINSTALL COMPONENT 'file://component_validate_password';
```

A loader service handles component loading and unloading, and also registers loaded components in the `mysql.component` system table.

The SQL statements for component manipulation affect server operation and the `mysql.component` system table as follows:

- `INSTALL COMPONENT` loads components into the server. The components become active immediately. The loader service also registers loaded components in the `mysql.component` system table. For subsequent server restarts, the loader service loads any components listed in `mysql.component` during the startup sequence. This occurs even if the server is started with the `--skip-grant-tables` option.
- `UNINSTALL COMPONENT` deactivates components and unloads them from the server. The loader service also unregisters the components from the `mysql.component` system table so that the server no longer loads them during its startup sequence for subsequent restarts.

Compared to the corresponding `INSTALL PLUGIN` statement for server plugins, the `INSTALL COMPONENT` statement for components offers the significant advantage that it is not necessary to know any platform-specific file name suffix for naming the component. This means that a given `INSTALL COMPONENT` statement can be executed uniformly across platforms.

A component when installed may also automatically install related loadable functions. If so, the component when uninstalled also automatically uninstalls those functions.

5.5.2 Obtaining Component Information

The `mysql.component` system table contains information about currently loaded components and shows which components have been registered using `INSTALL COMPONENT`. Selecting from the table shows which components are installed. For example:

```
mysql> SELECT * FROM mysql.component;
+--------------+--------------------+----------------------------------+
| component_id | component_group_id | component_urn                    |
+--------------+--------------------+----------------------------------+
|            1 |                  1 | file://component_validate_password |
|            2 |                  2 | file://component_log_sink_json     |
+--------------+--------------------+----------------------------------+
```

The `component_id` and `component_group_id` values are for internal use. The `component_urn` is the URN used in `INSTALL COMPONENT` and `UNINSTALL COMPONENT` statements to load and unload the component.

5.5.3 Error Log Components

This section describes the characteristics of individual error log components. For general information about configuring error logging, see Section 5.4.2, “The Error Log”.

A log component can be a filter or a sink:

- A filter processes log events, to add, remove, or modify event fields, or to delete events entirely. The resulting events pass to the next log component in the list of enabled components.
- A sink is a destination (writer) for log events. Typically, a sink processes log events into log messages that have a particular format and writes these messages to its associated output, such as a file or the system log. A sink may also write to the Performance Schema `error_log` table; see Section 27.12.21.1, “The error_log Table”. Events pass unmodified to the next log component in the list of enabled components (that is, although a sink formats events to produce output messages, it does not modify events as they pass internally to the next component).

The `log_error_services` system variable value lists the enabled log components. Components not named in the list are disabled. From MySQL 8.0.30, `log_error_services` also implicitly loads error

log components if they are not already loaded. For more information, see Section 5.4.2.1, “Error Log Configuration”.

The following sections describe individual log components, grouped by component type:

- Filter Error Log Components
- Sink Error Log Components

Component descriptions include these types of information:

- The component name and intended purpose.
- Whether the component is built in or must be loaded. For a loadable component, the description specifies the URN to use if explicitly loading or unloading the component with the `INSTALL COMPONENT` and `UNINSTALL COMPONENT` statements. Implicitly loading error log components requires only the component name. For more information, see Section 5.4.2.1, “Error Log Configuration”.
- Whether the component can be listed multiple times in the `log_error_services` value.
- For a sink component, the destination to which the component writes output.
- For a sink component, whether it supports an interface to the Performance Schema `error_log` table.

Filter Error Log Components

Error log filter components implement filtering of error log events. If no filter component is enabled, no filtering occurs.

Any enabled filter component affects log events only for components listed later in the `log_error_services` value. In particular, for any log sink component listed in `log_error_services` earlier than any filter component, no log event filtering occurs.

The log_filter_internal Component

- Purpose: Implements filtering based on log event priority and error code, in combination with the `log_error_verbosity` and `log_error_suppression_list` system variables. See Section 5.4.2.5, “Priority-Based Error Log Filtering (log_filter_internal)”.
- URN: This component is built in and need not be loaded.
- Multiple uses permitted: No.

If `log_filter_internal` is disabled, `log_error_verbosity` and `log_error_suppression_list` have no effect.

The log_filter_dragnet Component

- Purpose: Implements filtering based on the rules defined by the `dragnet.log_error_filter_rules` system variable setting. See Section 5.4.2.6, “Rule-Based Error Log Filtering (log_filter_dragnet)”.
- URN: `file://component_log_filter_dragnet`
- Multiple uses permitted: No.

Sink Error Log Components

Error log sink components are writers that implement error log output. If no sink component is enabled, no log output occurs.

Some sink component descriptions refer to the default error log destination. This is the console or a file and is indicated by the value of the `log_error` system variable, determined as described in Section 5.4.2.2, “Default Error Log Destination Configuration”.

The log_sink_internal Component

- Purpose: Implements traditional error log message output format.
- URN: This component is built in and need not be loaded.
- Multiple uses permitted: No.
- Output destination: Writes to the default error log destination.
- Performance Schema support: Writes to the `error_log` table. Provides a parser for reading error log files created by previous server instances.

The log_sink_json Component

- Purpose: Implements JSON-format error logging. See Section 5.4.2.7, “Error Logging in JSON Format”.
- URN: `file://component_log_sink_json`
- Multiple uses permitted: Yes.
- Output destination: This sink determines its output destination based on the default error log destination, which is given by the `log_error` system variable:
  - If `log_error` names a file, the sink bases output file naming on that file name, plus a numbered *`.NN`*`.json` suffix, with *`NN`* starting at 00. For example, if `log_error` is *`file_name`*, successive instances of `log_sink_json` named in the `log_error_services` value write to *`file_name`*`.00.json`, *`file_name`*`.01.json`, and so forth.
  - If `log_error` is `stderr`, the sink writes to the console. If `log_sink_json` is named multiple times in the `log_error_services` value, they all write to the console, which is likely not useful.
- Performance Schema support: Writes to the `error_log` table. Provides a parser for reading error log files created by previous server instances.

The log_sink_syseventlog Component

- Purpose: Implements error logging to the system log. This is the Event Log on Windows, and `syslog` on Unix and Unix-like systems. See Section 5.4.2.8, “Error Logging to the System Log”.
- URN: `file://component_log_sink_syseventlog`
- Multiple uses permitted: No.
- Output destination: Writes to the system log. Does not use the default error log destination.
- Performance Schema support: Does not write to the `error_log` table. Does not provide a parser for reading error log files created by previous server instances.

The log_sink_test Component

- Purpose: Intended for internal use in writing test cases, not for production use.
- URN: `file://component_log_sink_test`

Sink properties such as whether multiple uses are permitted and the output destination are not specified for `log_sink_test` because, as mentioned, it is for internal use. As such, its behavior is subject to change at any time.

5.5.4 Query Attribute Components

As of MySQL 8.0.23, a component service provides access to query attributes (see Section 9.6, “Query Attributes”). The `query_attributes` component uses this service to provide access to query attributes within SQL statements.

- Purpose: Implements the `mysql_query_attribute_string()` function that takes an attribute name argument and returns the attribute value as a string, or `NULL` if the attribute does not exist.
- URN: `file://component_query_attributes`

Developers who wish to incorporate the same query-attribute component service used by `query_attributes` should consult the `mysql_query_attributes.h` file in a MySQL source distribution.

5.6 MySQL Server Plugins

MySQL supports an plugin API that enables creation of server plugins. Plugins can be loaded at server startup, or loaded and unloaded at runtime without restarting the server. The plugins supported by this interface include, but are not limited to, storage engines, `INFORMATION_SCHEMA` tables, full-text parser plugins, and server extensions.

MySQL distributions include several plugins that implement server extensions:

- Plugins for authenticating attempts by clients to connect to MySQL Server. Plugins are available for several authentication protocols. See Section 6.2.17, “Pluggable Authentication”.
- A connection-control plugin that enables administrators to introduce an increasing delay after a certain number of consecutive failed client connection attempts. See Section 6.4.2, “The Connection-Control Plugins”.
- A password-validation plugin implements password strength policies and assesses the strength of potential passwords. See Section 6.4.3, “The Password Validation Component”.
- Semisynchronous replication plugins implement an interface to replication capabilities that permit the source to proceed as long as at least one replica has responded to each transaction. See Section 17.4.10, “Semisynchronous Replication”.
- Group Replication enables you to create a highly available distributed MySQL service across a group of MySQL server instances, with data consistency, conflict detection and resolution, and group membership services all built-in. See Chapter 18, *Group Replication*.
- MySQL Enterprise Edition includes a thread pool plugin that manages connection threads to increase server performance by efficiently managing statement execution threads for large numbers of client connections. See Section 5.6.3, “MySQL Enterprise Thread Pool”.
- MySQL Enterprise Edition includes an audit plugin for monitoring and logging of connection and query activity. See Section 6.4.5, “MySQL Enterprise Audit”.
- MySQL Enterprise Edition includes a firewall plugin that implements an application-level firewall to enable database administrators to permit or deny SQL statement execution based on matching against allowlists of accepted statement patterns. See Section 6.4.7, “MySQL Enterprise Firewall”.
- Query rewrite plugins examine statements received by MySQL Server and possibly rewrite them before the server executes them. See Section 5.6.4, “The Rewriter Query Rewrite Plugin”, and Section 5.6.5, “The ddl_rewriter Plugin”.
- Version Tokens enables creation of and synchronization around server tokens that applications can use to prevent accessing incorrect or out-of-date data. Version Tokens is based on a plugin library that implements a `version_tokens` plugin and a set of loadable functions. See Section 5.6.6, “Version Tokens”.

- Keyring plugins provide secure storage for sensitive information. See Section 6.4.4, “The MySQL Keyring”.

  In MySQL 8.0.24, MySQL Keyring began transitioning from plugins to use the component infrastructure, facilitated using the plugin named `daemon_keyring_proxy_plugin` that acts as a bridge between the plugin and component service APIs. See Section 5.6.8, “The Keyring Proxy Bridge Plugin”.

- X Plugin extends MySQL Server to be able to function as a document store. Running X Plugin enables MySQL Server to communicate with clients using the X Protocol, which is designed to expose the ACID compliant storage abilities of MySQL as a document store. See Section 20.5, “X Plugin”.

- Clone permits cloning `InnoDB` data from a local or remote MySQL server instance. See Section 5.6.7, “The Clone Plugin”.

- Test framework plugins test server services. For information about these plugins, see the Plugins for Testing Plugin Services section of the MySQL Server Doxygen documentation, available at https://dev.mysql.com/doc/index-other.html.

The following sections describe how to install and uninstall plugins, and how to determine at runtime which plugins are installed and obtain information about them. For information about writing plugins, see The MySQL Plugin API.

5.6.1 Installing and Uninstalling Plugins

Server plugins must be loaded into the server before they can be used. MySQL supports plugin loading at server startup and runtime. It is also possible to control the activation state of loaded plugins at startup, and to unload them at runtime.

While a plugin is loaded, information about it is available as described in Section 5.6.2, “Obtaining Server Plugin Information”.

- Installing Plugins
- Controlling Plugin Activation State
- Uninstalling Plugins
- Plugins and Loadable Functions

Installing Plugins

Before a server plugin can be used, it must be installed using one of the following methods. In the descriptions, *`plugin_name`* stands for a plugin name such as `innodb`, `csv`, or `validate_password`.

- Built-in Plugins
- Plugins Registered in the mysql.plugin System Table
- Plugins Named with Command-Line Options
- Plugins Installed with the INSTALL PLUGIN Statement

Built-in Plugins

A built-in plugin is known by the server automatically. By default, the server enables the plugin at startup. Some built-in plugins permit this to be changed with the *`--plugin_name[=activation_state]`* option.

Plugins Registered in the mysql.plugin System Table

The `mysql.plugin` system table serves as a registry of plugins (other than built-in plugins, which need not be registered). During the normal startup sequence, the server loads plugins registered in the table. By default, for a plugin loaded from the `mysql.plugin` table, the server also enables the plugin. This can be changed with the `--plugin_name[=activation_state]` option.

If the server is started with the `--skip-grant-tables` option, plugins registered in the `mysql.plugin` table are not loaded and are unavailable.

Plugins Named with Command-Line Options

A plugin located in a plugin library file can be loaded at server startup with the `--plugin-load`, `--plugin-load-add`, or `--early-plugin-load` option. Normally, for a plugin loaded at startup, the server also enables the plugin. This can be changed with the `--plugin_name[=activation_state]` option.

The `--plugin-load` and `--plugin-load-add` options load plugins after built-in plugins and storage engines have initialized during the server startup sequence. The `--early-plugin-load` option is used to load plugins that must be available prior to initialization of built-in plugins and storage engines.

The value of each plugin-loading option is a semicolon-separated list of *`plugin_library`* and *`name=plugin_library`* values. Each *`plugin_library`* is the name of a library file that contains plugin code, and each *`name`* is the name of a plugin to load. If a plugin library is named without any preceding plugin name, the server loads all plugins in the library. With a preceding plugin name, the server loads only the named plugin from the library. The server looks for plugin library files in the directory named by the `plugin_dir` system variable.

Plugin-loading options do not register any plugin in the `mysql.plugin` table. For subsequent restarts, the server loads the plugin again only if `--plugin-load`, `--plugin-load-add`, or `--early-plugin-load` is given again. That is, the option produces a one-time plugin-installation operation that persists for a single server invocation.

`--plugin-load`, `--plugin-load-add`, and `--early-plugin-load` enable plugins to be loaded even when `--skip-grant-tables` is given (which causes the server to ignore the `mysql.plugin` table). `--plugin-load`, `--plugin-load-add`, and `--early-plugin-load` also enable plugins to be loaded at startup that cannot be loaded at runtime.

The `--plugin-load-add` option complements the `--plugin-load` option:

- Each instance of `--plugin-load` resets the set of plugins to load at startup, whereas `--plugin-load-add` adds a plugin or plugins to the set of plugins to be loaded without resetting the current set. Consequently, if multiple instances of `--plugin-load` are specified, only the last one applies. With multiple instances of `--plugin-load-add`, all of them apply.
- The argument format is the same as for `--plugin-load`, but multiple instances of `--plugin-load-add` can be used to avoid specifying a large set of plugins as a single long unwieldy `--plugin-load` argument.
- `--plugin-load-add` can be given in the absence of `--plugin-load`, but any instance of `--plugin-load-add` that appears before `--plugin-load` has no effect because `--plugin-load` resets the set of plugins to load.

For example, these options:

```
--plugin-load=x --plugin-load-add=y
```

are equivalent to these options:

```
--plugin-load-add=x --plugin-load-add=y
```

and are also equivalent to this option:

```
--plugin-load="x;y"
```

But these options:

```
--plugin-load-add=y --plugin-load=x
```

are equivalent to this option:

```
--plugin-load=x
```

Plugins Installed with the INSTALL PLUGIN Statement

A plugin located in a plugin library file can be loaded at runtime with the `INSTALL PLUGIN` statement. The statement also registers the plugin in the `mysql.plugin` table to cause the server to load it on subsequent restarts. For this reason, `INSTALL PLUGIN` requires the `INSERT` privilege for the `mysql.plugin` table.

The plugin library file base name depends on your platform. Common suffixes are `.so` for Unix and Unix-like systems, `.dll` for Windows.

Example: The `--plugin-load-add` option installs a plugin at server startup. To install a plugin named `myplugin` from a plugin library file named `somepluglib.so`, use these lines in a `my.cnf` file:

```
[mysqld]
plugin-load-add=myplugin=somepluglib.so
```

In this case, the plugin is not registered in `mysql.plugin`. Restarting the server without the `--plugin-load-add` option causes the plugin not to be loaded at startup.

Alternatively, the `INSTALL PLUGIN` statement causes the server to load the plugin code from the library file at runtime:

```
INSTALL PLUGIN myplugin SONAME 'somepluglib.so';
```

`INSTALL PLUGIN` also causes “permanent” plugin registration: The plugin is listed in the `mysql.plugin` table to ensure that the server loads it on subsequent restarts.

Many plugins can be loaded either at server startup or at runtime. However, if a plugin is designed such that it must be loaded and initialized during server startup, attempts to load it at runtime using `INSTALL PLUGIN` produce an error:

```
mysql> INSTALL PLUGIN myplugin SONAME 'somepluglib.so';
ERROR 1721 (HY000): Plugin 'myplugin' is marked as not dynamically
installable. You have to stop the server to install it.
```

In this case, you must use `--plugin-load`, `--plugin-load-add`, or `--early-plugin-load`.

If a plugin is named both using a `--plugin-load`, `--plugin-load-add`, or `--early-plugin-load` option and (as a result of an earlier `INSTALL PLUGIN` statement) in the `mysql.plugin` table, the server starts but writes these messages to the error log:

```
[ERROR] Function 'plugin_name' already exists
[Warning] Couldn't load plugin named 'plugin_name'
with soname 'plugin_object_file'.
```

Controlling Plugin Activation State

If the server knows about a plugin when it starts (for example, because the plugin is named using a `--plugin-load-add` option or is registered in the `mysql.plugin` table), the server loads and enables the plugin by default. It is possible to control activation state for such a plugin using a

--plugin_name[=*activation_state*] startup option, where *plugin_name* is the name of the plugin to affect, such as innodb, csv, or validate_password. As with other options, dashes and underscores are interchangeable in option names. Also, activation state values are not case-sensitive. For example, --my_plugin=ON and --my-plugin=on are equivalent.

- *--plugin_name*=OFF

  Tells the server to disable the plugin. This may not be possible for certain built-in plugins, such as mysql_native_password.

- *--plugin_name*[=ON]

  Tells the server to enable the plugin. (Specifying the option as *--plugin_name* without a value has the same effect.) If the plugin fails to initialize, the server runs with the plugin disabled.

- *--plugin_name*=FORCE

  Tells the server to enable the plugin, but if plugin initialization fails, the server does not start. In other words, this option forces the server to run with the plugin enabled or not at all.

- *--plugin_name*=FORCE_PLUS_PERMANENT

  Like FORCE, but in addition prevents the plugin from being unloaded at runtime. If a user attempts to do so with UNINSTALL PLUGIN, an error occurs.

Plugin activation states are visible in the LOAD_OPTION column of the Information Schema PLUGINS table.

Suppose that CSV, BLACKHOLE, and ARCHIVE are built-in pluggable storage engines and that you want the server to load them at startup, subject to these conditions: The server is permitted to run if CSV initialization fails, must require that BLACKHOLE initialization succeeds, and should disable ARCHIVE. To accomplish that, use these lines in an option file:

```
[mysqld]
csv=ON
blackhole=FORCE
archive=OFF
```

The --enable-*plugin_name* option format is a synonym for *--plugin_name*=ON. The --disable-*plugin_name* and --skip-*plugin_name* option formats are synonyms for *--plugin_name*=OFF.

If a plugin is disabled, either explicitly with OFF or implicitly because it was enabled with ON but fails to initialize, aspects of server operation requiring the plugin change. For example, if the plugin implements a storage engine, existing tables for the storage engine become inaccessible, and attempts to create new tables for the storage engine result in tables that use the default storage engine unless the NO_ENGINE_SUBSTITUTION SQL mode is enabled to cause an error to occur instead.

Disabling a plugin may require adjustment to other options. For example, if you start the server using --skip-innodb to disable InnoDB, other innodb_*xxx* options likely also need to be omitted at startup. In addition, because InnoDB is the default storage engine, it cannot start unless you specify another available storage engine with --default_storage_engine. You must also set --default_tmp_storage_engine.

Uninstalling Plugins

At runtime, the UNINSTALL PLUGIN statement disables and uninstalls a plugin known to the server. The statement unloads the plugin and removes it from the mysql.plugin system table, if it is registered there. For this reason, UNINSTALL PLUGIN statement requires the DELETE privilege for the mysql.plugin table. With the plugin no longer registered in the table, the server does not load the plugin during subsequent restarts.

`UNINSTALL PLUGIN` can unload a plugin regardless of whether it was loaded at runtime with `INSTALL PLUGIN` or at startup with a plugin-loading option, subject to these conditions:

- It cannot unload plugins that are built in to the server. These can be identified as those that have a library name of `NULL` in the output from the Information Schema `PLUGINS` table or `SHOW PLUGINS`.
- It cannot unload plugins for which the server was started with `--plugin_name=FORCE_PLUS_PERMANENT`, which prevents plugin unloading at runtime. These can be identified from the `LOAD_OPTION` column of the `PLUGINS` table.

To uninstall a plugin that currently is loaded at server startup with a plugin-loading option, use this procedure.

1. Remove from the `my.cnf` file any options and system variables related to the plugin. If any plugin system variables were persisted to the `mysqld-auto.cnf` file, remove them using `RESET PERSIST var_name` for each one to remove it.
2. Restart the server.
3. Plugins normally are installed using either a plugin-loading option at startup or with `INSTALL PLUGIN` at runtime, but not both. However, removing options for a plugin from the `my.cnf` file may not be sufficient to uninstall it if at some point `INSTALL PLUGIN` has also been used. If the plugin still appears in the output from `PLUGINS` or `SHOW PLUGINS`, use `UNINSTALL PLUGIN` to remove it from the `mysql.plugin` table. Then restart the server again.

Plugins and Loadable Functions

A plugin when installed may also automatically install related loadable functions. If so, the plugin when uninstalled also automatically uninstalls those functions.

5.6.2 Obtaining Server Plugin Information

There are several ways to determine which plugins are installed in the server:

- The Information Schema `PLUGINS` table contains a row for each loaded plugin. Any that have a `PLUGIN_LIBRARY` value of `NULL` are built in and cannot be unloaded.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.PLUGINS\G
*************************** 1. row ***************************
           PLUGIN_NAME: binlog
        PLUGIN_VERSION: 1.0
         PLUGIN_STATUS: ACTIVE
           PLUGIN_TYPE: STORAGE ENGINE
   PLUGIN_TYPE_VERSION: 50158.0
        PLUGIN_LIBRARY: NULL
PLUGIN_LIBRARY_VERSION: NULL
         PLUGIN_AUTHOR: Oracle Corporation
    PLUGIN_DESCRIPTION: This is a pseudo storage engine to represent the binlog in a transaction
        PLUGIN_LICENSE: GPL
           LOAD_OPTION: FORCE
...
*************************** 10. row ***************************
           PLUGIN_NAME: InnoDB
        PLUGIN_VERSION: 1.0
         PLUGIN_STATUS: ACTIVE
           PLUGIN_TYPE: STORAGE ENGINE
   PLUGIN_TYPE_VERSION: 50158.0
        PLUGIN_LIBRARY: ha_innodb_plugin.so
PLUGIN_LIBRARY_VERSION: 1.0
         PLUGIN_AUTHOR: Oracle Corporation
    PLUGIN_DESCRIPTION: Supports transactions, row-level locking,
                        and foreign keys
        PLUGIN_LICENSE: GPL
           LOAD_OPTION: ON
...
```

- The `SHOW PLUGINS` statement displays a row for each loaded plugin. Any that have a `Library` value of `NULL` are built in and cannot be unloaded.

```
mysql> SHOW PLUGINS\G
*************************** 1. row ***************************
   Name: binlog
 Status: ACTIVE
   Type: STORAGE ENGINE
Library: NULL
License: GPL
...
*************************** 10. row ***************************
   Name: InnoDB
 Status: ACTIVE
   Type: STORAGE ENGINE
Library: ha_innodb_plugin.so
License: GPL
...
```

- The `mysql.plugin` table shows which plugins have been registered with `INSTALL PLUGIN`. The table contains only plugin names and library file names, so it does not provide as much information as the `PLUGINS` table or the `SHOW PLUGINS` statement.

5.6.3 MySQL Enterprise Thread Pool

Note

MySQL Enterprise Thread Pool is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, https://www.mysql.com/products/.

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. The 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.

The thread pool addresses several problems of the model that uses one thread per connection:

- Too many thread stacks make CPU caches almost useless in highly parallel execution workloads. The thread pool promotes thread stack reuse to minimize the CPU cache footprint.
- With too many threads executing in parallel, context switching overhead is high. This also presents a challenge to the operating system scheduler. The thread pool controls the number of active threads to keep the parallelism within the MySQL server at a level that it can handle and that is appropriate for the server host on which MySQL is executing.
- Too many transactions executing in parallel increases resource contention. In `InnoDB`, this increases the time spent holding central mutexes. The thread pool controls when transactions start to ensure that not too many execute in parallel.

Additional Resources

Section A.15, “MySQL 8.0 FAQ: MySQL Enterprise Thread Pool”

5.6.3.1 Thread Pool Elements

MySQL Enterprise Thread Pool comprises these elements:

- A plugin library file implements a plugin for the thread pool code as well as several associated monitoring tables that provide information about thread pool operation:

- As of MySQL 8.0.14, the monitoring tables are Performance Schema tables; see Section 27.12.16, “Performance Schema Thread Pool Tables”.
- Prior to MySQL 8.0.14, the monitoring tables are `INFORMATION_SCHEMA` tables; see Section 26.5, “INFORMATION_SCHEMA Thread Pool Tables”.

  The `INFORMATION_SCHEMA` tables now are deprecated; expect them to be removed in a future version of MySQL. Applications should transition away from the `INFORMATION_SCHEMA` tables to the Performance Schema tables. For example, if an application uses this query:

  ```
  SELECT * FROM INFORMATION_SCHEMA.TP_THREAD_STATE;
  ```

  The application should use this query instead:

  ```
  SELECT * FROM performance_schema.tp_thread_state;
  ```

  Note

  If you do not load all the monitoring tables, some or all MySQL Enterprise Monitor thread pool graphs may be empty.

  For a detailed description of how the thread pool works, see Section 5.6.3.3, “Thread Pool Operation”.

- Several system variables are related to the thread pool. The `thread_handling` system variable has a value of `loaded-dynamically` when the server successfully loads the thread pool plugin.

  The other related system variables are implemented by the thread pool plugin and are not available unless it is enabled. For information about using these variables, see Section 5.6.3.3, “Thread Pool Operation”, and Section 5.6.3.4, “Thread Pool Tuning”.

- The Performance Schema has instruments that expose information about the thread pool and may be used to investigate operational performance. To identify them, use this query:

  ```
  SELECT * FROM performance_schema.setup_instruments
  WHERE NAME LIKE '%thread_pool%';
  ```

  For more information, see Chapter 27, *MySQL Performance Schema*.

5.6.3.2 Thread Pool Installation

This section describes how to install MySQL Enterprise Thread Pool. For general information about installing plugins, see Section 5.6.1, “Installing and Uninstalling Plugins”.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The plugin library file base name is `thread_pool`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

- Thread Pool Installation as of MySQL 8.0.14
- Thread Pool Installation Prior to MySQL 8.0.14

Thread Pool Installation as of MySQL 8.0.14

In MySQL 8.0.14 and higher, the thread pool monitoring tables are Performance Schema tables that are loaded and unloaded along with the thread pool plugin. The `INFORMATION_SCHEMA` versions of the tables are deprecated but still available; they are installed per the instructions in Thread Pool Installation Prior to MySQL 8.0.14.

To enable thread pool capability, load the plugin by starting the server with the `--plugin-load-add` option. To do this, put these lines in the server `my.cnf` file, adjusting the `.so` suffix for your platform as necessary:

```
[mysqld]
plugin-load-add=thread_pool.so
```

To verify plugin installation, examine the Information Schema `PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, "Obtaining Server Plugin Information"). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE 'thread%';
+----------------------+---------------+
| PLUGIN_NAME          | PLUGIN_STATUS |
+----------------------+---------------+
| thread_pool          | ACTIVE        |
+----------------------+---------------+
```

To verify that the Performance Schema monitoring tables are available, examine the Information Schema `TABLES` table or use the `SHOW TABLES` statement. For example:

```
mysql> SELECT TABLE_NAME
       FROM INFORMATION_SCHEMA.TABLES
       WHERE TABLE_SCHEMA = 'performance_schema'
       AND TABLE_NAME LIKE 'tp%';
+-----------------------+
| TABLE_NAME            |
+-----------------------+
| tp_thread_group_state |
| tp_thread_group_stats |
| tp_thread_state       |
+-----------------------+
```

If the server loads the thread pool plugin successfully, it sets the `thread_handling` system variable to `loaded-dynamically`.

If the plugin fails to initialize, check the server error log for diagnostic messages.

Thread Pool Installation Prior to MySQL 8.0.14

Prior to MySQL 8.0.14, the thread pool monitoring tables are plugins separate from the thread pool plugin and can be installed separately.

To enable thread pool capability, load the plugins to be used by starting the server with the `--plugin-load-add` option. For example, if you name only the plugin library file, the server loads all plugins that it contains (that is, the thread pool plugin and all the `INFORMATION_SCHEMA` tables). To do this, put these lines in the server `my.cnf` file, adjusting the `.so` suffix for your platform as necessary:

```
[mysqld]
plugin-load-add=thread_pool.so
```

That is equivalent to loading all thread pool plugins by naming them individually:

```
[mysqld]
plugin-load-add=thread_pool=thread_pool.so
plugin-load-add=tp_thread_state=thread_pool.so
plugin-load-add=tp_thread_group_state=thread_pool.so
plugin-load-add=tp_thread_group_stats=thread_pool.so
```

If desired, you can load individual plugins from the library file. To load the thread pool plugin but not the `INFORMATION_SCHEMA` tables, use an option like this:

```
[mysqld]
plugin-load-add=thread_pool=thread_pool.so
```

To load the thread pool plugin and only the `TP_THREAD_STATE` `INFORMATION_SCHEMA` table, use options like this:

```
[mysqld]
plugin-load-add=thread_pool=thread_pool.so
plugin-load-add=tp_thread_state=thread_pool.so
```

To verify plugin installation, examine the Information Schema `PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, "Obtaining Server Plugin Information"). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE 'thread%' OR PLUGIN_NAME LIKE 'tp%';
+----------------------+---------------+
| PLUGIN_NAME          | PLUGIN_STATUS |
+----------------------+---------------+
| thread_pool          | ACTIVE        |
| TP_THREAD_STATE      | ACTIVE        |
| TP_THREAD_GROUP_STATE | ACTIVE        |
| TP_THREAD_GROUP_STATS | ACTIVE        |
+----------------------+---------------+
```

If the server loads the thread pool plugin successfully, it sets the `thread_handling` system variable to `loaded-dynamically`.

If a plugin fails to initialize, check the server error log for diagnostic messages.

5.6.3.3 Thread Pool Operation

The thread pool consists of a number of thread groups, each of which manages a set of client connections. As connections are established, the thread pool assigns them to thread groups in round-robin fashion.

The thread pool exposes system variables that may be used to configure its operation:

- `thread_pool_algorithm`: The concurrency algorithm to use for scheduling.
- `thread_pool_dedicated_listeners`: Dedicates a listener thread in each thread group to listen for incoming statements from connections assigned to the group.
- `thread_pool_high_priority_connection`: How to schedule statement execution for a session.
- `thread_pool_max_active_query_threads`: How many active threads per group to permit.
- `thread_pool_max_transactions_limit`: The maximum number of transactions permitted by the thread pool plugin.
- `thread_pool_max_unused_threads`: How many sleeping threads to permit.
- `thread_pool_prio_kickup_timer`: How long before the thread pool moves a statement awaiting execution from the low-priority queue to the high-priority queue.
- `thread_pool_query_threads_per_group`: The number of query threads permitted in a thread group (the default is a single query thread). Consider increasing the value if you experience slower response times due to long-running transactions.
- `thread_pool_size`: The number of thread groups in the thread pool. This is the most important parameter controlling thread pool performance.
- `thread_pool_stall_limit`: The time before an executing statement is considered to be stalled.
- `thread_pool_transaction_delay`: The delay period before starting a new transaction.

To configure the number of thread groups, use the `thread_pool_size` system variable. The default number of groups is 16. For guidelines on setting this variable, see Section 5.6.3.4, “Thread Pool Tuning”.

The maximum number of threads per group is 4096 (or 4095 on some systems where one thread is used internally).

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. This differs from the default thread-handling model that associates one thread with one connection such that a given thread executes all statements from its connection.

By default, the thread pool tries to ensure a maximum of one thread executing in each group at any time, but sometimes permits more threads to execute temporarily for best performance:

- Each thread group has a listener thread that listens for incoming statements from the connections assigned to the group. When a statement arrives, the thread group either begins executing it immediately or queues it for later execution:
  - Immediate execution occurs if the statement is the only one received, and there are no statements queued or currently executing.

    From MySQL 8.0.31, immediate execution can be delayed by configuring `thread_pool_transaction_delay`, which has a throttling effect on transactions. For more information, refer to the description of this variable in the discussion that follows.
  - Queuing occurs if the statement cannot begin executing immediately due to concurrently queued or executing statements.
- The `thread_pool_transaction_delay` variable specifies a transaction delay in milliseconds. Worker threads sleep for the specified period before executing a new transaction.

  A transaction delay can be used in cases where parallel transactions affect the performance of other operations due to resource contention. For example, if parallel transactions affect index creation or an online buffer pool resizing operation, you can configure a transaction delay to reduce resource contention while those operations are running. The delay has a throttling effect on transactions.

  The `thread_pool_transaction_delay` setting does not affect queries issued from a privileged connection (a connection assigned to the `Admin` thread group). These queries are not subject to a configured transaction delay.
- If immediate execution occurs, the listener thread performs it. (This means that temporarily no thread in the group is listening.) If the statement finishes quickly, the executing thread returns to listening for statements. Otherwise, the thread pool considers the statement stalled and starts another thread as a listener thread (creating it if necessary). To ensure that no thread group becomes blocked by stalled statements, the thread pool has a background thread that regularly monitors thread group states.

  By using the listening thread to execute a statement that can begin immediately, there is no need to create an additional thread if the statement finishes quickly. This ensures the most efficient execution possible in the case of a low number of concurrent threads.

  When the thread pool plugin starts, it creates one thread per group (the listener thread), plus the background thread. Additional threads are created as necessary to execute statements.
- The value of the `thread_pool_stall_limit` system variable determines the meaning of “finishes quickly” in the previous item. The default time before threads are considered stalled is 60ms but can be set to a maximum of 6s. This parameter is configurable to enable you to strike a balance appropriate for the server work load. Short wait values permit threads to start more quickly. Short values are also better for avoiding deadlock situations. Long wait values are useful for workloads that

include long-running statements, to avoid starting too many new statements while the current ones execute.

- If `thread_pool_max_active_query_threads` is 0, the default algorithm applies as just described for determining the maximum number of active threads per group. The default algorithm takes stalled threads into account and may temporarily permit more active threads. If `thread_pool_max_active_query_threads` is greater than 0, it places a limit on the number of active threads per group.

- The thread pool focuses on limiting the number of concurrent short-running statements. Before an executing statement reaches the stall time, it prevents other statements from beginning to execute. If the statement executes past the stall time, it is permitted to continue but no longer prevents other statements from starting. In this way, the thread pool tries to ensure that in each thread group there is never more than one short-running statement, although there might be multiple long-running statements. It is undesirable to let long-running statements prevent other statements from executing because there is no limit on the amount of waiting that might be necessary. For example, on a replication source server, a thread that is sending binary log events to a replica effectively runs forever.

- A statement becomes blocked if it encounters a disk I/O operation or a user level lock (row lock or table lock). The block would cause the thread group to become unused, so there are callbacks to the thread pool to ensure that the thread pool can immediately start a new thread in this group to execute another statement. When a blocked thread returns, the thread pool permits it to restart immediately.

- There are two queues, a high-priority queue and a low-priority queue. The first statement in a transaction goes to the low-priority queue. Any following statements for the transaction go to the high-priority queue if the transaction is ongoing (statements for it have begun executing), or to the low-priority queue otherwise. Queue assignment can be affected by enabling the `thread_pool_high_priority_connection` system variable, which causes all queued statements for a session to go into the high-priority queue.

  Statements for a nontransactional storage engine, or a transactional engine if `autocommit` is enabled, are treated as low-priority statements because in this case each statement is a transaction. Thus, given a mix of statements for `InnoDB` and `MyISAM` tables, the thread pool prioritizes those for `InnoDB` over those for `MyISAM` unless `autocommit` is enabled. With `autocommit` enabled, all statements have low priority.

- When the thread group selects a queued statement for execution, it first looks in the high-priority queue, then in the low-priority queue. If a statement is found, it is removed from its queue and begins to execute.

- If a statement stays in the low-priority queue too long, the thread pool moves to the high-priority queue. The value of the `thread_pool_prio_kickup_timer` system variable controls the time before movement. For each thread group, a maximum of one statement per 10ms (100 per second) is moved from the low-priority queue to the high-priority queue.

- The thread pool reuses the most active threads to obtain a much better use of CPU caches. This is a small adjustment that has a great impact on performance.

- While a thread executes a statement from a user connection, Performance Schema instrumentation accounts thread activity to the user connection. Otherwise, Performance Schema accounts activity to the thread pool.

Here are examples of conditions under which a thread group might have multiple threads started to execute statements:

- One thread begins executing a statement, but runs long enough to be considered stalled. The thread group permits another thread to begin executing another statement even through the first thread is still executing.

- One thread begins executing a statement, then becomes blocked and reports this back to the thread pool. The thread group permits another thread to begin executing another statement.
- One thread begins executing a statement, becomes blocked, but does not report back that it is blocked because the block does not occur in code that has been instrumented with thread pool callbacks. In this case, the thread appears to the thread group to be still running. If the block lasts long enough for the statement to be considered stalled, the group permits another thread to begin executing another statement.

The thread pool is designed to be scalable across an increasing number of connections. It is also designed to avoid deadlocks that can arise from limiting the number of actively executing statements. It is important that threads that do not report back to the thread pool do not prevent other statements from executing and thus cause the thread pool to become deadlocked. Examples of such statements follow:

- Long-running statements. These would lead to all resources used by only a few statements and they could prevent all others from accessing the server.
- Binary log dump threads that read the binary log and send it to replicas. This is a kind of long-running “statement” that runs for a very long time, and that should not prevent other statements from executing.
- Statements blocked on a row lock, table lock, sleep, or any other blocking activity that has not been reported back to the thread pool by MySQL Server or a storage engine.

In each case, to prevent deadlock, the statement is moved to the stalled category when it does not complete quickly, so that the thread group can permit another statement to begin executing. With this design, when a thread executes or becomes blocked for an extended time, the thread pool moves the thread to the stalled category and for the rest of the statement's execution, it does not prevent other statements from executing.

The maximum number of threads that can occur is the sum of `max_connections` and `thread_pool_size`. This can happen in a situation where all connections are in execution mode and an extra thread is created per group to listen for more statements. This is not necessarily a state that happens often, but it is theoretically possible.

Privileged Connections

When the limit defined by `thread_pool_max_transactions_limit` has been reached, new connections appear to hang until one or more existing transactions are completed. The same occurs when attempting to start a new transaction on an existing connection. If existing connections are blocked or long-running, the only way to access the server is using a privileged connection.

To establish a privileged connection, the user initiating the connection must have the `TP_CONNECTION_ADMIN` privilege. A privileged connection ignores the limit defined by `thread_pool_max_transactions_limit` and permits connecting to the server to increase the limit, remove the limit, or kill running transactions. `TP_CONNECTION_ADMIN` privilege must be granted explicitly. It is not granted to any user by default.

A privileged connection can execute statements and start transactions, and is assigned to a thread group designated as the `Admin` thread group.

When querying the `performance_schema.tp_thread_group_stats` table, which reports statistics per thread group, `Admin` thread group statistics are reported in the last row of the result set. For example, if `SELECT * FROM performance_schema.tp_thread_group_stats\G` returns 17 rows (one row per thread group), the `Admin` thread group statistics are reported in the 17th row.

5.6.3.4 Thread Pool Tuning

This section provides guidelines on setting thread pool system variables for best performance, measured using a metric such as transactions per second.

`thread_pool_size` is the most important parameter controlling thread pool performance. It can be set only at server startup. Our experience in testing the thread pool indicates the following:

- If the primary storage engine is `InnoDB`, the optimal `thread_pool_size` setting is likely to be between 16 and 36, with the most common optimal values tending to be from 24 to 36. We have not seen any situation where the setting has been optimal beyond 36. There may be special cases where a value smaller than 16 is optimal.

  For workloads such as DBT2 and Sysbench, the optimum for `InnoDB` seems to be usually around 36. For very write-intensive workloads, the optimal setting can sometimes be lower.

- If the primary storage engine is `MyISAM`, the `thread_pool_size` setting should be fairly low. Optimal performance is often seen with values from 4 to 8. Higher values tend to have a slightly negative but not dramatic impact on performance.

Another system variable, `thread_pool_stall_limit`, is important for handling of blocked and long-running statements. If all calls that block the MySQL Server are reported to the thread pool, it would always know when execution threads are blocked. However, this may not always be true. For example, blocks could occur in code that has not been instrumented with thread pool callbacks. For such cases, the thread pool must be able to identify threads that appear to be blocked. This is done by means of a timeout that can be tuned using the `thread_pool_stall_limit` system variable, the value of which is measured in 10ms units. This parameter ensures that the server does not become completely blocked. The value of `thread_pool_stall_limit` has an upper limit of 6 seconds to prevent the risk of a deadlocked server.

`thread_pool_stall_limit` also enables the thread pool to handle long-running statements. If a long-running statement was permitted to block a thread group, all other connections assigned to the group would be blocked and unable to start execution until the long-running statement completed. In the worst case, this could take hours or even days.

The value of `thread_pool_stall_limit` should be chosen such that statements that execute longer than its value are considered stalled. Stalled statements generate a lot of extra overhead since they involve extra context switches and in some cases even extra thread creations. On the other hand, setting the `thread_pool_stall_limit` parameter too high means that long-running statements block a number of short-running statements for longer than necessary. Short wait values permit threads to start more quickly. Short values are also better for avoiding deadlock situations. Long wait values are useful for workloads that include long-running statements, to avoid starting too many new statements while the current ones execute.

Suppose a server executes a workload where 99.9% of the statements complete within 100ms even when the server is loaded, and the remaining statements take between 100ms and 2 hours fairly evenly spread. In this case, it would make sense to set `thread_pool_stall_limit` to 10 (10 × 10ms = 100ms). The default value of 6 (60ms) is suitable for servers that primarily execute very simple statements.

The `thread_pool_stall_limit` parameter can be changed at runtime to enable you to strike a balance appropriate for the server work load. Assuming that the `tp_thread_group_stats` table is enabled, you can use the following query to determine the fraction of executed statements that stalled:

```
SELECT SUM(STALLED_QUERIES_EXECUTED) / SUM(QUERIES_EXECUTED)
FROM performance_schema.tp_thread_group_stats;
```

This number should be as low as possible. To decrease the likelihood of statements stalling, increase the value of `thread_pool_stall_limit`.

When a statement arrives, what is the maximum time it can be delayed before it actually starts executing? Suppose that the following conditions apply:

- There are 200 statements queued in the low-priority queue.
- There are 10 statements queued in the high-priority queue.

- `thread_pool_prio_kickup_timer` is set to 10000 (10 seconds).
- `thread_pool_stall_limit` is set to 100 (1 second).

In the worst case, the 10 high-priority statements represent 10 transactions that continue executing for a long time. Thus, in the worst case, no statements can be moved to the high-priority queue because it always already contains statements awaiting execution. After 10 seconds, the new statement is eligible to be moved to the high-priority queue. However, before it can be moved, all the statements before it must be moved as well. This could take another 2 seconds because a maximum of 100 statements per second are moved to the high-priority queue. Now when the statement reaches the high-priority queue, there could potentially be many long-running statements ahead of it. In the worst case, every one of those becomes stalled and 1 second is required for each statement before the next statement is retrieved from the high-priority queue. Thus, in this scenario, it takes 222 seconds before the new statement starts executing.

This example shows a worst case for an application. How to handle it depends on the application. If the application has high requirements for the response time, it should most likely throttle users at a higher level itself. Otherwise, it can use the thread pool configuration parameters to set some kind of a maximum waiting time.

5.6.4 The Rewriter Query Rewrite Plugin

MySQL supports query rewrite plugins that can examine and possibly modify SQL statements received by the server before the server executes them. See Query Rewrite Plugins.

MySQL distributions include a postparse query rewrite plugin named `Rewriter` and scripts for installing the plugin and its associated elements. These elements work together to provide statement-rewriting capability:

- A server-side plugin named `Rewriter` examines statements and may rewrite them, based on its in-memory cache of rewrite rules.
- These statements are subject to rewriting:
  - As of MySQL 8.0.12: `SELECT`, `INSERT`, `REPLACE`, `UPDATE`, and `DELETE`.
  - Prior to MySQL 8.0.12: `SELECT` only.

  Standalone statements and prepared statements are subject to rewriting. Statements occurring within view definitions or stored programs are not subject to rewriting.
- The `Rewriter` plugin uses a database named `query_rewrite` containing a table named `rewrite_rules`. The table provides persistent storage for the rules that the plugin uses to decide whether to rewrite statements. Users communicate with the plugin by modifying the set of rules stored in this table. The plugin communicates with users by setting the `message` column of table rows.
- The `query_rewrite` database contains a stored procedure named `flush_rewrite_rules()` that loads the contents of the rules table into the plugin.
- A loadable function named `load_rewrite_rules()` is used by the `flush_rewrite_rules()` stored procedure.
- The `Rewriter` plugin exposes system variables that enable plugin configuration and status variables that provide runtime operational information. In MySQL 8.0.31 and later, this plugin also supports a privilege (`SKIP_QUERY_REWRITE`) that protects a given user's queries from being rewritten.

The following sections describe how to install and use the `Rewriter` plugin, and provide reference information for its associated elements.

5.6.4.1 Installing or Uninstalling the Rewriter Query Rewrite Plugin

Note

If installed, the `Rewriter` plugin involves some overhead even when disabled. To avoid this overhead, do not install the plugin unless you plan to use it.

To install or uninstall the `Rewriter` query rewrite plugin, choose the appropriate script located in the `share` directory of your MySQL installation:

- `install_rewriter.sql`: Choose this script to install the `Rewriter` plugin and its associated elements.
- `uninstall_rewriter.sql`: Choose this script to uninstall the `Rewriter` plugin and its associated elements.

Run the chosen script as follows:

```
$> mysql -u root -p < install_rewriter.sql
Enter password: (enter root password here)
```

The example here uses the `install_rewriter.sql` installation script. Substitute `uninstall_rewriter.sql` if you are uninstalling the plugin.

Running an installation script should install and enable the plugin. To verify that, connect to the server and execute this statement:

```
mysql> SHOW GLOBAL VARIABLES LIKE 'rewriter_enabled';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| rewriter_enabled | ON    |
+------------------+-------+
```

For usage instructions, see Section 5.6.4.2, “Using the Rewriter Query Rewrite Plugin”. For reference information, see Section 5.6.4.3, “Rewriter Query Rewrite Plugin Reference”.

5.6.4.2 Using the Rewriter Query Rewrite Plugin

To enable or disable the plugin, enable or disable the `rewriter_enabled` system variable. By default, the `Rewriter` plugin is enabled when you install it (see Section 5.6.4.1, “Installing or Uninstalling the Rewriter Query Rewrite Plugin”). To set the initial plugin state explicitly, you can set the variable at server startup. For example, to enable the plugin in an option file, use these lines:

```
[mysqld]
rewriter_enabled=ON
```

It is also possible to enable or disable the plugin at runtime:

```
SET GLOBAL rewriter_enabled = ON;
SET GLOBAL rewriter_enabled = OFF;
```

Assuming that the `Rewriter` plugin is enabled, it examines and possibly modifies each rewritable statement received by the server. The plugin determines whether to rewrite statements based on its in-memory cache of rewriting rules, which are loaded from the `rewrite_rules` table in the `query_rewrite` database.

These statements are subject to rewriting:

- As of MySQL 8.0.12: `SELECT`, `INSERT`, `REPLACE`, `UPDATE`, and `DELETE`.
- Prior to MySQL 8.0.12: `SELECT` only.

Standalone statements and prepared statements are subject to rewriting. Statements occurring within view definitions or stored programs are not subject to rewriting.

Beginning with MySQL 8.0.31, statements run by users with the `SKIP_QUERY_REWRITE` privilege are not subject to rewriting, provided that the `rewriter_enabled_for_threads_without_privilege_checks` system variable is set to `OFF` (default `ON`). This can be used for control statements and statements that should be replicated unchanged, such as those from the `SOURCE_USER` specified by `CHANGE REPLICATION SOURCE TO`. This is also true for statements executed by MySQL client programs including `mysqlbinlog`, `mysqladmin`, `mysqldump`, and `mysqlpump`; for this reason, you should grant `SKIP_QUERY_REWRITE` to the user account or accounts used by these utilities to connect to MySQL.

- Adding Rewrite Rules
- How Statement Matching Works
- Rewriting Prepared Statements
- Rewriter Plugin Operational Information
- Rewriter Plugin Use of Character Sets

Adding Rewrite Rules

To add rules for the `Rewriter` plugin, add rows to the `rewrite_rules` table, then invoke the `flush_rewrite_rules()` stored procedure to load the rules from the table into the plugin. The following example creates a simple rule to match statements that select a single literal value:

```
INSERT INTO query_rewrite.rewrite_rules (pattern, replacement)
VALUES('SELECT ?', 'SELECT ? + 1');
```

The resulting table contents look like this:

```
mysql> SELECT * FROM query_rewrite.rewrite_rules\G
*************************** 1. row ***************************
                id: 1
           pattern: SELECT ?
  pattern_database: NULL
       replacement: SELECT ? + 1
           enabled: YES
           message: NULL
    pattern_digest: NULL
normalized_pattern: NULL
```

The rule specifies a pattern template indicating which `SELECT` statements to match, and a replacement template indicating how to rewrite matching statements. However, adding the rule to the `rewrite_rules` table is not sufficient to cause the `Rewriter` plugin to use the rule. You must invoke `flush_rewrite_rules()` to load the table contents into the plugin in-memory cache:

```
mysql> CALL query_rewrite.flush_rewrite_rules();
```

Tip

If your rewrite rules seem not to be working properly, make sure that you have reloaded the rules table by calling `flush_rewrite_rules()`.

When the plugin reads each rule from the rules table, it computes a normalized (statement digest) form from the pattern and a digest hash value, and uses them to update the `normalized_pattern` and `pattern_digest` columns:

```
mysql> SELECT * FROM query_rewrite.rewrite_rules\G
*************************** 1. row ***************************
                id: 1
           pattern: SELECT ?
```

```
  pattern_database: NULL
       replacement: SELECT ? + 1
           enabled: YES
           message: NULL
    pattern_digest: d1b44b0c19af710b5a679907e284acd2ddc285201794bc69a2389d77baedddae
normalized_pattern: select ?
```

For information about statement digesting, normalized statements, and digest hash values, see Section 27.10, “Performance Schema Statement Digests and Sampling”.

If a rule cannot be loaded due to some error, calling `flush_rewrite_rules()` produces an error:

```
mysql> CALL query_rewrite.flush_rewrite_rules();
ERROR 1644 (45000): Loading of some rule(s) failed.
```

When this occurs, the plugin writes an error message to the `message` column of the rule row to communicate the problem. Check the `rewrite_rules` table for rows with non-`NULL` `message` column values to see what problems exist.

Patterns use the same syntax as prepared statements (see Section 13.5.1, “PREPARE Statement”). Within a pattern template, `?` characters act as parameter markers that match data values. The `?` characters should not be enclosed within quotation marks. Parameter markers can be used only where data values should appear, and they cannot be used for SQL keywords, identifiers, functions, and so on. The plugin parses a statement to identify the literal values (as defined in Section 9.1, “Literal Values”), so you can put a parameter marker in place of any literal value.

Like the pattern, the replacement can contain `?` characters. For a statement that matches a pattern template, the plugin rewrites it, replacing `?` parameter markers in the replacement using data values matched by the corresponding markers in the pattern. The result is a complete statement string. The plugin asks the server to parse it, and returns the result to the server as the representation of the rewritten statement.

After adding and loading the rule, check whether rewriting occurs according to whether statements match the rule pattern:

```
mysql> SELECT PI();
+----------+
| PI()     |
+----------+
| 3.141593 |
+----------+
1 row in set (0.01 sec)

mysql> SELECT 10;
+--------+
| 10 + 1 |
+--------+
|     11 |
+--------+
1 row in set, 1 warning (0.00 sec)
```

No rewriting occurs for the first `SELECT` statement, but does for the second. The second statement illustrates that when the `Rewriter` plugin rewrites a statement, it produces a warning message. To view the message, use `SHOW WARNINGS`:

```
mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Note
   Code: 1105
Message: Query 'SELECT 10' rewritten to 'SELECT 10 + 1' by a query rewrite plugin
```

A statement need not be rewritten to a statement of the same type. The following example loads a rule that rewrites `DELETE` statements to `UPDATE` statements:

```
INSERT INTO query_rewrite.rewrite_rules (pattern, replacement)
```

```
VALUES('DELETE FROM db1.t1 WHERE col = ?',
       'UPDATE db1.t1 SET col = NULL WHERE col = ?');
CALL query_rewrite.flush_rewrite_rules();
```

To enable or disable an existing rule, modify its `enabled` column and reload the table into the plugin. To disable rule 1:

```
UPDATE query_rewrite.rewrite_rules SET enabled = 'NO' WHERE id = 1;
CALL query_rewrite.flush_rewrite_rules();
```

This enables you to deactivate a rule without removing it from the table.

To re-enable rule 1:

```
UPDATE query_rewrite.rewrite_rules SET enabled = 'YES' WHERE id = 1;
CALL query_rewrite.flush_rewrite_rules();
```

The `rewrite_rules` table contains a `pattern_database` column that `Rewriter` uses for matching table names that are not qualified with a database name:

- Qualified table names in statements match qualified names in the pattern if corresponding database and table names are identical.
- Unqualified table names in statements match unqualified names in the pattern only if the default database is the same as `pattern_database` and the table names are identical.

Suppose that a table named `appdb.users` has a column named `id` and that applications are expected to select rows from the table using a query of one of these forms, where the second can be used when `appdb` is the default database:

```
SELECT * FROM users WHERE appdb.id = id_value;
SELECT * FROM users WHERE id = id_value;
```

Suppose also that the `id` column is renamed to `user_id` (perhaps the table must be modified to add another type of ID and it is necessary to indicate more specifically what type of ID the `id` column represents).

The change means that applications must refer to `user_id` rather than `id` in the `WHERE` clause, but old applications that cannot be updated no longer work properly. The `Rewriter` plugin can solve this problem by matching and rewriting problematic statements. To match the statement `SELECT * FROM appdb.users WHERE id = value` and rewrite it as `SELECT * FROM appdb.users WHERE user_id = value`, you can insert a row representing a replacement rule into the rewrite rules table. If you also want to match this `SELECT` using the unqualified table name, it is also necessary to add an explicit rule. Using `?` as a value placeholder, the two `INSERT` statements needed look like this:

```
INSERT INTO query_rewrite.rewrite_rules
    (pattern, replacement) VALUES(
    'SELECT * FROM appdb.users WHERE id = ?',
    'SELECT * FROM appdb.users WHERE user_id = ?'
    );
INSERT INTO query_rewrite.rewrite_rules
    (pattern, replacement, pattern_database) VALUES(
    'SELECT * FROM users WHERE id = ?',
    'SELECT * FROM users WHERE user_id = ?',
    'appdb'
    );
```

After adding the two new rules, execute the following statement to cause them to take effect:

```
CALL query_rewrite.flush_rewrite_rules();
```

`Rewriter` uses the first rule to match statements that use the qualified table name, and the second to match statements that use the unqualified name. The second rule works only when `appdb` is the default database.

How Statement Matching Works

The `Rewriter` plugin uses statement digests and digest hash values to match incoming statements against rewrite rules in stages. The `max_digest_length` system variable determines the size of the buffer used for computing statement digests. Larger values enable computation of digests that distinguish longer statements. Smaller values use less memory but increase the likelihood of longer statements colliding with the same digest value.

The plugin matches each statement to the rewrite rules as follows:

1. Compute the statement digest hash value and compare it to the rule digest hash values. This is subject to false positives, but serves as a quick rejection test.
2. If the statement digest hash value matches any pattern digest hash values, match the normalized (statement digest) form of the statement to the normalized form of the matching rule patterns.
3. If the normalized statement matches a rule, compare the literal values in the statement and the pattern. A `?` character in the pattern matches any literal value in the statement. If the statement prepares a statement, `?` in the pattern also matches `?` in the statement. Otherwise, corresponding literals must be the same.

If multiple rules match a statement, it is nondeterministic which one the plugin uses to rewrite the statement.

If a pattern contains more markers than the replacement, the plugin discards excess data values. If a pattern contains fewer markers than the replacement, it is an error. The plugin notices this when the rules table is loaded, writes an error message to the `message` column of the rule row to communicate the problem, and sets the `Rewriter_reload_error` status variable to `ON`.

Rewriting Prepared Statements

Prepared statements are rewritten at parse time (that is, when they are prepared), not when they are executed later.

Prepared statements differ from nonprepared statements in that they may contain `?` characters as parameter markers. To match a `?` in a prepared statement, a `Rewriter` pattern must contain `?` in the same location. Suppose that a rewrite rule has this pattern:

```
SELECT ?, 3
```

The following table shows several prepared `SELECT` statements and whether the rule pattern matches them.

Prepared Statement	Whether Pattern Matches Statement
`PREPARE s AS 'SELECT 3, 3'`	Yes
`PREPARE s AS 'SELECT ?, 3'`	Yes
`PREPARE s AS 'SELECT 3, ?'`	No
`PREPARE s AS 'SELECT ?, ?'`	No

Rewriter Plugin Operational Information

The `Rewriter` plugin makes information available about its operation by means of several status variables:

```
mysql> SHOW GLOBAL STATUS LIKE 'Rewriter%';
+----------------------------------+-------+
| Variable_name                    | Value |
+----------------------------------+-------+
```

```
| Rewriter_number_loaded_rules      | 1     |
| Rewriter_number_reloads           | 5     |
| Rewriter_number_rewritten_queries | 1     |
| Rewriter_reload_error             | ON    |
+-----------------------------------+-------+
```

For descriptions of these variables, see Rewriter Query Rewrite Plugin Status Variables.

When you load the rules table by calling the `flush_rewrite_rules()` stored procedure, if an error occurs for some rule, the `CALL` statement produces an error, and the plugin sets the `Rewriter_reload_error` status variable to `ON`:

```
mysql> CALL query_rewrite.flush_rewrite_rules();
ERROR 1644 (45000): Loading of some rule(s) failed.

mysql> SHOW GLOBAL STATUS LIKE 'Rewriter_reload_error';
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| Rewriter_reload_error | ON    |
+----------------------+-------+
```

In this case, check the `rewrite_rules` table for rows with non-`NULL` `message` column values to see what problems exist.

Rewriter Plugin Use of Character Sets

When the `rewrite_rules` table is loaded into the `Rewriter` plugin, the plugin interprets statements using the current global value of the `character_set_client` system variable. If the global `character_set_client` value is changed subsequently, the rules table must be reloaded.

A client must have a session `character_set_client` value identical to what the global value was when the rules table was loaded or rule matching does not work for that client.

5.6.4.3 Rewriter Query Rewrite Plugin Reference

The following discussion serves as a reference to these elements associated with the `Rewriter` query rewrite plugin:

- The `Rewriter` rules table in the `query_rewrite` database
- `Rewriter` procedures and functions
- `Rewriter` system and status variables

Rewriter Query Rewrite Plugin Rules Table

The `rewrite_rules` table in the `query_rewrite` database provides persistent storage for the rules that the `Rewriter` plugin uses to decide whether to rewrite statements.

Users communicate with the plugin by modifying the set of rules stored in this table. The plugin communicates information to users by setting the table's `message` column.

Note

The rules table is loaded into the plugin by the `flush_rewrite_rules` stored procedure. Unless that procedure has been called following the most recent table modification, the table contents do not necessarily correspond to the set of rules the plugin is using.

The `rewrite_rules` table has these columns:

- `id`

The rule ID. This column is the table primary key. You can use the ID to uniquely identify any rule.

- `pattern`

  The template that indicates the pattern for statements that the rule matches. Use `?` to represent parameter markers that match data values.

- `pattern_database`

  The database used to match unqualified table names in statements. Qualified table names in statements match qualified names in the pattern if corresponding database and table names are identical. Unqualified table names in statements match unqualified names in the pattern only if the default database is the same as `pattern_database` and the table names are identical.

- `replacement`

  The template that indicates how to rewrite statements matching the `pattern` column value. Use `?` to represent parameter markers that match data values. In rewritten statements, the plugin replaces `?` parameter markers in `replacement` using data values matched by the corresponding markers in `pattern`.

- `enabled`

  Whether the rule is enabled. Load operations (performed by invoking the `flush_rewrite_rules()` stored procedure) load the rule from the table into the `Rewriter` in-memory cache only if this column is `YES`.

  This column makes it possible to deactivate a rule without removing it: Set the column to a value other than `YES` and reload the table into the plugin.

- `message`

  The plugin uses this column for communicating with users. If no error occurs when the rules table is loaded into memory, the plugin sets the `message` column to `NULL`. A non-`NULL` value indicates an error and the column contents are the error message. Errors can occur under these circumstances:

  - Either the pattern or the replacement is an incorrect SQL statement that produces syntax errors.
  - The replacement contains more `?` parameter markers than the pattern.

  If a load error occurs, the plugin also sets the `Rewriter_reload_error` status variable to `ON`.

- `pattern_digest`

  This column is used for debugging and diagnostics. If the column exists when the rules table is loaded into memory, the plugin updates it with the pattern digest. This column may be useful if you are trying to determine why some statement fails to be rewritten.

- `normalized_pattern`

  This column is used for debugging and diagnostics. If the column exists when the rules table is loaded into memory, the plugin updates it with the normalized form of the pattern. This column may be useful if you are trying to determine why some statement fails to be rewritten.

Rewriter Query Rewrite Plugin Procedures and Functions

`Rewriter` plugin operation uses a stored procedure that loads the rules table into its in-memory cache, and a helper loadable function. Under normal operation, users invoke only the stored procedure. The function is intended to be invoked by the stored procedure, not directly by users.

- `flush_rewrite_rules()`

This stored procedure uses the `load_rewrite_rules()` function to load the contents of the `rewrite_rules` table into the `Rewriter` in-memory cache.

Calling `flush_rewrite_rules()` implies `COMMIT`.

Invoke this procedure after you modify the rules table to cause the plugin to update its cache from the new table contents. If any errors occur, the plugin sets the `message` column for the appropriate rule rows in the table and sets the `Rewriter_reload_error` status variable to `ON`.

- `load_rewrite_rules()`

  This function is a helper routine used by the `flush_rewrite_rules()` stored procedure.

Rewriter Query Rewrite Plugin System Variables

The `Rewriter` query rewrite plugin supports the following system variables. These variables are available only if the plugin is installed (see Section 5.6.4.1, “Installing or Uninstalling the Rewriter Query Rewrite Plugin”).

- `rewriter_enabled`

System Variable	`rewriter_enabled`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Boolean
Default Value	`ON`
Valid Values	`OFF`

Whether the `Rewriter` query rewrite plugin is enabled.

- `rewriter_enabled_for_threads_without_privilege_checks`

Introduced	8.0.31
System Variable	`rewriter_enabled_for_threads_without_privil`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Boolean
Default Value	`ON`
Valid Values	`OFF`

Whether to apply rewrites for replication threads which execute with privilege checks disabled. If set to `OFF`, such rewrites are skipped. Requires the `SYSTEM_VARIABLES_ADMIN` privilege or `SUPER` privilege to set.

This variable has no effect if `rewriter_enabled` is `OFF`.

- `rewriter_verbose`

System Variable	`rewriter_verbose`
Scope	Global
Dynamic	Yes

SET_VAR Hint Applies	No
Type	Integer

For internal use.

Rewriter Query Rewrite Plugin Status Variables

The Rewriter query rewrite plugin supports the following status variables. These variables are available only if the plugin is installed (see Section 5.6.4.1, “Installing or Uninstalling the Rewriter Query Rewrite Plugin”).

- Rewriter_number_loaded_rules

  The number of rewrite plugin rewrite rules successfully loaded from the rewrite_rules table into memory for use by the Rewriter plugin.

- Rewriter_number_reloads

  The number of times the rewrite_rules table has been loaded into the in-memory cache used by the Rewriter plugin.

- Rewriter_number_rewritten_queries

  The number of queries rewritten by the Rewriter query rewrite plugin since it was loaded.

- Rewriter_reload_error

  Whether an error occurred the most recent time that the rewrite_rules table was loaded into the in-memory cache used by the Rewriter plugin. If the value is OFF, no error occurred. If the value is ON, an error occurred; check the message column of the rewriter_rules table for error messages.

5.6.5 The ddl_rewriter Plugin

MySQL 8.0.16 and higher includes a ddl_rewriter plugin that modifies CREATE TABLE statements received by the server before it parses and executes them. The plugin removes ENCRYPTION, DATA DIRECTORY, and INDEX DIRECTORY clauses, which may be helpful when restoring tables from SQL dump files created from databases that are encrypted or that have their tables stored outside the data directory. For example, the plugin may enable restoring such dump files into an unencrypted instance or in an environment where the paths outside the data directory are not accessible.

Before using the ddl_rewriter plugin, install it according to the instructions provided in Section 5.6.5.1, “Installing or Uninstalling ddl_rewriter”.

ddl_rewriter examines SQL statements received by the server prior to parsing, rewriting them according to these conditions:

- ddl_rewriter considers only CREATE TABLE statements, and only if they are standalone statements that occur at the beginning of an input line or at the beginning of prepared statement text. ddl_rewriter does not consider CREATE TABLE statements within stored program definitions. Statements can extend over multiple lines.
- Within statements considered for rewrite, instances of the following clauses are rewritten and each instance replaced by a single space:
  - ENCRYPTION
  - DATA DIRECTORY (at the table and partition levels)
  - INDEX DIRECTORY (at the table and partition levels)

- Rewriting does not depend on lettercase.

If `ddl_rewriter` rewrites a statement, it generates a warning:

```
mysql> CREATE TABLE t (i INT) DATA DIRECTORY '/var/mysql/data';
Query OK, 0 rows affected, 1 warning (0.03 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Note
   Code: 1105
Message: Query 'CREATE TABLE t (i INT) DATA DIRECTORY '/var/mysql/data''
         rewritten to 'CREATE TABLE t (i INT) ' by a query rewrite plugin
1 row in set (0.00 sec)
```

If the general query log or binary log is enabled, the server writes to it statements as they appear after any rewriting by `ddl_rewriter`.

When installed, `ddl_rewriter` exposes the Performance Schema `memory/rewriter/ddl_rewriter` instrument for tracking plugin memory use. See Section 27.12.20.10, “Memory Summary Tables”

5.6.5.1 Installing or Uninstalling ddl_rewriter

This section describes how to install or uninstall the `ddl_rewriter` plugin. For general information about installing plugins, see Section 5.6.1, “Installing and Uninstalling Plugins”.

Note

If installed, the `ddl_rewriter` plugin involves some minimal overhead even when disabled. To avoid this overhead, install `ddl_rewriter` only for the period during which you intend to use it.

The primary use case is modification of statements restored from dump files, so the typical usage pattern is: 1) Install the plugin; 2) restore the dump file or files; 3) uninstall the plugin.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The plugin library file base name is `ddl_rewriter`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To install the `ddl_rewriter` plugin, use the `INSTALL PLUGIN` statement, adjusting the `.so` suffix for your platform as necessary:

```
INSTALL PLUGIN ddl_rewriter SONAME 'ddl_rewriter.so';
```

To verify plugin installation, examine the Information Schema `PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, “Obtaining Server Plugin Information”). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS, PLUGIN_TYPE
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE 'ddl%';
+--------------+---------------+-------------+
| PLUGIN_NAME  | PLUGIN_STATUS | PLUGIN_TYPE |
+--------------+---------------+-------------+
| ddl_rewriter | ACTIVE        | AUDIT       |
+--------------+---------------+-------------+
```

As the preceding result shows, `ddl_rewriter` is implemented as an audit plugin.

If the plugin fails to initialize, check the server error log for diagnostic messages.

Once installed as just described, `ddl_rewriter` remains installed until uninstalled. To remove it, use `UNINSTALL PLUGIN`:

```
UNINSTALL PLUGIN ddl_rewriter;
```

If `ddl_rewriter` is installed, you can use the `--ddl-rewriter` option for subsequent server startups to control `ddl_rewriter` plugin activation. For example, to prevent the plugin from being enabled at runtime, use this option:

```
[mysqld]
ddl-rewriter=OFF
```

5.6.5.2 ddl_rewriter Plugin Options

This section describes the command options that control operation of the `ddl_rewriter` plugin. If values specified at startup time are incorrect, the `ddl_rewriter` plugin may fail to initialize properly and the server does not load it.

To control activation of the `ddl_rewriter` plugin, use this option:

- `--ddl-rewriter[=value]`

Command-Line Format	`--ddl-rewriter[=value]`
Introduced	8.0.16
Type	Enumeration
Default Value	`ON`

Valid Values	`ON`
	`OFF`
	`FORCE`
	`FORCE_PLUS_PERMANENT`

This option controls how the server loads the `ddl_rewriter` plugin at startup. It is available only if the plugin has been previously registered with `INSTALL PLUGIN` or is loaded with `--plugin-load` or `--plugin-load-add`. See Section 5.6.5.1, “Installing or Uninstalling ddl_rewriter”.

The option value should be one of those available for plugin-loading options, as described in Section 5.6.1, “Installing and Uninstalling Plugins”. For example, `--ddl-rewriter=OFF` disables the plugin at server startup.

5.6.6 Version Tokens

MySQL includes Version Tokens, a feature that enables creation of and synchronization around server tokens that applications can use to prevent accessing incorrect or out-of-date data.

The Version Tokens interface has these characteristics:

- Version tokens are pairs consisting of a name that serves as a key or identifier, plus a value.
- Version tokens can be locked. An application can use token locks to indicate to other cooperating applications that tokens are in use and should not be modified.
- Version token lists are established per server (for example, to specify the server assignment or operational state). In addition, an application that communicates with a server can register its own list of tokens that indicate the state it requires the server to be in. An SQL statement sent by the application to a server not in the required state produces an error. This is a signal to the application that it should seek a different server in the required state to receive the SQL statement.

The following sections describe the elements of Version Tokens, discuss how to install and use it, and provide reference information for its elements.

5.6.6.1 Version Tokens Elements

Version Tokens is based on a plugin library that implements these elements:

- A server-side plugin named `version_tokens` holds the list of version tokens associated with the server and subscribes to notifications for statement execution events. The `version_tokens` plugin uses the audit plugin API to monitor incoming statements from clients and matches each client's session-specific version token list against the server version token list. If there is a match, the plugin lets the statement through and the server continues to process it. Otherwise, the plugin returns an error to the client and the statement fails.
- A set of loadable functions provides an SQL-level API for manipulating and inspecting the list of server version tokens maintained by the plugin. The `VERSION_TOKEN_ADMIN` privilege (or the deprecated `SUPER` privilege) is required to call any of the Version Token functions.
- When the `version_tokens` plugin loads, it defines the `VERSION_TOKEN_ADMIN` dynamic privilege. This privilege can be granted to users of the functions.
- A system variable enables clients to specify the list of version tokens that register the required server state. If the server has a different state when a client sends a statement, the client receives an error.

5.6.6.2 Installing or Uninstalling Version Tokens

Note

If installed, Version Tokens involves some overhead. To avoid this overhead, do not install it unless you plan to use it.

This section describes how to install or uninstall Version Tokens, which is implemented in a plugin library file containing a plugin and loadable functions. For general information about installing or uninstalling plugins and loadable functions, see Section 5.6.1, “Installing and Uninstalling Plugins”, and Section 5.7.1, “Installing and Uninstalling Loadable Functions”.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The plugin library file base name is `version_tokens`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To install the Version Tokens plugin and functions, use the `INSTALL PLUGIN` and `CREATE FUNCTION` statements, adjusting the `.so` suffix for your platform as necessary:

```
INSTALL PLUGIN version_tokens SONAME 'version_token.so';
CREATE FUNCTION version_tokens_set RETURNS STRING
  SONAME 'version_token.so';
CREATE FUNCTION version_tokens_show RETURNS STRING
  SONAME 'version_token.so';
CREATE FUNCTION version_tokens_edit RETURNS STRING
  SONAME 'version_token.so';
CREATE FUNCTION version_tokens_delete RETURNS STRING
  SONAME 'version_token.so';
CREATE FUNCTION version_tokens_lock_shared RETURNS INT
  SONAME 'version_token.so';
CREATE FUNCTION version_tokens_lock_exclusive RETURNS INT
  SONAME 'version_token.so';
CREATE FUNCTION version_tokens_unlock RETURNS INT
  SONAME 'version_token.so';
```

You must install the functions to manage the server's version token list, but you must also install the plugin because the functions do not work correctly without it.

If the plugin and functions are used on a replication source server, install them on all replica servers as well to avoid replication problems.

Once installed as just described, the plugin and functions remain installed until uninstalled. To remove them, use the `UNINSTALL PLUGIN` and `DROP FUNCTION` statements:

```
UNINSTALL PLUGIN version_tokens;
DROP FUNCTION version_tokens_set;
DROP FUNCTION version_tokens_show;
DROP FUNCTION version_tokens_edit;
DROP FUNCTION version_tokens_delete;
DROP FUNCTION version_tokens_lock_shared;
DROP FUNCTION version_tokens_lock_exclusive;
DROP FUNCTION version_tokens_unlock;
```

5.6.6.3 Using Version Tokens

Before using Version Tokens, install it according to the instructions provided at Section 5.6.6.2, “Installing or Uninstalling Version Tokens”.

A scenario in which Version Tokens can be useful is a system that accesses a collection of MySQL servers but needs to manage them for load balancing purposes by monitoring them and adjusting server assignments according to load changes. Such a system comprises these elements:

- The collection of MySQL servers to be managed.
- An administrative or management application that communicates with the servers and organizes them into high-availability groups. Groups serve different purposes, and servers within each group may have different assignments. Assignment of a server within a certain group can change at any time.
- Client applications that access the servers to retrieve and update data, choosing servers according to the purposes assigned them. For example, a client should not send an update to a read-only server.

Version Tokens permit server access to be managed according to assignment without requiring clients to repeatedly query the servers about their assignments:

- The management application performs server assignments and establishes version tokens on each server to reflect its assignment. The application caches this information to provide a central access point to it.

  If at some point the management application needs to change a server assignment (for example, to change it from permitting writes to read only), it changes the server's version token list and updates its cache.

- To improve performance, client applications obtain cache information from the management application, enabling them to avoid having to retrieve information about server assignments for each statement. Based on the type of statements it issues (for example, reads versus writes), a client selects an appropriate server and connects to it.

- In addition, the client sends to the server its own client-specific version tokens to register the assignment it requires of the server. For each statement sent by the client to the server, the server compares its own token list with the client token list. If the server token list contains all tokens present in the client token list with the same values, there is a match and the server executes the statement.

  On the other hand, perhaps the management application has changed the server assignment and its version token list. In this case, the new server assignment may now be incompatible with the client requirements. A token mismatch between the server and client token lists occurs and the server returns an error in reply to the statement. This is an indication to the client to refresh its version token information from the management application cache, and to select a new server to communicate with.

The client-side logic for detecting version token errors and selecting a new server can be implemented different ways:

- The client can handle all version token registration, mismatch detection, and connection switching itself.
- The logic for those actions can be implemented in a connector that manages connections between clients and MySQL servers. Such a connector might handle mismatch error detection and statement resending itself, or it might pass the error to the application and leave it to the application to resend the statement.

The following example illustrates the preceding discussion in more concrete form.

When Version Tokens initializes on a given server, the server's version token list is empty. Token list maintenance is performed by calling functions. The `VERSION_TOKEN_ADMIN` privilege (or the deprecated `SUPER` privilege) is required to call any of the Version Token functions, so token list modification is expected to be done by a management or administrative application that has that privilege.

Suppose that a management application communicates with a set of servers that are queried by clients to access employee and product databases (named `emp` and `prod`, respectively). All servers are permitted to process data retrieval statements, but only some of them are permitted to make database updates. To handle this on a database-specific basis, the management application establishes a list of version tokens on each server. In the token list for a given server, token names represent database names and token values are `read` or `write` depending on whether the database must be used in read-only fashion or whether it can take reads and writes.

Client applications register a list of version tokens they require the server to match by setting a system variable. Variable setting occurs on a client-specific basis, so different clients can register different requirements. By default, the client token list is empty, which matches any server token list. When a client sets its token list to a nonempty value, matching may succeed or fail, depending on the server version token list.

To define the version token list for a server, the management application calls the `version_tokens_set()` function. (There are also functions for modifying and displaying the token list, described later.) For example, the application might send these statements to a group of three servers:

Server 1:

```
mysql> SELECT version_tokens_set('emp=read;prod=read');
+-----------------------------------------+
| version_tokens_set('emp=read;prod=read') |
+-----------------------------------------+
| 2 version tokens set.                   |
+-----------------------------------------+
```

Server 2:

```
mysql> SELECT version_tokens_set('emp=write;prod=read');
+-----------------------------------------+
| version_tokens_set('emp=write;prod=read') |
+-----------------------------------------+
| 2 version tokens set.                   |
+-----------------------------------------+
```

Server 3:

```
mysql> SELECT version_tokens_set('emp=read;prod=write');
+-----------------------------------------+
| version_tokens_set('emp=read;prod=write') |
+-----------------------------------------+
| 2 version tokens set.                   |
```

```
+-----------------------------------------+
```

The token list in each case is specified as a semicolon-separated list of *`name=value`* pairs. The resulting token list values result in these server assignments:

- Any server accepts reads for either database.
- Only server 2 accepts updates for the `emp` database.
- Only server 3 accepts updates for the `prod` database.

In addition to assigning each server a version token list, the management application also maintains a cache that reflects the server assignments.

Before communicating with the servers, a client application contacts the management application and retrieves information about server assignments. Then the client selects a server based on those assignments. Suppose that a client wants to perform both reads and writes on the `emp` database. Based on the preceding assignments, only server 2 qualifies. The client connects to server 2 and registers its server requirements there by setting its `version_tokens_session` system variable:

```
mysql> SET @@SESSION.version_tokens_session = 'emp=write';
```

For subsequent statements sent by the client to server 2, the server compares its own version token list to the client list to check whether they match. If so, statements execute normally:

```
mysql> UPDATE emp.employee SET salary = salary * 1.1 WHERE id = 4981;
Query OK, 1 row affected (0.07 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT last_name, first_name FROM emp.employee WHERE id = 4981;
+-----------+------------+
| last_name | first_name |
+-----------+------------+
| Smith     | Abe        |
+-----------+------------+
1 row in set (0.01 sec)
```

Discrepancies between the server and client version token lists can occur two ways:

- A token name in the `version_tokens_session` value is not present in the server token list. In this case, an `ER_VTOKEN_PLUGIN_TOKEN_NOT_FOUND` error occurs.
- A token value in the `version_tokens_session` value differs from the value of the corresponding token in the server token list. In this case, an `ER_VTOKEN_PLUGIN_TOKEN_MISMATCH` error occurs.

As long as the assignment of server 2 does not change, the client continues to use it for reads and writes. But suppose that the management application wants to change server assignments so that writes for the `emp` database must be sent to server 1 instead of server 2. To do this, it uses `version_tokens_edit()` to modify the `emp` token value on the two servers (and updates its cache of server assignments):

Server 1:

```
mysql> SELECT version_tokens_edit('emp=write');
+---------------------------------+
| version_tokens_edit('emp=write') |
+---------------------------------+
| 1 version tokens updated.       |
+---------------------------------+
```

Server 2:

```
mysql> SELECT version_tokens_edit('emp=read');
+--------------------------------+
| version_tokens_edit('emp=read') |
```

```
+--------------------------------+
| 1 version tokens updated.      |
+--------------------------------+
```

`version_tokens_edit()` modifies the named tokens in the server token list and leaves other tokens unchanged.

The next time the client sends a statement to server 2, its own token list no longer matches the server token list and an error occurs:

```
mysql> UPDATE emp.employee SET salary = salary * 1.1 WHERE id = 4982;
ERROR 3136 (42000): Version token mismatch for emp. Correct value read
```

In this case, the client should contact the management application to obtain updated information about server assignments, select a new server, and send the failed statement to the new server.

Note

Each client must cooperate with Version Tokens by sending only statements in accordance with the token list that it registers with a given server. For example, if a client registers a token list of `'emp=read'`, there is nothing in Version Tokens to prevent the client from sending updates for the `emp` database. The client itself must refrain from doing so.

For each statement received from a client, the server implicitly uses locking, as follows:

- Take a shared lock for each token named in the client token list (that is, in the `version_tokens_session` value)
- Perform the comparison between the server and client token lists
- Execute the statement or produce an error depending on the comparison result
- Release the locks

The server uses shared locks so that comparisons for multiple sessions can occur without blocking, while preventing changes to the tokens for any session that attempts to acquire an exclusive lock before it manipulates tokens of the same names in the server token list.

The preceding example uses only a few of the functions included in the Version Tokens plugin library, but there are others. One set of functions permits the server's list of version tokens to be manipulated and inspected. Another set of functions permits version tokens to be locked and unlocked.

These functions permit the server's list of version tokens to be created, changed, removed, and inspected:

- `version_tokens_set()` completely replaces the current list and assigns a new list. The argument is a semicolon-separated list of *`name=value`* pairs.
- `version_tokens_edit()` enables partial modifications to the current list. It can add new tokens or change the values of existing tokens. The argument is a semicolon-separated list of *`name=value`* pairs.
- `version_tokens_delete()` deletes tokens from the current list. The argument is a semicolon-separated list of token names.
- `version_tokens_show()` displays the current token list. It takes no argument.

Each of those functions, if successful, returns a binary string indicating what action occurred. The following example establishes the server token list, modifies it by adding a new token, deletes some tokens, and displays the resulting token list:

```
mysql> SELECT version_tokens_set('tok1=a;tok2=b');
```

```
+-----------------------------------+
| version_tokens_set('tok1=a;tok2=b') |
+-----------------------------------+
| 2 version tokens set.               |
+-----------------------------------+
mysql> SELECT version_tokens_edit('tok3=c');
+-------------------------------+
| version_tokens_edit('tok3=c') |
+-------------------------------+
| 1 version tokens updated.     |
+-------------------------------+
mysql> SELECT version_tokens_delete('tok2;tok1');
+-----------------------------------+
| version_tokens_delete('tok2;tok1') |
+-----------------------------------+
| 2 version tokens deleted.         |
+-----------------------------------+
mysql> SELECT version_tokens_show();
+-----------------------+
| version_tokens_show() |
+-----------------------+
| tok3=c;               |
+-----------------------+
```

Warnings occur if a token list is malformed:

```
mysql> SELECT version_tokens_set('tok1=a; =c');
+---------------------------------+
| version_tokens_set('tok1=a; =c') |
+---------------------------------+
| 1 version tokens set.            |
+---------------------------------+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 42000
Message: Invalid version token pair encountered. The list provided
         is only partially updated.
1 row in set (0.00 sec)
```

As mentioned previously, version tokens are defined using a semicolon-separated list of *`name=value`* pairs. Consider this invocation of `version_tokens_set()`:

```
mysql> SELECT version_tokens_set('tok1=b;;; tok2= a = b ; tok1 = 1\'2 3"4')
+-----------------------------------------------------------------+
| version_tokens_set('tok1=b;;; tok2= a = b ; tok1 = 1\'2 3"4') |
+-----------------------------------------------------------------+
| 3 version tokens set.                                           |
+-----------------------------------------------------------------+
```

Version Tokens interprets the argument as follows:

- Whitespace around names and values is ignored. Whitespace within names and values is permitted. (For `version_tokens_delete()`, which takes a list of names without values, whitespace around names is ignored.)
- There is no quoting mechanism.
- Order of tokens is not significant except that if a token list contains multiple instances of a given token name, the last value takes precedence over earlier values.

Given those rules, the preceding `version_tokens_set()` call results in a token list with two tokens: `tok1` has the value `1'2 3"4`, and `tok2` has the value `a = b`. To verify this, call `version_tokens_show()`:

```
mysql> SELECT version_tokens_show();
+-------------------------+
```

```
| version_tokens_show()    |
+--------------------------+
| tok2=a = b;tok1=1'2 3"4; |
+--------------------------+
```

If the token list contains two tokens, why did `version_tokens_set()` return the value `3 version tokens set`? That occurred because the original token list contained two definitions for `tok1`, and the second definition replaced the first.

The Version Tokens token-manipulation functions place these constraints on token names and values:

- Token names cannot contain `=` or `;` characters and have a maximum length of 64 characters.
- Token values cannot contain `;` characters. Length of values is constrained by the value of the `max_allowed_packet` system variable.
- Version Tokens treats token names and values as binary strings, so comparisons are case-sensitive.

Version Tokens also includes a set of functions enabling tokens to be locked and unlocked:

- `version_tokens_lock_exclusive()` acquires exclusive version token locks. It takes a list of one or more lock names and a timeout value.
- `version_tokens_lock_shared()` acquires shared version token locks. It takes a list of one or more lock names and a timeout value.
- `version_tokens_unlock()` releases version token locks (exclusive and shared). It takes no argument.

Each locking function returns nonzero for success. Otherwise, an error occurs:

```
mysql> SELECT version_tokens_lock_shared('lock1', 'lock2', 0);
+-----------------------------------------------+
| version_tokens_lock_shared('lock1', 'lock2', 0) |
+-----------------------------------------------+
|                                             1 |
+-----------------------------------------------+

mysql> SELECT version_tokens_lock_shared(NULL, 0);
ERROR 3131 (42000): Incorrect locking service lock name '(null)'.
```

Locking using Version Tokens locking functions is advisory; applications must agree to cooperate.

It is possible to lock nonexisting token names. This does not create the tokens.

Note

Version Tokens locking functions are based on the locking service described at Section 5.6.9.1, “The Locking Service”, and thus have the same semantics for shared and exclusive locks. (Version Tokens uses the locking service routines built into the server, not the locking service function interface, so those functions need not be installed to use Version Tokens.) Locks acquired by Version Tokens use a locking service namespace of `version_token_locks`. Locking service locks can be monitored using the Performance Schema, so this is also true for Version Tokens locks. For details, see Locking Service Monitoring.

For the Version Tokens locking functions, token name arguments are used exactly as specified. Surrounding whitespace is not ignored and `=` and `;` characters are permitted. This is because Version Tokens simply passes the token names to be locked as is to the locking service.

5.6.6.4 Version Tokens Reference

The following discussion serves as a reference to these Version Tokens elements:

- Version Tokens Functions
- Version Tokens System Variables

Version Tokens Functions

The Version Tokens plugin library includes several functions. One set of functions permits the server's list of version tokens to be manipulated and inspected. Another set of functions permits version tokens to be locked and unlocked. The `VERSION_TOKEN_ADMIN` privilege (or the deprecated `SUPER` privilege) is required to invoke any Version Tokens function.

The following functions permit the server's list of version tokens to be created, changed, removed, and inspected. Interpretation of *`name_list`* and *`token_list`* arguments (including whitespace handling) occurs as described in Section 5.6.6.3, “Using Version Tokens”, which provides details about the syntax for specifying tokens, as well as additional examples.

- `version_tokens_delete(`*`name_list`*`)`

  Deletes tokens from the server's list of version tokens using the *`name_list`* argument and returns a binary string that indicates the outcome of the operation. *`name_list`* is a semicolon-separated list of version token names to delete.

```
mysql> SELECT version_tokens_delete('tok1;tok3');
+-----------------------------------+
| version_tokens_delete('tok1;tok3') |
+-----------------------------------+
| 2 version tokens deleted.         |
+-----------------------------------+
```

  An argument of `NULL` is treated as an empty string, which has no effect on the token list.

  `version_tokens_delete()` deletes the tokens named in its argument, if they exist. (It is not an error to delete nonexisting tokens.) To clear the token list entirely without knowing which tokens are in the list, pass `NULL` or a string containing no tokens to `version_tokens_set()`:

```
mysql> SELECT version_tokens_set(NULL);
+------------------------------+
| version_tokens_set(NULL)     |
+------------------------------+
| Version tokens list cleared. |
+------------------------------+
mysql> SELECT version_tokens_set('');
+------------------------------+
| version_tokens_set('')       |
+------------------------------+
| Version tokens list cleared. |
+------------------------------+
```

- `version_tokens_edit(`*`token_list`*`)`

  Modifies the server's list of version tokens using the *`token_list`* argument and returns a binary string that indicates the outcome of the operation. *`token_list`* is a semicolon-separated list of *`name=value`* pairs specifying the name of each token to be defined and its value. If a token exists, its value is updated with the given value. If a token does not exist, it is created with the given value. If the argument is `NULL` or a string containing no tokens, the token list remains unchanged.

```
mysql> SELECT version_tokens_set('tok1=value1;tok2=value2');
+-------------------------------------------+
| version_tokens_set('tok1=value1;tok2=value2') |
+-------------------------------------------+
| 2 version tokens set.                     |
+-------------------------------------------+
mysql> SELECT version_tokens_edit('tok2=new_value2;tok3=new_value3');
+-----------------------------------------------------+
| version_tokens_edit('tok2=new_value2;tok3=new_value3') |
+-----------------------------------------------------+
```

```
| 2 version tokens updated.                              |
+--------------------------------------------------------+
```

- `version_tokens_set(`*`token_list`*`)`

  Replaces the server's list of version tokens with the tokens defined in the *`token_list`* argument and returns a binary string that indicates the outcome of the operation. *`token_list`* is a semicolon-separated list of *`name=value`* pairs specifying the name of each token to be defined and its value. If the argument is `NULL` or a string containing no tokens, the token list is cleared.

```
mysql> SELECT version_tokens_set('tok1=value1;tok2=value2');
+-----------------------------------------------+
| version_tokens_set('tok1=value1;tok2=value2') |
+-----------------------------------------------+
| 2 version tokens set.                         |
+-----------------------------------------------+
```

- `version_tokens_show()`

  Returns the server's list of version tokens as a binary string containing a semicolon-separated list of *`name=value`* pairs.

```
mysql> SELECT version_tokens_show();
+--------------------------+
| version_tokens_show()    |
+--------------------------+
| tok2=value2;tok1=value1; |
+--------------------------+
```

The following functions permit version tokens to be locked and unlocked:

- `version_tokens_lock_exclusive(`*`token_name`*`[, `*`token_name`*`] ..., `*`timeout`*`)`

  Acquires exclusive locks on one or more version tokens, specified by name as strings, timing out with an error if the locks are not acquired within the given timeout value.

```
mysql> SELECT version_tokens_lock_exclusive('lock1', 'lock2', 10);
+-----------------------------------------------------+
| version_tokens_lock_exclusive('lock1', 'lock2', 10) |
+-----------------------------------------------------+
|                                                   1 |
+-----------------------------------------------------+
```

- `version_tokens_lock_shared(`*`token_name`*`[, `*`token_name`*`] ..., `*`timeout`*`)`

  Acquires shared locks on one or more version tokens, specified by name as strings, timing out with an error if the locks are not acquired within the given timeout value.

```
mysql> SELECT version_tokens_lock_shared('lock1', 'lock2', 10);
+-------------------------------------------------+
| version_tokens_lock_shared('lock1', 'lock2', 10) |
+-------------------------------------------------+
|                                               1 |
+-------------------------------------------------+
```

- `version_tokens_unlock()`

  Releases all locks that were acquired within the current session using `version_tokens_lock_exclusive()` and `version_tokens_lock_shared()`.

```
mysql> SELECT version_tokens_unlock();
+-------------------------+
| version_tokens_unlock() |
+-------------------------+
|                       1 |
+-------------------------+
```

The locking functions share these characteristics:

- The return value is nonzero for success. Otherwise, an error occurs.
- Token names are strings.
- In contrast to argument handling for the functions that manipulate the server token list, whitespace surrounding token name arguments is not ignored and `=` and `;` characters are permitted.
- It is possible to lock nonexisting token names. This does not create the tokens.
- Timeout values are nonnegative integers representing the time in seconds to wait to acquire locks before timing out with an error. If the timeout is 0, there is no waiting and the function produces an error if locks cannot be acquired immediately.
- Version Tokens locking functions are based on the locking service described at Section 5.6.9.1, “The Locking Service”.

Version Tokens System Variables

Version Tokens supports the following system variables. These variables are unavailable unless the Version Tokens plugin is installed (see Section 5.6.6.2, “Installing or Uninstalling Version Tokens”).

System variables:

- `version_tokens_session`

Command-Line Format	`--version-tokens-session=value`
System Variable	`version_tokens_session`
Scope	Global, Session
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String
Default Value	`NULL`

The session value of this variable specifies the client version token list and indicates the tokens that the client session requires the server version token list to have.

If the `version_tokens_session` variable is `NULL` (the default) or has an empty value, any server version token list matches. (In effect, an empty value disables matching requirements.)

If the `version_tokens_session` variable has a nonempty value, any mismatch between its value and the server version token list results in an error for any statement the session sends to the server. A mismatch occurs under these conditions:

- A token name in the `version_tokens_session` value is not present in the server token list. In this case, an `ER_VTOKEN_PLUGIN_TOKEN_NOT_FOUND` error occurs.
- A token value in the `version_tokens_session` value differs from the value of the corresponding token in the server token list. In this case, an `ER_VTOKEN_PLUGIN_TOKEN_MISMATCH` error occurs.

It is not a mismatch for the server version token list to include a token not named in the `version_tokens_session` value.

Suppose that a management application has set the server token list as follows:

```
mysql> SELECT version_tokens_set('tok1=a;tok2=b;tok3=c');
+-----------------------------------------+
| version_tokens_set('tok1=a;tok2=b;tok3=c') |
+-----------------------------------------+
```

```
| 3 version tokens set.                      |
+--------------------------------------------+
```

A client registers the tokens it requires the server to match by setting its `version_tokens_session` value. Then, for each subsequent statement sent by the client, the server checks its token list against the client `version_tokens_session` value and produces an error if there is a mismatch:

```
mysql> SET @@SESSION.version_tokens_session = 'tok1=a;tok2=b';
mysql> SELECT 1;
+---+
| 1 |
+---+
| 1 |
+---+

mysql> SET @@SESSION.version_tokens_session = 'tok1=b';
mysql> SELECT 1;
ERROR 3136 (42000): Version token mismatch for tok1. Correct value a
```

The first `SELECT` succeeds because the client tokens `tok1` and `tok2` are present in the server token list and each token has the same value in the server list. The second `SELECT` fails because, although `tok1` is present in the server token list, it has a different value than specified by the client.

At this point, any statement sent by the client fails, unless the server token list changes such that it matches again. Suppose that the management application changes the server token list as follows:

```
mysql> SELECT version_tokens_edit('tok1=b');
+------------------------------+
| version_tokens_edit('tok1=b') |
+------------------------------+
| 1 version tokens updated.     |
+------------------------------+
mysql> SELECT version_tokens_show();
+----------------------+
| version_tokens_show() |
+----------------------+
| tok3=c;tok1=b;tok2=b; |
+----------------------+
```

Now the client `version_tokens_session` value matches the server token list and the client can once again successfully execute statements:

```
mysql> SELECT 1;
+---+
| 1 |
+---+
| 1 |
+---+
```

- `version_tokens_session_number`

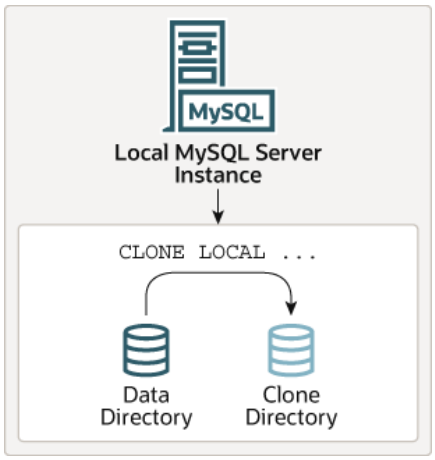
Command-Line Format	`--version-tokens-session-number=#`
System Variable	`version_tokens_session_number`
Scope	Global, Session
Dynamic	No
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`0`

This variable is for internal use.

5.6.7 The Clone Plugin

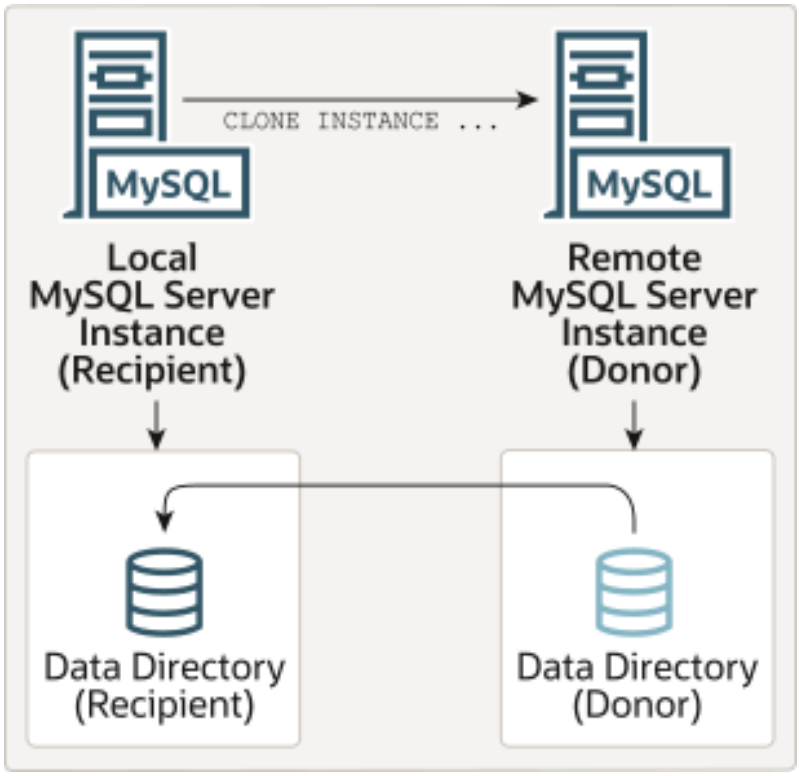
The clone plugin, introduced in MySQL 8.0.17, permits cloning data locally or from a remote MySQL server instance. Cloned data is a physical snapshot of data stored in `InnoDB` that includes schemas, tables, tablespaces, and data dictionary metadata. The cloned data comprises a fully functional data directory, which permits using the clone plugin for MySQL server provisioning.

Figure 5.1 Local Cloning Operation



A local cloning operation clones data from the MySQL server instance where the cloning operation is initiated to a directory on the same server or node where MySQL server instance runs.

Figure 5.2 Remote Cloning Operation



A remote cloning operation involves a local MySQL server instance (the “recipient”) where the cloning operation is initiated, and a remote MySQL server instance (the “donor”) where the source data is located. When a remote cloning operation is initiated on the recipient, cloned data is transferred over the network from the donor to the recipient. By default, a remote cloning operation removes existing user-created data (schemas, tables, tablespaces) and binary logs from the recipient data directory

before cloning data from the donor. Optionally, you can clone data to a different directory on the recipient to avoid removing data from the current recipient data directory.

There is no difference with respect to data that is cloned by a local cloning operation as compared to a remote cloning operation. Both operations clone the same set of data.

The clone plugin supports replication. In addition to cloning data, a cloning operation extracts and transfers replication coordinates from the donor and applies them on the recipient, which enables using the clone plugin for provisioning Group Replication members and replicas. Using the clone plugin for provisioning is considerably faster and more efficient than replicating a large number of transactions (see Section 5.6.7.7, “Cloning for Replication”). Group Replication members can also be configured to use the clone plugin as an alternative method of recovery, so that members automatically choose the most efficient way to retrieve group data from seed members. For more information, see Section 18.5.4.2, “Cloning for Distributed Recovery”.

The clone plugin supports cloning of encrypted and page-compressed data. See Section 5.6.7.5, “Cloning Encrypted Data”, and Section 5.6.7.6, “Cloning Compressed Data”.

The clone plugin must be installed before you can use it. For installation instructions, see Section 5.6.7.1, “Installing the Clone Plugin”. For cloning instructions, see Section 5.6.7.2, “Cloning Data Locally”, and Section 5.6.7.3, “Cloning Remote Data”.

Performance Schema tables and instrumentation are provided for monitoring cloning operations. See Section 5.6.7.10, “Monitoring Cloning Operations”.

5.6.7.1 Installing the Clone Plugin

This section describes how to install and configure the clone plugin. For remote cloning operations, the clone plugin must be installed on the donor and recipient MySQL server instances.

For general information about installing or uninstalling plugins, see Section 5.6.1, “Installing and Uninstalling Plugins”.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, set the value of `plugin_dir` at server startup to tell the server the plugin directory location.

The plugin library file base name is `mysql_clone.so`. The file name suffix differs by platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To load the plugin at server startup, use the `--plugin-load-add` option to name the library file that contains it. With this plugin-loading method, the option must be given each time the server starts. For example, put these lines in your `my.cnf` file, adjusting the plugin library file name extension for your platform as necessary. (The plugin library file name extension depends on your platform. Common suffixes are `.so` for Unix and Unix-like systems, `.dll` for Windows.)

```
[mysqld]
plugin-load-add=mysql_clone.so
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Note

The `--plugin-load-add` option cannot be used to load the clone plugin when restarting the server during an upgrade from a previous MySQL version. For example, after upgrading binaries or packages from MySQL 5.7 to MySQL 8.0, attempting to restart the server with `plugin-load-add=mysql_clone.so` causes this error: `[ERROR] [MY-013238] [Server] Error installing plugin 'clone': Cannot install during upgrade.` The workaround is to upgrade the server before attempting to start the server with `plugin-load-add=mysql_clone.so`.

Alternatively, to load the plugin at runtime, use this statement, adjusting the `.so` suffix for your platform as necessary:

```
INSTALL PLUGIN clone SONAME 'mysql_clone.so';
```

`INSTALL PLUGIN` loads the plugin, and also registers it in the `mysql.plugins` system table to cause the plugin to be loaded for each subsequent normal server startup without the need for `--plugin-load-add`.

To verify plugin installation, examine the Information Schema `PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, "Obtaining Server Plugin Information"). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME = 'clone';
+-----------------------+---------------+
| PLUGIN_NAME           | PLUGIN_STATUS |
+-----------------------+---------------+
| clone                 | ACTIVE        |
+-----------------------+---------------+
```

If the plugin fails to initialize, check the server error log for clone or plugin-related diagnostic messages.

If the plugin has been previously registered with `INSTALL PLUGIN` or is loaded with `--plugin-load-add`, you can use the `--clone` option at server startup to control the plugin activation state. For example, to load the plugin at startup and prevent it from being removed at runtime, use these options:

```
[mysqld]
plugin-load-add=mysql_clone.so
clone=FORCE_PLUS_PERMANENT
```

If you want to prevent the server from running without the clone plugin, use `--clone` with a value of `FORCE` or `FORCE_PLUS_PERMANENT` to force server startup to fail if the plugin does not initialize successfully.

For more information about plugin activation states, see Controlling Plugin Activation State.

5.6.7.2 Cloning Data Locally

The clone plugin supports the following syntax for cloning data locally; that is, cloning data from the local MySQL data directory to another directory on the same server or node where the MySQL server instance runs:

```
CLONE LOCAL DATA DIRECTORY [=] 'clone_dir';
```

To use `CLONE` syntax, the clone plugin must be installed. For installation instructions, see Section 5.6.7.1, "Installing the Clone Plugin".

The `BACKUP_ADMIN` privilege is required to execute `CLONE LOCAL DATA DIRECTORY` statements.

```
mysql> GRANT BACKUP_ADMIN ON *.* TO 'clone_user';
```

where *`clone_user`* is the MySQL user that performs the cloning operation. The user you select to perform the cloning operation can be any MySQL user with the `BACKUP_ADMIN` privilege on *.*.

The following example demonstrates cloning data locally:

```
mysql> CLONE LOCAL DATA DIRECTORY = '/path/to/clone_dir';
```

where */path/to/clone_dir* is the full path of the local directory that data is cloned to. An absolute path is required, and the specified directory ("*`clone_dir`*") must not exist, but the specified path must be an existent path. The MySQL server must have the necessary write access to create the directory.

Note

A local cloning operation does not support cloning of user-created tables or tablespaces that reside outside of the data directory. Attempting to clone such tables or tablespaces causes the following error: `ERROR 1086 (HY000): File '/path/to/tablespace_name.ibd' already exists`. Cloning a tablespace with the same path as the source tablespace would cause a conflict and is therefore prohibited.

All other user-created `InnoDB` tables and tablespaces, the `InnoDB` system tablespace, redo logs, and undo tablespaces are cloned to the specified directory.

If desired, you can start the MySQL server on the cloned directory after the cloning operation is complete.

```
$> mysqld_safe --datadir=clone_dir
```

where *`clone_dir`* is the directory that data was cloned to.

For information about monitoring cloning operation status and progress, see Section 5.6.7.10, “Monitoring Cloning Operations”.

5.6.7.3 Cloning Remote Data

The clone plugin supports the following syntax for cloning remote data; that is, cloning data from a remote MySQL server instance (the donor) and transferring it to the MySQL instance where the cloning operation was initiated (the recipient).

```
CLONE INSTANCE FROM 'user'@'host':port
IDENTIFIED BY 'password'
[DATA DIRECTORY [=] 'clone_dir']
[REQUIRE [NO] SSL];
```

where:

- *`user`* is the clone user on the donor MySQL server instance.
- *`password`* is the *`user`* password.
- *`host`* is the `hostname` address of the donor MySQL server instance. Internet Protocol version 6 (IPv6) address format is not supported. An alias to the IPv6 address can be used instead. An IPv4 address can be used as is.
- *`port`* is the `port` number of the donor MySQL server instance. (The X Protocol port specified by `mysqlx_port` is not supported. Connecting to the donor MySQL server instance through MySQL Router is also not supported.)
- `DATA DIRECTORY [=] 'clone_dir'` is an optional clause used to specify a directory on the recipient for the data you are cloning. Use this option if you do not want to remove existing user-created data (schemas, tables, tablespaces) and binary logs from the recipient data directory. An absolute path is required, and the directory must not exist. The MySQL server must have the necessary write access to create the directory.

  When the optional `DATA DIRECTORY [=] 'clone_dir'` clause is not used, a cloning operation removes user-created data (schemas, tables, tablespaces) and binary logs from the recipient data directory, clones the new data to the recipient data directory, and automatically restarts the server afterward.
- `[REQUIRE [NO] SSL]` explicitly specifies whether an encrypted connection is to be used or not when transferring cloned data over the network. An error is returned if the explicit specification cannot be satisfied. If an SSL clause is not specified, clone attempts to establish an encrypted connection by default, falling back to an insecure connection if the secure connection attempt fails.

A secure connection is required when cloning encrypted data regardless of whether this clause is specified. For more information, see Configuring an Encrypted Connection for Cloning.

Note

By default, user-created `InnoDB` tables and tablespaces that reside in the data directory on the donor MySQL server instance are cloned to the data directory on the recipient MySQL server instance. If the `DATA DIRECTORY [=] 'clone_dir'` clause is specified, they are cloned to the specified directory.

User-created `InnoDB` tables and tablespaces that reside outside of the data directory on the donor MySQL server instance are cloned to the same path on the recipient MySQL server instance. An error is reported if a table or tablespace already exists.

By default, the `InnoDB` system tablespace, redo logs, and undo tablespaces are cloned to the same locations that are configured on the donor (as defined by `innodb_data_home_dir` and `innodb_data_file_path`, `innodb_log_group_home_dir`, and `innodb_undo_directory`, respectively). If the `DATA DIRECTORY [=] 'clone_dir'` clause is specified, those tablespaces and logs are cloned to the specified directory.

Remote Cloning Prerequisites

To perform a cloning operation, the clone plugin must be active on both the donor and recipient MySQL server instances. For installation instructions, see Section 5.6.7.1, “Installing the Clone Plugin”.

A MySQL user on the donor and recipient is required for executing the cloning operation (the “clone user”).

- On the donor, the clone user requires the `BACKUP_ADMIN` privilege for accessing and transferring data from the donor and blocking concurrent DDL during the cloning operation. Concurrent DDL during the cloning operation is blocked on the donor prior to MySQL 8.0.27. From MySQL 8.0.27, concurrent DDL is permitted on the donor by default. See Section 5.6.7.4, “Cloning and Concurrent DDL”.
- On the recipient, the clone user requires the `CLONE_ADMIN` privilege for replacing recipient data, blocking DDL on the recipient during the cloning operation, and automatically restarting the server. The `CLONE_ADMIN` privilege includes `BACKUP_ADMIN` and `SHUTDOWN` privileges implicitly.

Instructions for creating the clone user and granting the required privileges are included in the remote cloning example that follows this prerequisite information.

The following prerequisites are checked when the `CLONE INSTANCE` statement is executed:

- The clone plugin is supported in MYSQL 8.0.17 and higher. The donor and recipient must be the same MySQL server version and release. To determine the MySQL server version and release, issue the following query:

```
mysql> SHOW VARIABLES LIKE 'version';
 +---------------+--------+
| Variable_name | Value  |
+---------------+--------+
| version       | 8.0.17 |
+---------------+--------+
```

  Cloning from a donor MySQL server instance to a hotfix MySQL server instance of the same version and release is supported as of MySQL 8.0.26.

- The donor and recipient MySQL server instances must run on the same operating system and platform. For example, if the donor instance runs on a Linux 64-bit platform, the recipient instance must also run on that platform. Refer to your operating system documentation for information about how to determine your operating system platform.

- The recipient must have enough disk space for the cloned data. By default, user-created data (schemas, tables, tablespaces) and binary logs are removed on the recipient prior to cloning the donor data, so you only require enough space for the donor data. If you clone to a named directory using the `DATA DIRECTORY` clause, you must have enough disk space for the existing recipient data and the cloned data. You can estimate the size of your data by checking the data directory size on your file system and the size of any tablespaces that reside outside of the data directory. When estimating data size on the donor, remember that only `InnoDB` data is cloned. If you store data in other storage engines, adjust your data size estimate accordingly.

- `InnoDB` permits creating some tablespace types outside of the data directory. If the donor MySQL server instance has tablespaces that reside outside of the data directory, the cloning operation must be able access those tablespaces. You can query the Information Schema `FILES` table to identify tablespaces that reside outside of the data directory. Files that reside outside of the data directory have a fully qualified path to a directory other than the data directory.

```
mysql> SELECT FILE_NAME FROM INFORMATION_SCHEMA.FILES;
```

- Plugins that are active on the donor, including any keyring plugin, must also be active on the recipient. You can identify active plugins by issuing a `SHOW PLUGINS` statement or by querying the Information Schema `PLUGINS` table.

- The donor and recipient must have the same MySQL server character set and collation. For information about MySQL server character set and collation configuration, see Section 10.15, “Character Set Configuration”.

- The same `innodb_page_size` and `innodb_data_file_path` settings are required on the donor and recipient. The `innodb_data_file_path` setting on the donor and recipient must specify the same number of data files of an equivalent size. You can check variable settings using `SHOW VARIABLES` syntax.

```
mysql> SHOW VARIABLES LIKE 'innodb_page_size';
mysql> SHOW VARIABLES LIKE 'innodb_data_file_path';
```

- If cloning encrypted or page-compressed data, the donor and recipient must have the same file system block size. For page-compressed data, the recipient file system must support sparse files and hole punching for hole punching to occur on the recipient. For information about these features and how to identify tables and tablespaces that use them, see Section 5.6.7.5, “Cloning Encrypted Data”, and Section 5.6.7.6, “Cloning Compressed Data”. To determine your file system block size, refer to your operating system documentation.

- A secure connection is required if you are cloning encrypted data. See Configuring an Encrypted Connection for Cloning.

- The `clone_valid_donor_list` setting on the recipient must include the host address of the donor MySQL server instance. You can only clone data from a host on the valid donor list. A MySQL user with the `SYSTEM_VARIABLES_ADMIN` privilege is required to configure this variable. Instructions for setting the `clone_valid_donor_list` variable are provided in the remote cloning example that follows this section. You can check the `clone_valid_donor_list` setting using `SHOW VARIABLES` syntax.

```
mysql> SHOW VARIABLES LIKE 'clone_valid_donor_list';
```

- There must be no other cloning operation running. Only a single cloning operation is permitted at a time. To determine if a clone operation is running, query the `clone_status` table. See Monitoring Cloning Operations using Performance Schema Clone Tables.

- The clone plugin transfers data in 1MB packets plus metadata. The minimum required `max_allowed_packet` value is therefore 2MB on the donor and the recipient MySQL server instances. A `max_allowed_packet` value less than 2MB results in an error. Use the following query to check your `max_allowed_packet` setting:

```
mysql> SHOW VARIABLES LIKE 'max_allowed_packet';
```

The following prerequisites also apply:

- Undo tablespace file names on the donor must be unique. When data is cloned to the recipient, undo tablespaces, regardless of their location on the donor, are cloned to the `innodb_undo_directory` location on the recipient or to the directory specified by the `DATA DIRECTORY [=] 'clone_dir'` clause, if used. Duplicate undo tablespace file names on the donor are not permitted for this reason. As of MySQL 8.0.18, an error is reported if duplicate undo tablespace file names are encountered during a cloning operation. Prior to MySQL 8.0.18, cloning undo tablespaces with the same file name could result in undo tablespace files being overwritten on the recipient.

  To view undo tablespace file names on the donor to ensure that they are unique, query `INFORMATION_SCHEMA.FILES`:

  ```
  mysql> SELECT TABLESPACE_NAME, FILE_NAME FROM INFORMATION_SCHEMA.FILES
         WHERE FILE_TYPE LIKE 'UNDO LOG';
  ```

  For information about dropping and adding undo tablespace files, see Section 15.6.3.4, “Undo Tablespaces”.

- By default, the recipient MySQL server instance is restarted (stopped and started) automatically after the data is cloned. For an automatic restart to occur, a monitoring process must be available on the recipient to detect server shutdowns. Otherwise, the cloning operation halts with the following error after the data is cloned, and the recipient MySQL server instance is shut down:

  ```
  ERROR 3707 (HY000): Restart server failed (mysqld is not managed by supervisor process).
  ```

  This error does not indicate a cloning failure. It means that the recipient MySQL server instance must be started again manually after the data is cloned. After starting the server manually, you can connect to the recipient MySQL server instance and check the Performance Schema clone tables to verify that the cloning operation completed successfully (see Monitoring Cloning Operations using Performance Schema Clone Tables.) The `RESTART` statement has the same monitoring process requirement. For more information, see Section 13.7.8.8, “RESTART Statement”. This requirement is not applicable if cloning to a named directory using the `DATA DIRECTORY` clause, as an automatic restart is not performed in this case.

- Several variables control various aspects of a remote cloning operation. Before performing a remote cloning operation, review the variables and adjust settings as necessary to suit your computing environment. Clone variables are set on recipient MySQL server instance where the cloning operation is executed. See Section 5.6.7.13, “Clone System Variables”.

Cloning Remote Data

The following example demonstrates cloning remote data. By default, a remote cloning operation removes user-created data (schemas, tables, tablespaces) and binary logs on the recipient, clones the new data to the recipient data directory, and restarts the MySQL server afterward.

The example assumes that remote cloning prerequisites are met. See Remote Cloning Prerequisites.

1. Login to the donor MySQL server instance with an administrative user account.

   a. Create a clone user with the `BACKUP_ADMIN` privilege.

   ```
   mysql> CREATE USER 'donor_clone_user'@'example.donor.host.com' IDENTIFIED BY 'password';
   mysql> GRANT BACKUP_ADMIN on *.* to 'donor_clone_user'@'example.donor.host.com';
   ```

   b. Install the clone plugin:

   ```
   mysql> INSTALL PLUGIN clone SONAME 'mysql_clone.so';
   ```

2. Login to the recipient MySQL server instance with an administrative user account.

   a. Create a clone user with the `CLONE_ADMIN` privilege.

```
mysql> CREATE USER 'recipient_clone_user'@'example.recipient.host.com' IDENTIFIED BY 'password';
mysql> GRANT CLONE_ADMIN on *.* to 'recipient_clone_user'@'example.recipient.host.com';
```

b. Install the clone plugin:

```
mysql> INSTALL PLUGIN clone SONAME 'mysql_clone.so';
```

c. Add the host address of the donor MySQL server instance to the `clone_valid_donor_list` variable setting.

```
mysql> SET GLOBAL clone_valid_donor_list = 'example.donor.host.com:3306';
```

3. Log on to the recipient MySQL server instance as the clone user you created previously (`recipient_clone_user'@'example.recipient.host.com`) and execute the `CLONE INSTANCE` statement.

```
mysql> CLONE INSTANCE FROM 'donor_clone_user'@'example.donor.host.com':3306
       IDENTIFIED BY 'password';
```

After the data is cloned, the MySQL server instance on the recipient is restarted automatically.

For information about monitoring cloning operation status and progress, see Section 5.6.7.10, “Monitoring Cloning Operations”.

Cloning to a Named Directory

By default, a remote cloning operation removes user-creates data (schemas, tables, tablespaces) and binary logs from the recipient data directory before cloning data from the donor MySQL Server instance. By cloning to a named directory, you can avoid removing data from the current recipient data directory.

The procedure for cloning to a named directory is the same procedure described in Cloning Remote Data with one exception: The `CLONE INSTANCE` statement must include the `DATA DIRECTORY` clause. For example:

```
mysql> CLONE INSTANCE FROM 'user'@'example.donor.host.com':3306
       IDENTIFIED BY 'password'
       DATA DIRECTORY = '/path/to/clone_dir';
```

An absolute path is required, and the directory must not exist. The MySQL server must have the necessary write access to create the directory.

When cloning to a named directory, the recipient MySQL server instance is not restarted automatically after the data is cloned. If you want to restart the MySQL server on the named directory, you must do so manually:

```
$> mysqld_safe --datadir=/path/to/clone_dir
```

where `/path/to/clone_dir` is the path to the named directory on the recipient.

Configuring an Encrypted Connection for Cloning

You can configure an encrypted connection for remote cloning operations to protect data as it is cloned over the network. An encrypted connection is required by default when cloning encrypted data. (see Section 5.6.7.5, “Cloning Encrypted Data”.)

The instructions that follow describe how to configure the recipient MySQL server instance to use an encrypted connection. It is assumed that the donor MySQL server instance is already configured to use encrypted connections. If not, refer to Section 6.3.1, “Configuring MySQL to Use Encrypted Connections” for server-side configuration instructions.

To configure the recipient MySQL server instance to use an encrypted connection:

1. Make the client certificate and key files of the donor MySQL server instance available to the recipient host. Either distribute the files to the recipient host using a secure channel or place them on a mounted partition that is accessible to the recipient host. The client certificate and key files to make available include:

   - `ca.pem`

     The self-signed certificate authority (CA) file.

   - `client-cert.pem`

     The client public key certificate file.

   - `client-key.pem`

     The client private key file.

2. Configure the following SSL options on the recipient MySQL server instance.

   - `clone_ssl_ca`

     Specifies the path to the self-signed certificate authority (CA) file.

   - `clone_ssl_cert`

     Specifies the path to the client public key certificate file.

   - `clone_ssl_key`

     Specifies the path to the client private key file.

   For example:

   ```
   clone_ssl_ca=/path/to/ca.pem
   clone_ssl_cert=/path/to/client-cert.pem
   clone_ssl_key=/path/to/client-key.pem
   ```

3. To require that an encrypted connection is used, include the `REQUIRE SSL` clause when issuing the `CLONE` statement on the recipient.

   ```
   mysql> CLONE INSTANCE FROM 'user'@'example.donor.host.com':3306
          IDENTIFIED BY 'password'
          DATA DIRECTORY = '/path/to/clone_dir'
          REQUIRE SSL;
   ```

   If an SSL clause is not specified, the clone plugin attempts to establish an encrypted connection by default, falling back to an unencrypted connection if the encrypted connection attempt fails.

Note

If you are cloning encrypted data, an encrypted connection is required by default regardless of whether the `REQUIRE SSL` clause is specified. Using `REQUIRE NO SSL` causes an error if you attempt to clone encrypted data.

5.6.7.4 Cloning and Concurrent DDL

Prior to MySQL 8.0.27, DDL operations on the donor and recipient MySQL Server instances, including `TRUNCATE TABLE`, are not permitted during a cloning operation. This limitation should be considered when selecting data sources. A workaround is to use dedicated donor instances, which can accommodate DDL operations being blocked while data is cloned.

To prevent concurrent DDL during a cloning operation, an exclusive backup lock is acquired on the donor and recipient. The `clone_ddl_timeout` variable defines the time in seconds on the donor

and recipient that a cloning operation waits for a backup lock. The default setting is 300 seconds. If a backup lock is not obtained with the specified time limit, the cloning operation fails with an error.

From MySQL 8.0.27, concurrent DDL is permitted on the donor by default. Concurrent DDL support on the donor is controlled by the `clone_block_ddl` variable. Concurrent DDL support can be enabled and disabled dynamically using a `SET` statement.

```
SET GLOBAL clone_block_ddl={OFF|ON}
```

The default setting is `clone_block_ddl=OFF`, which permits concurrent DDL on the donor.

Whether the effect of a concurrent DDL operation is cloned or not depends on whether the DDL operation finishes before the dynamic snapshot is taken by the cloning operation.

DDL operations that are not permitted during a cloning operation regardless of the `clone_block_ddl` setting include:

- `ALTER TABLE tbl_name DISCARD TABLESPACE;`
- `ALTER TABLE tbl_name IMPORT TABLESPACE;`
- `ALTER INSTANCE DISABLE INNODB REDO_LOG;`

5.6.7.5 Cloning Encrypted Data

Cloning of encrypted data is supported. The following requirements apply:

- A secure connection is required when cloning remote data to ensure safe transfer of unencrypted tablespace keys over the network. Tablespace keys are decrypted at the donor before transport and re-encrypted at the recipient using the recipient master key. An error is reported if an encrypted connection is not available or the `REQUIRE NO SSL` clause is used in the `CLONE INSTANCE` statement. For information about configuring an encrypted connection for cloning, see Configuring an Encrypted Connection for Cloning.
- When cloning data to a local data directory that uses a locally managed keyring, the same keyring must be used when starting the MySQL server on the clone directory.
- When cloning data to a remote data directory (the recipient directory) that uses a locally managed keyring, the recipient keyring must be used when starting the MySQL sever on the cloned directory.

Note

The `innodb_redo_log_encrypt` and `innodb_undo_log_encrypt` variable settings cannot be modified while a cloning operation is in progress.

For information about the data encryption feature, see Section 15.13, “InnoDB Data-at-Rest Encryption”.

5.6.7.6 Cloning Compressed Data

Cloning of page-compressed data is supported. The following requirements apply when cloning remote data:

- The recipient file system must support sparse files and hole punching for hole punching to occur on the recipient.
- The donor and recipient file systems must have the same block size. If file system block sizes differ, an error similar to the following is reported: `ERROR 3868 (HY000): Clone Configuration FS Block Size: Donor value: 114688 is different from Recipient value: 4096`.

For information about the page compression feature, see Section 15.9.2, “InnoDB Page Compression”.

5.6.7.7 Cloning for Replication

The clone plugin supports replication. In addition to cloning data, a cloning operation extracts replication coordinates from the donor and transfers them to the recipient, which enables using the clone plugin for provisioning Group Replication members and replicas. Using the clone plugin for provisioning is considerably faster and more efficient than replicating a large number of transactions.

Group Replication members can also be configured to use the clone plugin as an option for distributed recovery, in which case joining members automatically choose the most efficient way to retrieve group data from existing group members. For more information, see Section 18.5.4.2, “Cloning for Distributed Recovery”.

During the cloning operation, both the binary log position (filename, offset) and the `gtid_executed` GTID set are extracted and transferred from the donor MySQL server instance to the recipient. This data permits initiating replication at a consistent position in the replication stream. The binary logs and relay logs, which are held in files, are not copied from the donor to the recipient. To initiate replication, the binary logs required for the recipient to catch up to the donor must not be purged between the time that the data is cloned and the time that replication is started. If the required binary logs are not available, a replication handshake error is reported. A cloned instance should therefore be added to a replication group without excessive delay to avoid required binary logs being purged or the new member lagging behind significantly, requiring more recovery time.

- Issue this query on a cloned MySQL server instance to check the binary log position that was transferred to the recipient:

```
mysql> SELECT BINLOG_FILE, BINLOG_POSITION FROM performance_schema.clone_status;
```

- Issue this query on a cloned MySQL server instance to check the `gtid_executed` GTID set that was transferred to the recipient:

```
mysql> SELECT @@GLOBAL.GTID_EXECUTED;
```

By default in MySQL 8.0, the replication metadata repositories are held in tables that are copied from the donor to the recipient during the cloning operation. The replication metadata repositories hold replication-related configuration settings that can be used to resume replication correctly after the cloning operation.

- In MySQL 8.0.17 and 8.0.18, only the table `mysql.slave_master_info` (the connection metadata repository) is copied.
- From MySQL 8.0.19, the tables `mysql.slave_relay_log_info` (the applier metadata repository) and `mysql.slave_worker_info` (the applier worker metadata repository) are also copied.

For a list of what is included in each table, see Section 17.2.4.2, “Replication Metadata Repositories”. Note that if the settings `master_info_repository=FILE` and `relay_log_info_repository=FILE` are used on the server (which is not the default in MySQL 8.0 and is deprecated), the replication metadata repositories are not cloned; they are only cloned if `TABLE` is set.

To clone for replication, perform the following steps:

1. For a new group member for Group Replication, first configure the MySQL Server instance for Group Replication, following the instructions in Section 18.2.1.6, “Adding Instances to the Group”. Also set up the prerequisites for cloning described in Section 18.5.4.2, “Cloning for Distributed Recovery”. When you issue `START GROUP_REPLICATION` on the joining member, the cloning operation is managed automatically by Group Replication, so you do not need to carry out the operation manually, and you do not need to perform any further setup steps on the joining member.
2. For a replica in a source/replica MySQL replication topology, first clone the data from the donor MySQL server instance to the recipient manually. The donor must be a source or replica in the replication topology. For cloning instructions, see Section 5.6.7.3, “Cloning Remote Data”.

3. After the cloning operation completes successfully, if you want to use the same replication channels on the recipient MySQL server instance that were present on the donor, verify which of them can resume replication automatically in the source/replica MySQL replication topology, and which need to be set up manually.

   - For GTID-based replication, if the recipient is configured with `gtid_mode=ON` and has cloned from a donor with `gtid_mode=ON`, `ON_PERMISSIVE`, or `OFF_PERMISSIVE`, the `gtid_executed` GTID set from the donor is applied on the recipient. If the recipient is cloned from a replica already in the topology, replication channels on the recipient that use GTID auto-positioning can resume replication automatically after the cloning operation when the channel is started. You do not need to perform any manual setup if you just want to use these same channels.

   - For binary log file position based replication, if the recipient is at MySQL 8.0.17 or 8.0.18, the binary log position from the donor is not applied on the recipient, only recorded in the Performance Schema `clone_status` table. Replication channels on the recipient that use binary log file position based replication must therefore be set up manually to resume replication after the cloning operation. Ensure that these channels are not configured to start replication automatically at server startup, as they do not yet have the binary log position and attempt to start replication from the beginning.

   - For binary log file position based replication, if the recipient is at MySQL 8.0.19 or above, the binary log position from the donor is applied on the recipient. Replication channels on the recipient that use binary log file position based replication automatically attempt to carry out the relay log recovery process, using the cloned relay log information, before restarting replication. For a single-threaded replica (`replica_parallel_workers` or `slave_parallel_workers` is set to 0), relay log recovery should succeed in the absence of any other issues, enabling the channel to resume replication with no further setup. For a multithreaded replica (`replica_parallel_workers` or `slave_parallel_workers` is greater than 0), relay log recovery is likely to fail because it cannot usually be completed automatically. In this case, an error message is issued, and you must set the channel up manually.

4. If you need to set up cloned replication channels manually, or want to use different replication channels on the recipient, the following instructions provide a summary and abbreviated examples for adding a recipient MySQL server instance to a replication topology. Also refer to the detailed instructions that apply to your replication setup.

   - To add a recipient MySQL server instance to a MySQL replication topology that uses GTID-based transactions as the replication data source, configure the instance as required, following the instructions in Section 17.1.3.4, “Setting Up Replication Using GTIDs”. Add replication channels for the instance as shown in the following abbreviated example. The `CHANGE REPLICATION SOURCE TO` statement (from MySQL 8.0.23) or `CHANGE MASTER TO` statement (before MySQL 8.0.23) must define the host address and port number of the source, and the `SOURCE_AUTO_POSITION` | `MASTER_AUTO_POSITION` option should be enabled, as shown:

     ```
     mysql> CHANGE MASTER TO MASTER_HOST = 'source_host_name', MASTER_PORT = source_port_num,
            ...
            MASTER_AUTO_POSITION = 1,
            FOR CHANNEL 'setup_channel';
     mysql> START SLAVE USER = 'user_name' PASSWORD = 'password' FOR CHANNEL 'setup_channel';

     Or from MySQL 8.0.22 and 8.0.23:

     mysql> CHANGE SOURCE TO SOURCE_HOST = 'source_host_name', SOURCE_PORT = source_port_num,
            ...
            SOURCE_AUTO_POSITION = 1,
            FOR CHANNEL 'setup_channel';
     mysql> START REPLICA USER = 'user_name' PASSWORD = 'password' FOR CHANNEL 'setup_channel';
     ```

   - To add a recipient MySQL server instance to a MySQL replication topology that uses binary log file position based replication, configure the instance as required, following the instructions in

Section 17.1.2, “Setting Up Binary Log File Position Based Replication”. Add replication channels for the instance as shown in the following abbreviated example, using the binary log position that was transferred to the recipient during the cloning operation:

```
mysql> SELECT BINLOG_FILE, BINLOG_POSITION FROM performance_schema.clone_status;
mysql> CHANGE MASTER TO MASTER_HOST = 'source_host_name', MASTER_PORT = source_port_num,
       ...
       MASTER_LOG_FILE = 'source_log_name',
       MASTER_LOG_POS = source_log_pos,
       FOR CHANNEL 'setup_channel';
mysql> START SLAVE USER = 'user_name' PASSWORD = 'password' FOR CHANNEL 'setup_channel';

Or from MySQL 8.0.22 and 8.0.23:

mysql> SELECT BINLOG_FILE, BINLOG_POSITION FROM performance_schema.clone_status;
mysql> CHANGE SOURCE TO SOURCE_HOST = 'source_host_name', SOURCE_PORT = source_port_num,
       ...
       SOURCE_LOG_FILE = 'source_log_name',
       SOURCE_LOG_POS = source_log_pos,
       FOR CHANNEL 'setup_channel';
mysql> START REPLICA USER = 'user_name' PASSWORD = 'password' FOR CHANNEL 'setup_channel';
```

5.6.7.8 Directories and Files Created During a Cloning Operation

When data is cloned, the following directories and files are created for internal use. They should not be modified.

- `#clone`: Contains internal clone files used by the cloning operation. Created in the directory that data is cloned to.
- `#ib_archive`: Contains internally archived log files, archived on the donor during the cloning operation.
- `*.#clone` files: Temporary data files created on the recipient while data is removed from the recipient data directory and new data is cloned during a remote cloning operation.

5.6.7.9 Remote Cloning Operation Failure Handling

This section describes failure handing at different stages of a cloning operation.

1. Prerequisites are checked (see Remote Cloning Prerequisites).
   - If a failure occurs during the prerequisite check, the `CLONE INSTANCE` operation reports an error.
2. Prior to MySQL 8.0.27, a backup lock on the donor and recipient blocks concurrent DDL operations. From MySQL 8.0.27, concurrent DDL on the donor is blocked only if the `clone_block_ddl` variable is set to `ON` (the default setting is `OFF`). See Section 5.6.7.4, “Cloning and Concurrent DDL”.
   - If the cloning operation is unable to obtain a DDL lock within the time limit specified by the `clone_ddl_timeout` variable, an error is reported.
3. User-created data (schemas, tables, tablespaces) and binary logs on the recipient are removed before data is cloned to the recipient data directory.
   - When user-created data and binary logs are removed from the recipient data directory during a remote cloning operation, the data is not saved and may be lost if a failure occurs. If the data is of importance, a backup should be taken before initiating a remote cloning operation.

     For informational purposes, warnings are printed to the server error log to specify when data removal starts and finishes:

     ```
     [Warning] [MY-013453] [InnoDB] Clone removing all user data for provisioning:
     Started...
     ```

```
[Warning] [MY-013453] [InnoDB] Clone removing all user data for provisioning:
Finished
```

If a failure occurs while removing data, the recipient may be left with a partial set of schemas, tables, and tablespaces that existed before the cloning operation. Any time during the execution of a cloning operation or after a failure, the server is always in a consistent state.

4. Data is cloned from the donor. User-created data, dictionary metadata, and other system data are cloned.

   - If a failure occurs while cloning data, the cloning operation is rolled back and all cloned data removed. At this stage, the previously existing user-created data and binary logs on the recipient have also been removed.

     Should this scenario occur, you can either rectify the cause of the failure and re-execute the cloning operation, or forgo the cloning operation and restore the recipient data from a backup taken before the cloning operation.

5. The server is restarted automatically (applies to remote cloning operations that do not clone to a named directory). During startup, typical server startup tasks are performed.

   - If the automatic server restart fails, you can restart the server manually to complete the cloning operation.

Before MySQL 8.0.24, if a network error occurs during a cloning operation, the operation resumes if the error is resolved within five minutes. From MySQL 8.0.24, the operation resumes if the error is resolved within the time specified by the `clone_donor_timeout_after_network_failure` variable defined on the donor instance. The `clone_donor_timeout_after_network_failure` default setting is 5 minutes but a range of 0 to 30 minutes is supported. If the operation does not resume within the allotted time, it aborts and returns an error, and the donor drops the snapshot. A setting of zero causes the donor to drop the snapshot immediately when a network error occurs. Configuring a longer timeout allows more time for resolving network issues but also increases the size of the delta on the donor instance, which increases clone recovery time as well as replication lag in cases where the clone is intended as a replica or replication group member.

Prior to MySQL 8.0.24, donor threads use the MySQL Server `wait_timeout` setting when listening for Clone protocol commands. As a result, a low `wait_timeout` setting could cause a long running remote cloning operation to timeout. From MySQL 8.0.24, the Clone idle timeout is set to the default `wait_timeout` setting, which is 28800 seconds (8 hours).

5.6.7.10 Monitoring Cloning Operations

This section describes options for monitoring cloning operations.

- Monitoring Cloning Operations using Performance Schema Clone Tables
- Monitoring Cloning Operations Using Performance Schema Stage Events
- Monitoring Cloning Operations Using Performance Schema Clone Instrumentation
- The Com_clone Status Variable

Monitoring Cloning Operations using Performance Schema Clone Tables

A cloning operation may take some time to complete, depending on the amount of data and other factors related to data transfer. You can monitor the status and progress of a cloning operation on the recipient MySQL server instance using the `clone_status` and `clone_progress` Performance Schema tables.

Note

The `clone_status` and `clone_progress` Performance Schema tables can be used to monitor a cloning operation on the recipient MySQL server instance

> only. To monitor a cloning operation on the donor MySQL server instance, use the clone stage events, as described in Monitoring Cloning Operations Using Performance Schema Stage Events.

- The `clone_status` table provides the state of the current or last executed cloning operation. A clone operation has four possible states: `Not Started`, `In Progress`, `Completed`, and `Failed`.
- The `clone_progress` table provides progress information for the current or last executed clone operation, by stage. The stages of a cloning operation include `DROP DATA`, `FILE COPY`, `PAGE_COPY`, `REDO_COPY`, `FILE_SYNC`, `RESTART`, and `RECOVERY`.

The `SELECT` and `EXECUTE` privileges on the Performance Schema is required to access the Performance Schema clone tables.

To check the state of a cloning operation:

1. Connect to the recipient MySQL server instance.
2. Query the `clone_status` table:

```
mysql> SELECT STATE FROM performance_schema.clone_status;
+-----------+
| STATE     |
+-----------+
| Completed |
+-----------+
```

Should a failure occur during a cloning operation, you can query the `clone_status` table for error information:

```
mysql> SELECT STATE, ERROR_NO, ERROR_MESSAGE FROM performance_schema.clone_status;
+-----------+----------+---------------+
| STATE     | ERROR_NO | ERROR_MESSAGE |
+-----------+----------+---------------+
| Failed    |      xxx | "xxxxxxxxxx" |
+-----------+----------+---------------+
```

To review the details of each stage of a cloning operation:

1. Connect to the recipient MySQL server instance.
2. Query the `clone_progress` table. For example, the following query provides state and end time data for each stage of the cloning operation:

```
mysql> SELECT STAGE, STATE, END_TIME FROM performance_schema.clone_progress;
+-----------+-----------+----------------------------+
| stage     | state     | end_time                   |
+-----------+-----------+----------------------------+
| DROP DATA | Completed | 2019-01-27 22:45:43.141261 |
| FILE COPY | Completed | 2019-01-27 22:45:44.457572 |
| PAGE COPY | Completed | 2019-01-27 22:45:44.577330 |
| REDO COPY | Completed | 2019-01-27 22:45:44.679570 |
| FILE SYNC | Completed | 2019-01-27 22:45:44.918547 |
| RESTART   | Completed | 2019-01-27 22:45:48.583565 |
| RECOVERY  | Completed | 2019-01-27 22:45:49.626595 |
+-----------+-----------+----------------------------+
```

For other clone status and progress data points that you can monitor, refer to Section 27.12.19, “Performance Schema Clone Tables”.

Monitoring Cloning Operations Using Performance Schema Stage Events

A cloning operation may take some time to complete, depending on the amount of data and other factors related to data transfer. There are three stage events for monitoring the progress of a cloning operation. Each stage event reports `WORK_COMPLETED` and `WORK_ESTIMATED` values. Reported values are revised as the operation progresses.

This method of monitoring a cloning operation can be used on the donor or recipient MySQL server instance.

In order of occurrence, cloning operation stage events include:

- `stage/innodb/clone (file copy)`: Indicates progress of the file copy phase of the cloning operation. `WORK_ESTIMATED` and `WORK_COMPLETED` units are file chunks. The number of files to be transferred is known at the start of the file copy phase, and the number of chunks is estimated based on the number of files. `WORK_ESTIMATED` is set to the number of estimated file chunks. `WORK_COMPLETED` is updated after each chunk is sent.
- `stage/innodb/clone (page copy)`: Indicates progress of the page copy phase of cloning operation. `WORK_ESTIMATED` and `WORK_COMPLETED` units are pages. Once the file copy phase is completed, the number of pages to be transferred is known, and `WORK_ESTIMATED` is set to this value. `WORK_COMPLETED` is updated after each page is sent.
- `stage/innodb/clone (redo copy)`: Indicates progress of the redo copy phase of cloning operation. `WORK_ESTIMATED` and `WORK_COMPLETED` units are redo chunks. Once the page copy phase is completed, the number of redo chunks to be transferred is known, and `WORK_ESTIMATED` is set to this value. `WORK_COMPLETED` is updated after each chunk is sent.

The following example demonstrates how to enable `stage/innodb/clone%` event instruments and related consumer tables to monitor a cloning operation. For information about Performance Schema stage event instruments and related consumers, see Section 27.12.5, “Performance Schema Stage Event Tables”.

1. Enable the `stage/innodb/clone%` instruments:

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

2. Enable the stage event consumer tables, which include `events_stages_current`, `events_stages_history`, and `events_stages_history_long`.

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

3. Run a cloning operation. In this example, a local data directory is cloned to a directory named `cloned_dir`.

```
mysql> CLONE LOCAL DATA DIRECTORY = '/path/to/cloned_dir';
```

4. Check the progress of the cloning operation by querying the Performance Schema `events_stages_current` table. The stage event shown differs depending on the cloning phase that is in progress. The `WORK_COMPLETED` column shows the work completed. The `WORK_ESTIMATED` column shows the work required in total.

```
mysql> SELECT EVENT_NAME, WORK_COMPLETED, WORK_ESTIMATED FROM performance_schema.events_stages_curre
       WHERE EVENT_NAME LIKE 'stage/innodb/clone%';
+-------------------------------+----------------+----------------+
| EVENT_NAME                    | WORK_COMPLETED | WORK_ESTIMATED |
+-------------------------------+----------------+----------------+
| stage/innodb/clone (redo copy) |              1 |              1 |
+-------------------------------+----------------+----------------+
```

   The `events_stages_current` table returns an empty set if the cloning operation has finished. In this case, you can check the `events_stages_history` table to view event data for the completed operation. For example:

```
mysql> SELECT EVENT_NAME, WORK_COMPLETED, WORK_ESTIMATED FROM events_stages_history
       WHERE EVENT_NAME LIKE 'stage/innodb/clone%';
+-------------------------------+----------------+----------------+
| EVENT_NAME                    | WORK_COMPLETED | WORK_ESTIMATED |
+-------------------------------+----------------+----------------+
| stage/innodb/clone (file copy) |            301 |            301 |
| stage/innodb/clone (page copy) |              0 |              0 |
```

```
| stage/innodb/clone (redo copy) |              1 |              1 |
+--------------------------------+----------------+----------------+
```

Monitoring Cloning Operations Using Performance Schema Clone Instrumentation

Performance Schema provides instrumentation for advanced performance monitoring of clone operations. To view the available clone instrumentation, and issue the following query:

```
mysql> SELECT NAME,ENABLED FROM performance_schema.setup_instruments
       WHERE NAME LIKE '%clone%';
+-----------------------------------------------------+---------+
| NAME                                                | ENABLED |
+-----------------------------------------------------+---------+
| wait/synch/mutex/innodb/clone_snapshot_mutex        | NO      |
| wait/synch/mutex/innodb/clone_sys_mutex             | NO      |
| wait/synch/mutex/innodb/clone_task_mutex            | NO      |
| wait/synch/mutex/group_rpl/LOCK_clone_donor_list    | NO      |
| wait/synch/mutex/group_rpl/LOCK_clone_handler_run   | NO      |
| wait/synch/mutex/group_rpl/LOCK_clone_query         | NO      |
| wait/synch/mutex/group_rpl/LOCK_clone_read_mode     | NO      |
| wait/synch/cond/group_rpl/COND_clone_handler_run    | NO      |
| wait/io/file/innodb/innodb_clone_file               | YES     |
| stage/innodb/clone (file copy)                      | YES     |
| stage/innodb/clone (redo copy)                      | YES     |
| stage/innodb/clone (page copy)                      | YES     |
| statement/abstract/clone                            | YES     |
| statement/clone/local                               | YES     |
| statement/clone/client                              | YES     |
| statement/clone/server                              | YES     |
| memory/innodb/clone                                 | YES     |
| memory/clone/data                                   | YES     |
+-----------------------------------------------------+---------+
```

Wait Instruments

Performance schema wait instruments track events that take time. Clone wait event instruments include:

- `wait/synch/mutex/innodb/clone_snapshot_mutex`: Tracks wait events for the clone snapshot mutex, which synchronizes access to the dynamic snapshot object (on the donor and recipient) between multiple clone threads.
- `wait/synch/mutex/innodb/clone_sys_mutex`: Tracks wait events for the clone sys mutex. There is one clone system object in a MySQL server instance. This mutex synchronizes access to the clone system object on the donor and recipient. It is acquired by clone threads and other foreground and background threads.
- `wait/synch/mutex/innodb/clone_task_mutex`: Tracks wait events for the clone task mutex, used for clone task management. The `clone_task_mutex` is acquired by clone threads.
- `wait/io/file/innodb/innodb_clone_file`: Tracks all I/O wait operations for files that clone operates on.

For information about monitoring `InnoDB` mutex waits, see Section 15.16.2, “Monitoring InnoDB Mutex Waits Using Performance Schema”. For information about monitoring wait events in general, see Section 27.12.4, “Performance Schema Wait Event Tables”.

Stage Instruments

Performance Schema stage events track steps that occur during the statement-execution process. Clone stage event instruments include:

- `stage/innodb/clone (file copy)`: Indicates progress of the file copy phase of the cloning operation.
- `stage/innodb/clone (redo copy)`: Indicates progress of the redo copy phase of cloning operation.

- `stage/innodb/clone (page copy)`: Indicates progress of the page copy phase of cloning operation.

For information about monitoring cloning operations using stage events, see Monitoring Cloning Operations Using Performance Schema Stage Events. For general information about monitoring stage events, see Section 27.12.5, “Performance Schema Stage Event Tables”.

Statement Instruments

Performance Schema statement events track statement execution. When a clone operation is initiated, the different statement types tracked by clone statement instruments may be executed in parallel. You can observe these statement events in the Performance Schema statement event tables. The number of statements that execute depends on the `clone_max_concurrency` and `clone_autotune_concurrency` settings.

Clone statement event instruments include:

- `statement/abstract/clone`: Tracks statement events for any clone operation before it is classified as a local, client, or server operation type.
- `statement/clone/local`: Tracks clone statement events for local clone operations; generated when executing a `CLONE LOCAL` statement.
- `statement/clone/client`: Tracks remote cloning statement events that occur on the recipient MySQL server instance; generated when executing a `CLONE INSTANCE` statement on the recipient.
- `statement/clone/server`: Tracks remote cloning statement events that occur on the donor MySQL server instance; generated when executing a `CLONE INSTANCE` statement on the recipient.

For information about monitoring Performance Schema statement events, see Section 27.12.6, “Performance Schema Statement Event Tables”.

Memory Instruments

Performance Schema memory instruments track memory usage. Clone memory usage instruments include:

- `memory/innodb/clone`: Tracks memory allocated by `InnoDB` for the dynamic snapshot.
- `memory/clone/data`: Tracks memory allocated by the clone plugin during a clone operation.

For information about monitoring memory usage using Performance Schema, see Section 27.12.20.10, “Memory Summary Tables”.

The Com_clone Status Variable

The `Com_clone` status variable provides a count of `CLONE` statement executions.

For more information, refer to the discussion about `Com_xxx` statement counter variables in Section 5.1.10, “Server Status Variables”.

5.6.7.11 Stopping a Cloning Operation

If necessary, you can stop a cloning operation with a `KILL QUERY` *`processlist_id`* statement.

On the recipient MySQL server instance, you can retrieve the processlist identifier (PID) for a cloning operation from the `PID` column of the `clone_status` table.

```
mysql> SELECT * FROM performance_schema.clone_status\G
*************************** 1. row ***************************
           ID: 1
          PID: 8
        STATE: In Progress
   BEGIN_TIME: 2019-07-15 11:58:36.767
     END_TIME: NULL
```

```
         SOURCE: LOCAL INSTANCE
    DESTINATION: /path/to/clone_dir/
       ERROR_NO: 0
  ERROR_MESSAGE:
    BINLOG_FILE:
BINLOG_POSITION: 0
  GTID_EXECUTED:
```

You can also retrieve the processlist identifier from the `ID` column of the `INFORMATION_SCHEMA PROCESSLIST` table, the `Id` column of `SHOW PROCESSLIST` output, or the `PROCESSLIST_ID` column of the Performance Schema `threads` table. These methods of obtaining the PID information can be used on the donor or recipient MySQL server instance.

5.6.7.12 Clone System Variable Reference

Table 5.7 Clone System Variable Reference

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
clone_autotune_concurrency	Yes	Yes	Yes		Global	Yes
clone_block_ddl	Yes	Yes	Yes		Global	Yes
clone_buffer_size	Yes	Yes	Yes		Global	Yes
clone_ddl_timeout	Yes	Yes	Yes		Global	Yes
clone_delay_after_data_drop	Yes	Yes	Yes		Global	Yes
clone_donor_timeout_after_network_failure	Yes	Yes	Yes		Global	Yes
clone_enable_compression	Yes	Yes	Yes		Global	Yes
clone_max_concurrency	Yes	Yes	Yes		Global	Yes
clone_max_data_bandwidth	Yes	Yes	Yes		Global	Yes
clone_max_network_bandwidth	Yes	Yes	Yes		Global	Yes
clone_ssl_ca	Yes	Yes	Yes		Global	Yes
clone_ssl_cert	Yes	Yes	Yes		Global	Yes
clone_ssl_key	Yes	Yes	Yes		Global	Yes
clone_valid_donor_list	Yes	Yes	Yes		Global	Yes

5.6.7.13 Clone System Variables

This section describes the system variables that control operation of the clone plugin. If values specified at startup are incorrect, the clone plugin may fail to initialize properly and the server does not load it. In this case, the server may also produce error messages for other clone settings because it does not recognize them.

Each system variable has a default value. System variables can be set at server startup using options on the command line or in an option file. They can be changed dynamically at runtime using the `SET` statement, which enables you to modify operation of the server without having to stop and restart it.

Setting a global system variable runtime value normally requires the `SYSTEM_VARIABLES_ADMIN` privilege (or the deprecated `SUPER` privilege). For more information, see Section 5.1.9.1, “System Variable Privileges”.

Clone variables are configured on the recipient MySQL server instance where the cloning operation is executed.

- `clone_autotune_concurrency`

Command-Line Format	`--clone-autotune-concurrency`
Introduced	8.0.17
System Variable	`clone_autotune_concurrency`

Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	ON

When clone_autotune_concurrency is enabled (the default), additional threads for remote cloning operations are spawned dynamically to optimize data transfer speed. The setting is applicable to recipient MySQL server instance only.

During a cloning operation, the number of threads increases incrementally toward a target of double the current thread count. The effect on the data transfer speed is evaluated at each increment. The process either continues or stops according to the following rules:

- If the data transfer speed degrades more than 5% with an incremental increase, the process stops.
- If there is at least a 5% improvement after reaching 25% of the target, the process continues. Otherwise, the process stops.
- If there is at least a 10% improvement after reaching 50% of the target, the process continues. Otherwise, the process stops.
- If there is at least a 25% improvement after reaching the target, the process continues toward a new target of double the current thread count. Otherwise, the process stops.

The autotuning process does not support decreasing the number of threads.

The clone_max_concurrency variable defines the maximum number of threads that can be spawned.

If clone_autotune_concurrency is disabled, clone_max_concurrency defines the number of threads spawned for a remote cloning operation.

- clone_buffer_size

Command-Line Format	--clone-buffer-size
Introduced	8.0.17
System Variable	clone_buffer_size
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	4194304
Minimum Value	1048576
Maximum Value	268435456
Unit	bytes

Defines the size of the intermediate buffer used when transferring data during a local cloning operation. The default value is 4 mebibytes (MiB). A larger buffer size may permit I/O device drivers to fetch data in parallel, which can improve cloning performance.

- clone_block_ddl

Command-Line Format	--clone-block-ddl

Introduced	8.0.27
System Variable	clone_block_ddl
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

Enables an exclusive backup lock on the donor MySQL Server instance during a cloning operation, which blocks concurrent DDL operations on the donor. See Section 5.6.7.4, “Cloning and Concurrent DDL”.

- clone_delay_after_data_drop

Command-Line Format	--clone-delay-after-data-drop
Introduced	8.0.29
System Variable	clone_delay_after_data_drop
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	0
Minimum Value	0
Maximum Value	3600
Unit	bytes

Specifies a delay period immediately after removing existing data on the recipient MySQL Server instance at the start of a remote cloning operation. The delay is intended to provide enough time for the file system on the recipient host to free space before data is cloned from the donor MySQL Server instance. Certain file systems such as VxFS free space asynchronously in a background process. On these file systems, cloning data too soon after dropping existing data can result in clone operation failures due to insufficient space. The maximum delay period is 3600 seconds (1 hour). The default setting is 0 (no delay).

This variable is applicable to remote cloning operation only and is configured on the recipient MySQL Server instance.

- clone_ddl_timeout

Command-Line Format	--clone-ddl-timeout
Introduced	8.0.17
System Variable	clone_ddl_timeout
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	300
Minimum Value	0
Maximum Value	2592000

Unit	seconds

The time in seconds that a cloning operation waits for a backup lock. The backup lock blocks concurrent DDL when executing a cloning operation. This setting is applied on both the donor and recipient MySQL server instances.

A setting of 0 means that the cloning operation does not wait for a backup lock. In this case, executing a concurrent DDL operation can cause the cloning operation to fail.

Prior to MySQL 8.0.27, the backup lock blocks concurrent DDL operations on both the donor and recipient during a cloning operation, and a cloning operation cannot proceed until current DDL operations finish. As of MySQL 8.0.27, concurrent DDL is permitted on the donor during a cloning operation if `clone_block_ddl` variable is set to `OFF` (the default). In this case, the cloning operation does not have to wait for a backup lock on the donor. See Section 5.6.7.4, “Cloning and Concurrent DDL”.

- `clone_donor_timeout_after_network_failure`

Command-Line Format	`--clone-donor-timeout-after-network-failure`
Introduced	8.0.24
System Variable	`clone_donor_timeout_after_network_failure`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`5`
Minimum Value	`0`
Maximum Value	`30`
Unit	minutes

Defines the amount of time in minutes the donor allows for the recipient to reconnect and restart a cloning operation after a network failure. For more information, see Section 5.6.7.9, “Remote Cloning Operation Failure Handling”.

This variable is set on the donor MySQL server instance. Setting it on the recipient MySQL server instance has no effect.

- `clone_enable_compression`

Command-Line Format	`--clone-enable-compression`
Introduced	8.0.17
System Variable	`clone_enable_compression`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Boolean
Default Value	`OFF`

Enables compression of data at the network layer during a remote cloning operation. Compression saves network bandwidth at the cost of CPU. Enabling compression may improve the data transfer rate. This setting is only applied on the recipient MySQL server instance.

- `clone_max_concurrency`

Command-Line Format	`--clone-max-concurrency`
Introduced	8.0.17
System Variable	`clone_max_concurrency`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`16`
Minimum Value	`1`
Maximum Value	`128`
Unit	threads

Defines the maximum number of concurrent threads for a remote cloning operation. The default value is 16. A greater number of threads can improve cloning performance but also reduces the number of permitted simultaneous client connections, which can affect the performance of existing client connections. This setting is only applied on the recipient MySQL server instance.

If `clone_autotune_concurrency` is enabled (the default), `clone_max_concurrency` is the maximum number of threads that can be dynamically spawned for a remote cloning operation. If `clone_autotune_concurrency` is disabled, `clone_max_concurrency` defines the number of threads spawned for a remote cloning operation.

A minimum data transfer rate of 1 mebibyte (MiB) per thread is recommended for remote cloning operations. The data transfer rate for a remote cloning operation is controlled by the `clone_max_data_bandwidth` variable.

- `clone_max_data_bandwidth`

Command-Line Format	`--clone-max-data-bandwidth`
Introduced	8.0.17
System Variable	`clone_max_data_bandwidth`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`1048576`
Unit	miB/second

Defines the maximum data transfer rate in mebibytes (MiB) per second for a remote cloning operation. This variable helps manage the performance impact of a cloning operation. A limit should be set only when donor disk I/O bandwidth is saturated, affecting performance. A value of 0 means

"unlimited", which permits cloning operations to run at the highest possible data transfer rate. This setting is only applicable to the recipient MySQL server instance.

The minimum data transfer rate is 1 MiB per second, per thread. For example, if there are 8 threads, the minimum transfer rate is 8 MiB per second. The `clone_max_concurrency` variable controls the maximum number threads spawned for a remote cloning operation.

The requested data transfer rate specified by `clone_max_data_bandwidth` may differ from the actual data transfer rate reported by the `DATA_SPEED` column in the `performance_schema.clone_progress` table. If your cloning operation is not achieving the desired data transfer rate and you have available bandwidth, check I/O usage on the recipient and donor. If there is underutilized bandwidth, I/O is the next mostly likely bottleneck.

- `clone_max_network_bandwidth`

Command-Line Format	`--clone-max-network-bandwidth`
Introduced	8.0.17
System Variable	`clone_max_network_bandwidth`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`1048576`
Unit	miB/second

Specifies the maximum approximate network transfer rate in mebibytes (MiB) per second for a remote cloning operation. This variable can be used to manage the performance impact of a cloning operation on network bandwidth. It should be set only when network bandwidth is saturated, affecting performance on the donor instance. A value of 0 means "unlimited", which permits cloning at the highest possible data transfer rate over the network, providing the best performance. This setting is only applicable to the recipient MySQL server instance.

- `clone_ssl_ca`

Command-Line Format	`--clone-ssl-ca=file_name`
Introduced	8.0.14
System Variable	`clone_ssl_ca`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	File name
Default Value	`empty string`

Specifies the path to the certificate authority (CA) file. Used to configure an encrypted connection for a remote cloning operation. This setting configured on the recipient and used when connecting to the donor.

- `clone_ssl_cert`

Command-Line Format	`--clone-ssl-cert=file_name`

Introduced	8.0.14
System Variable	clone_ssl_cert
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	File name
Default Value	empty string

Specifies the path to the public key certificate. Used to configure an encrypted connection for a remote cloning operation. This setting configured on the recipient and used when connecting to the donor.

- clone_ssl_key

Command-Line Format	--clone-ssl-key=file_name
Introduced	8.0.14
System Variable	clone_ssl_key
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	File name
Default Value	empty string

Specifies the path to the private key file. Used to configure an encrypted connection for a remote cloning operation. This setting configured on the recipient and used when connecting to the donor.

- clone_valid_donor_list

Command-Line Format	--clone-valid-donor-list=value
Introduced	8.0.17
System Variable	clone_valid_donor_list
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	NULL

Defines valid donor host addresses for remote cloning operations. This setting is applied on the recipient MySQL server instance. A comma-separated list of values is permitted in the following format: "HOST1:PORT1,HOST2:PORT2,HOST3:PORT3". Spaces are not permitted.

The clone_valid_donor_list variable adds a layer of security by providing control over the sources of cloned data. The privilege required to configure clone_valid_donor_list is different from the privilege required to execute remote cloning operations, which permits assigning those responsibilities to different roles. Configuring clone_valid_donor_list requires the SYSTEM_VARIABLES_ADMIN privilege, whereas executing a remote cloning operation requires the CLONE_ADMIN privilege.

Internet Protocol version 6 (IPv6) address format is not supported. Internet Protocol version 6 (IPv6) address format is not supported. An alias to the IPv6 address can be used instead. An IPv4 address can be used as is.

5.6.7.14 Clone Plugin Limitations

The clone plugin is subject to these limitations:

- Prior to MySQL 8.0.27, DDL on the donor and recipient, including `TRUNCATE TABLE`, is not permitted during a cloning operation. This limitation should be considered when selecting data sources. A workaround is to use dedicated donor instances, which can accommodate DDL operations being blocked while data is cloned. Concurrent DML is permitted.

  From MySQL 8.0.27, concurrent DDL is permitted on the donor by default. Support for concurrent DDL on the donor is controlled by the `clone_block_ddl` variable. See Section 5.6.7.4, “Cloning and Concurrent DDL”.

- An instance cannot be cloned from a different MySQL server version or release. The donor and recipient must have exactly the same MySQL server version and release. For example, you cannot clone between MySQL 5.7 and MySQL 8.0, or between MySQL 8.0.19 and MySQL 8.0.20. The clone plugin is only supported in MySQL 8.0.17 and higher.

- Cloning from a donor MySQL server instance to a hotfix MySQL server instance of the same version and release is only supported with MySQL 8.0.26 and higher.

- Only a single MySQL instance can be cloned at a time. Cloning multiple MySQL instances in a single cloning operation is not supported.

- The X Protocol port specified by `mysqlx_port` is not supported for remote cloning operations (when specifying the port number of the donor MySQL server instance in a `CLONE INSTANCE` statement).

- The clone plugin does not support cloning of MySQL server configurations. The recipient MySQL server instance retains its configuration, including persisted system variable settings (see Section 5.1.9.3, “Persisted System Variables”.)

- The clone plugin does not support cloning of binary logs.

- The clone plugin only clones data stored in `InnoDB`. Other storage engine data is not cloned. `MyISAM` and `CSV` tables stored in any schema including the `sys` schema are cloned as empty tables.

- Connecting to the donor MySQL server instance through MySQL Router is not supported.

- Local cloning operations do not support cloning of general tablespaces that were created with an absolute path. A cloned tablespace file with the same path as the source tablespace file would cause a conflict.

5.6.8 The Keyring Proxy Bridge Plugin

MySQL Keyring originally implemented keystore capabilities using server plugins, but began transitioning to use the component infrastructure in MySQL 8.0.24. The transition includes revising the underlying implementation of keyring plugins to use the component infrastructure. This is facilitated using the plugin named `daemon_keyring_proxy_plugin` that acts as a bridge between the plugin and component service APIs, and enables keyring plugins to continue to be used with no change to user-visible characteristics.

`daemon_keyring_proxy_plugin` is built in and nothing need be done to install or enable it.

5.6.9 MySQL Plugin Services

MySQL server plugins have access to server “plugin services.” The plugin services interface complements the plugin API by exposing server functionality that plugins can call. For developer information about writing plugin services, see MySQL Services for Plugins. The following sections describe plugin services available at the SQL and C-language levels.

5.6.9.1 The Locking Service

MySQL distributions provide a locking interface that is accessible at two levels:

- At the SQL level, as a set of loadable functions that each map onto calls to the service routines.
- As a C language interface, callable as a plugin service from server plugins or loadable functions.

For general information about plugin services, see Section 5.6.9, “MySQL Plugin Services”. For general information about loadable functions, see Adding a Loadable Function.

The locking interface has these characteristics:

- Locks have three attributes: Lock namespace, lock name, and lock mode:
  - Locks are identified by the combination of namespace and lock name. The namespace enables different applications to use the same lock names without colliding by creating locks in separate namespaces. For example, if applications A and B use namespaces of `ns1` and `ns2`, respectively, each application can use lock names `lock1` and `lock2` without interfering with the other application.
  - A lock mode is either read or write. Read locks are shared: If a session has a read lock on a given lock identifier, other sessions can acquire a read lock on the same identifier. Write locks are exclusive: If a session has a write lock on a given lock identifier, other sessions cannot acquire a read or write lock on the same identifier.
- Namespace and lock names must be non-`NULL`, nonempty, and have a maximum length of 64 characters. A namespace or lock name specified as `NULL`, the empty string, or a string longer than 64 characters results in an `ER_LOCKING_SERVICE_WRONG_NAME` error.
- The locking interface treats namespace and lock names as binary strings, so comparisons are case-sensitive.
- The locking interface provides functions to acquire locks and release locks. No special privilege is required to call these functions. Privilege checking is the responsibility of the calling application.
- Locks can be waited for if not immediately available. Lock acquisition calls take an integer timeout value that indicates how many seconds to wait to acquire locks before giving up. If the timeout is reached without successful lock acquisition, an `ER_LOCKING_SERVICE_TIMEOUT` error occurs. If the timeout is 0, there is no waiting and the call produces an error if locks cannot be acquired immediately.
- The locking interface detects deadlock between lock-acquisition calls in different sessions. In this case, the locking service chooses a caller and terminates its lock-acquisition request with an `ER_LOCKING_SERVICE_DEADLOCK` error. This error does not cause transactions to roll back. To choose a session in case of deadlock, the locking service prefers sessions that hold read locks over sessions that hold write locks.
- A session can acquire multiple locks with a single lock-acquisition call. For a given call, lock acquisition is atomic: The call succeeds if all locks are acquired. If acquisition of any lock fails, the call acquires no locks and fails, typically with an `ER_LOCKING_SERVICE_TIMEOUT` or `ER_LOCKING_SERVICE_DEADLOCK` error.
- A session can acquire multiple locks for the same lock identifier (namespace and lock name combination). These lock instances can be read locks, write locks, or a mix of both.
- Locks acquired within a session are released explicitly by calling a release-locks function, or implicitly when the session terminates (either normally or abnormally). Locks are not released when transactions commit or roll back.
- Within a session, all locks for a given namespace when released are released together.

The interface provided by the locking service is distinct from that provided by `GET_LOCK()` and related SQL functions (see Section 12.15, “Locking Functions”). For example, `GET_LOCK()` does not implement namespaces and provides only exclusive locks, not distinct read and write locks.

The Locking Service C Interface

This section describes how to use the locking service C language interface. To use the function interface instead, see The Locking Service Function Interface For general characteristics of the locking service interface, see Section 5.6.9.1, “The Locking Service”. For general information about plugin services, see Section 5.6.9, “MySQL Plugin Services”.

Source files that use the locking service should include this header file:

```
#include <mysql/service_locking.h>
```

To acquire one or more locks, call this function:

```
int mysql_acquire_locking_service_locks(MYSQL_THD opaque_thd,
                                        const char* lock_namespace,
                                        const char**lock_names,
                                        size_t lock_num,
                                        enum enum_locking_service_lock_type lock_type,
                                        unsigned long lock_timeout);
```

The arguments have these meanings:

- `opaque_thd`: A thread handle. If specified as `NULL`, the handle for the current thread is used.
- `lock_namespace`: A null-terminated string that indicates the lock namespace.
- `lock_names`: An array of null-terminated strings that provides the names of the locks to acquire.
- `lock_num`: The number of names in the `lock_names` array.
- `lock_type`: The lock mode, either `LOCKING_SERVICE_READ` or `LOCKING_SERVICE_WRITE` to acquire read locks or write locks, respectively.
- `lock_timeout`: An integer number of seconds to wait to acquire the locks before giving up.

To release locks acquired for a given namespace, call this function:

```
int mysql_release_locking_service_locks(MYSQL_THD opaque_thd,
                                        const char* lock_namespace);
```

The arguments have these meanings:

- `opaque_thd`: A thread handle. If specified as `NULL`, the handle for the current thread is used.
- `lock_namespace`: A null-terminated string that indicates the lock namespace.

Locks acquired or waited for by the locking service can be monitored at the SQL level using the Performance Schema. For details, see Locking Service Monitoring.

The Locking Service Function Interface

This section describes how to use the locking service interface provided by its loadable functions. To use the C language interface instead, see The Locking Service C Interface For general characteristics of the locking service interface, see Section 5.6.9.1, “The Locking Service”. For general information about loadable functions, see Adding a Loadable Function.

- Installing or Uninstalling the Locking Service Function Interface
- Using the Locking Service Function Interface
- Locking Service Monitoring
- Locking Service Interface Function Reference

Installing or Uninstalling the Locking Service Function Interface

The locking service routines described in The Locking Service C Interface need not be installed because they are built into the server. The same is not true of the loadable functions that map onto

calls to the service routines: The functions must be installed before use. This section describes how to do that. For general information about loadable function installation, see Section 5.7.1, “Installing and Uninstalling Loadable Functions”.

The locking service functions are implemented in a plugin library file located in the directory named by the `plugin_dir` system variable. The file base name is `locking_service`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To install the locking service functions, use the `CREATE FUNCTION` statement, adjusting the `.so` suffix for your platform as necessary:

```
CREATE FUNCTION service_get_read_locks RETURNS INT
  SONAME 'locking_service.so';
CREATE FUNCTION service_get_write_locks RETURNS INT
  SONAME 'locking_service.so';
CREATE FUNCTION service_release_locks RETURNS INT
  SONAME 'locking_service.so';
```

If the functions are used on a replication source server, install them on all replica servers as well to avoid replication problems.

Once installed, the functions remain installed until uninstalled. To remove them, use the `DROP FUNCTION` statement:

```
DROP FUNCTION service_get_read_locks;
DROP FUNCTION service_get_write_locks;
DROP FUNCTION service_release_locks;
```

Using the Locking Service Function Interface

Before using the locking service functions, install them according to the instructions provided at Installing or Uninstalling the Locking Service Function Interface.

To acquire one or more read locks, call this function:

```
mysql> SELECT service_get_read_locks('mynamespace', 'rlock1', 'rlock2', 10);
+-------------------------------------------------------------+
| service_get_read_locks('mynamespace', 'rlock1', 'rlock2', 10) |
+-------------------------------------------------------------+
|                                                           1 |
+-------------------------------------------------------------+
```

The first argument is the lock namespace. The final argument is an integer timeout indicating how many seconds to wait to acquire the locks before giving up. The arguments in between are the lock names.

For the example just shown, the function acquires locks with lock identifiers `(mynamespace, rlock1)` and `(mynamespace, rlock2)`.

To acquire write locks rather than read locks, call this function:

```
mysql> SELECT service_get_write_locks('mynamespace', 'wlock1', 'wlock2', 10);
+--------------------------------------------------------------+
| service_get_write_locks('mynamespace', 'wlock1', 'wlock2', 10) |
+--------------------------------------------------------------+
|                                                            1 |
+--------------------------------------------------------------+
```

In this case, the lock identifiers are `(mynamespace, wlock1)` and `(mynamespace, wlock2)`.

To release all locks for a namespace, use this function:

```
mysql> SELECT service_release_locks('mynamespace');
+--------------------------------------+
| service_release_locks('mynamespace') |
+--------------------------------------+
|                                    1 |
+--------------------------------------+
```

Each locking function returns nonzero for success. If the function fails, an error occurs. For example, the following error occurs because lock names cannot be empty:

```
mysql> SELECT service_get_read_locks('mynamespace', '', 10);
ERROR 3131 (42000): Incorrect locking service lock name ''.
```

A session can acquire multiple locks for the same lock identifier. As long as a different session does not have a write lock for an identifier, the session can acquire any number of read or write locks. Each lock request for the identifier acquires a new lock. The following statements acquire three write locks with the same identifier, then three read locks for the same identifier:

```
SELECT service_get_write_locks('ns', 'lock1', 'lock1', 'lock1', 0);
SELECT service_get_read_locks('ns', 'lock1', 'lock1', 'lock1', 0);
```

If you examine the Performance Schema `metadata_locks` table at this point, you should find that the session holds six distinct locks with the same `(ns, lock1)` identifier. (For details, see Locking Service Monitoring.)

Because the session holds at least one write lock on `(ns, lock1)`, no other session can acquire a lock for it, either read or write. If the session held only read locks for the identifier, other sessions could acquire read locks for it, but not write locks.

Locks for a single lock-acquisition call are acquired atomically, but atomicity does not hold across calls. Thus, for a statement such as the following, where `service_get_write_locks()` is called once per row of the result set, atomicity holds for each individual call, but not for the statement as a whole:

```
SELECT service_get_write_locks('ns', 'lock1', 'lock2', 0) FROM t1 WHERE ... ;
```

Caution

Because the locking service returns a separate lock for each successful request for a given lock identifier, it is possible for a single statement to acquire a large number of locks. For example:

```
INSERT INTO ... SELECT service_get_write_locks('ns', t1.col_name, 0) FROM t1;
```

These types of statements may have certain adverse effects. For example, if the statement fails part way through and rolls back, locks acquired up to the point of failure still exist. If the intent is for there to be a correspondence between rows inserted and locks acquired, that intent is not satisfied. Also, if it is important that locks are granted in a certain order, be aware that result set order may differ depending on which execution plan the optimizer chooses. For these reasons, it may be best to limit applications to a single lock-acquisition call per statement.

Locking Service Monitoring

The locking service is implemented using the MySQL Server metadata locks framework, so you monitor locking service locks acquired or waited for by examining the Performance Schema `metadata_locks` table.

First, enable the metadata lock instrument:

```
mysql> UPDATE performance_schema.setup_instruments SET ENABLED = 'YES'
    -> WHERE NAME = 'wait/lock/metadata/sql/mdl';
```

Then acquire some locks and check the contents of the `metadata_locks` table:

```
mysql> SELECT service_get_write_locks('mynamespace', 'lock1', 0);
+-----------------------------------------------------+
| service_get_write_locks('mynamespace', 'lock1', 0) |
+-----------------------------------------------------+
|                                                   1 |
+-----------------------------------------------------+
mysql> SELECT service_get_read_locks('mynamespace', 'lock2', 0);
```

```
+-------------------------------------------------+
| service_get_read_locks('mynamespace', 'lock2', 0) |
+-------------------------------------------------+
|                                               1 |
+-------------------------------------------------+
mysql> SELECT OBJECT_TYPE, OBJECT_SCHEMA, OBJECT_NAME, LOCK_TYPE, LOCK_STATUS
    -> FROM performance_schema.metadata_locks
    -> WHERE OBJECT_TYPE = 'LOCKING SERVICE'\G
*************************** 1. row ***************************
  OBJECT_TYPE: LOCKING SERVICE
OBJECT_SCHEMA: mynamespace
  OBJECT_NAME: lock1
    LOCK_TYPE: EXCLUSIVE
  LOCK_STATUS: GRANTED
*************************** 2. row ***************************
  OBJECT_TYPE: LOCKING SERVICE
OBJECT_SCHEMA: mynamespace
  OBJECT_NAME: lock2
    LOCK_TYPE: SHARED
  LOCK_STATUS: GRANTED
```

Locking service locks have an `OBJECT_TYPE` value of `LOCKING SERVICE`. This is distinct from, for example, locks acquired with the `GET_LOCK()` function, which have an `OBJECT_TYPE` of `USER LEVEL LOCK`.

The lock namespace, name, and mode appear in the `OBJECT_SCHEMA`, `OBJECT_NAME`, and `LOCK_TYPE` columns. Read and write locks have `LOCK_TYPE` values of `SHARED` and `EXCLUSIVE`, respectively.

The `LOCK_STATUS` value is `GRANTED` for an acquired lock, `PENDING` for a lock that is being waited for. You can expect to see `PENDING` if one session holds a write lock and another session is attempting to acquire a lock having the same identifier.

Locking Service Interface Function Reference

The SQL interface to the locking service implements the loadable functions described in this section. For usage examples, see Using the Locking Service Function Interface.

The functions share these characteristics:

- The return value is nonzero for success. Otherwise, an error occurs.
- Namespace and lock names must be non-`NULL`, nonempty, and have a maximum length of 64 characters.
- Timeout values must be integers indicating how many seconds to wait to acquire locks before giving up with an error. If the timeout is 0, there is no waiting and the function produces an error if locks cannot be acquired immediately.

These locking service functions are available:

- `service_get_read_locks(`*`namespace, lock_name`*`[, `*`lock_name`*`] ..., `*`timeout`*`)`

  Acquires one or more read (shared) locks in the given namespace using the given lock names, timing out with an error if the locks are not acquired within the given timeout value.

- `service_get_write_locks(`*`namespace, lock_name`*`[, `*`lock_name`*`] ..., `*`timeout`*`)`

  Acquires one or more write (exclusive) locks in the given namespace using the given lock names, timing out with an error if the locks are not acquired within the given timeout value.

- `service_release_locks(`*`namespace`*`)`

  For the given namespace, releases all locks that were acquired within the current session using `service_get_read_locks()` and `service_get_write_locks()`.

It is not an error for there to be no locks in the namespace.

5.6.9.2 The Keyring Service

MySQL Server supports a keyring service that enables internal components and plugins to securely store sensitive information for later retrieval. MySQL distributions provide a keyring interface that is accessible at two levels:

- At the SQL level, as a set of loadable functions that each map onto calls to the service routines.
- As a C language interface, callable as a plugin service from server plugins or loadable functions.

This section describes how to use the keyring service functions to store, retrieve, and remove keys in the MySQL keyring keystore. For information about the SQL interface that uses functions, Section 6.4.4.15, “General-Purpose Keyring Key-Management Functions”. For general keyring information, see Section 6.4.4, “The MySQL Keyring”.

The keyring service uses whatever underlying keyring plugin is enabled, if any. If no keyring plugin is enabled, keyring service calls fail.

A “record” in the keystore consists of data (the key itself) and a unique identifier through which the key is accessed. The identifier has two parts:

- `key_id`: The key ID or name. `key_id` values that begin with `mysql_` are reserved by MySQL Server.
- `user_id`: The session effective user ID. If there is no user context, this value can be `NULL`. The value need not actually be a “user”; the meaning depends on the application.

  Functions that implement the keyring function interface pass the value of `CURRENT_USER()` as the `user_id` value to keyring service functions.

The keyring service functions have these characteristics in common:

- Each function returns 0 for success, 1 for failure.
- The `key_id` and `user_id` arguments form a unique combination indicating which key in the keyring to use.
- The `key_type` argument provides additional information about the key, such as its encryption method or intended use.
- Keyring service functions treat key IDs, user names, types, and values as binary strings, so comparisons are case-sensitive. For example, IDs of `MyKey` and `mykey` refer to different keys.

These keyring service functions are available:

- `my_key_fetch()`

  Deobfuscates and retrieves a key from the keyring, along with its type. The function allocates the memory for the buffers used to store the returned key and key type. The caller should zero or obfuscate the memory when it is no longer needed, then free it.

  Syntax:

  ```
  bool my_key_fetch(const char *key_id, const char **key_type,
                    const char* user_id, void **key, size_t *key_len)
  ```

  Arguments:

  - `key_id`, `user_id`: Null-terminated strings that as a pair form a unique identifier indicating which key to fetch.

- `key_type`: The address of a buffer pointer. The function stores into it a pointer to a null-terminated string that provides additional information about the key (stored when the key was added).
- `key`: The address of a buffer pointer. The function stores into it a pointer to the buffer containing the fetched key data.
- `key_len`: The address of a variable into which the function stores the size in bytes of the `*key` buffer.

Return value:

Returns 0 for success, 1 for failure.

- `my_key_generate()`

  Generates a new random key of a given type and length and stores it in the keyring. The key has a length of `key_len` and is associated with the identifier formed from `key_id` and `user_id`. The type and length values must be consistent with the values supported by the underlying keyring plugin. See Section 6.4.4.13, “Supported Keyring Key Types and Lengths”.

  Syntax:

```
bool my_key_generate(const char *key_id, const char *key_type,
                     const char *user_id, size_t key_len)
```

  Arguments:

  - `key_id`, `user_id`: Null-terminated strings that as a pair form a unique identifier for the key to be generated.
  - `key_type`: A null-terminated string that provides additional information about the key.
  - `key_len`: The size in bytes of the key to be generated.

  Return value:

  Returns 0 for success, 1 for failure.

- `my_key_remove()`

  Removes a key from the keyring.

  Syntax:

```
bool my_key_remove(const char *key_id, const char* user_id)
```

  Arguments:

  - `key_id`, `user_id`: Null-terminated strings that as a pair form a unique identifier for the key to be removed.

  Return value:

  Returns 0 for success, 1 for failure.

- `my_key_store()`

  Obfuscates and stores a key in the keyring.

  Syntax:

```
bool my_key_store(const char *key_id, const char *key_type,
```

```
                const char* user_id, void *key, size_t key_len)
```

Arguments:

- `key_id`, `user_id`: Null-terminated strings that as a pair form a unique identifier for the key to be stored.
- `key_type`: A null-terminated string that provides additional information about the key.
- `key`: The buffer containing the key data to be stored.
- `key_len`: The size in bytes of the `key` buffer.

Return value:

Returns 0 for success, 1 for failure.

5.7 MySQL Server Loadable Functions

MySQL supports loadable functions, that is, functions that are not built in but can be loaded at runtime (either during startup or later) to extend server capabilities, or unloaded to remove capabilities. For a table describing the available loadable functions, see Section 12.2, “Loadable Function Reference”. Loadable functions contrast with built-in (native) functions, which are implemented as part of the server and are always available; for a table, see Section 12.1, “Built-In Function and Operator Reference”.

Note

Loadable functions previously were known as user-defined functions (UDFs). That terminology was something of a misnomer because “user-defined” also can apply to other types of functions, such as stored functions (a type of stored object written using SQL) and native functions added by modifying the server source code.

MySQL distributions include loadable functions that implement, in whole or in part, these server capabilities:

- Group Replication enables you to create a highly available distributed MySQL service across a group of MySQL server instances, with data consistency, conflict detection and resolution, and group membership services all built-in. See Chapter 18, *Group Replication*.
- MySQL Enterprise Edition includes functions that perform encryption operations based on the OpenSSL library. See Section 6.6, “MySQL Enterprise Encryption”.
- MySQL Enterprise Edition includes functions that provide an SQL-level API for masking and de-identification operations. See Section 6.5.1, “MySQL Enterprise Data Masking and De-Identification Elements”.
- MySQL Enterprise Edition includes audit logging for monitoring and logging of connection and query activity. See Section 6.4.5, “MySQL Enterprise Audit”, and Section 6.4.6, “The Audit Message Component”.
- MySQL Enterprise Edition includes a firewall capability that implements an application-level firewall to enable database administrators to permit or deny SQL statement execution based on matching against patterns for accepted statement. See Section 6.4.7, “MySQL Enterprise Firewall”.
- A query rewriter examines statements received by MySQL Server and possibly rewrites them before the server executes them. See Section 5.6.4, “The Rewriter Query Rewrite Plugin”
- Version Tokens enables creation of and synchronization around server tokens that applications can use to prevent accessing incorrect or out-of-date data. See Section 5.6.6, “Version Tokens”.
- The MySQL Keyring provides secure storage for sensitive information. See Section 6.4.4, “The MySQL Keyring”.

- A locking service provides a locking interface for application use. See Section 5.6.9.1, “The Locking Service”.
- A function provides access to query attributes. See Section 9.6, “Query Attributes”.

The following sections describe how to install and uninstall loadable functions, and how to determine at runtime which loadable functions are installed and obtain information about them.

In some cases, a loadable function is loaded by installing the component that implements the function, rather than by loading the function directly. For details about a particular loadable function, see the installation instructions for the server feature that includes it.

For information about writing loadable functions, see Adding Functions to MySQL.

5.7.1 Installing and Uninstalling Loadable Functions

Loadable functions, as the name implies, must be loaded into the server before they can be used. MySQL supports automatic function loading during server startup and manual loading thereafter.

While a loadable function is loaded, information about it is available as described in Section 5.7.2, “Obtaining Information About Loadable Functions”.

- Installing Loadable Functions
- Uninstalling Loadable Functions
- Reinstalling or Upgrading Loadable Functions

Installing Loadable Functions

To load a loadable function manually, use the `CREATE FUNCTION` statement. For example:

```
CREATE FUNCTION metaphon
  RETURNS STRING
  SONAME 'udf_example.so';
```

The file base name depends on your platform. Common suffixes are `.so` for Unix and Unix-like systems, `.dll` for Windows.

`CREATE FUNCTION` has these effects:

- It loads the function into the server to make it available immediately.
- It registers the function in the `mysql.func` system table to make it persistent across server restarts. For this reason, `CREATE FUNCTION` requires the `INSERT` privilege for the `mysql` system database.
- It adds the function to the Performance Schema `user_defined_functions` table that provides runtime information about installed loadable functions. See Section 5.7.2, “Obtaining Information About Loadable Functions”.

Automatic loading of loadable functions occurs during the normal server startup sequence:

- Functions registered in the `mysql.func` table are installed.
- Components or plugins that are installed at startup may automatically install related functions.
- Automatic function installation adds the functions to the Performance Schema `user_defined_functions` table that provides runtime information about installed functions.

If the server is started with the `--skip-grant-tables` option, functions registered in the `mysql.func` table are not loaded and are unavailable. This does not apply to functions installed automatically by a component or plugin.

Uninstalling Loadable Functions

To remove a loadable function, use the `DROP FUNCTION` statement. For example:

```
DROP FUNCTION metaphon;
```

`DROP FUNCTION` has these effects:

- It unloads the function to make it unavailable.
- It removes the function from the `mysql.func` system table. For this reason, `DROP FUNCTION` requires the `DELETE` privilege for the `mysql` system database. With the function no longer registered in the `mysql.func` table, the server does not load the function during subsequent restarts.
- It removes the function from the Performance Schema `user_defined_functions` table that provides runtime information about installed loadable functions.

`DROP FUNCTION` cannot be used to drop a loadable function that is installed automatically by components or plugins rather than by using `CREATE FUNCTION`. Such a function is also dropped automatically, when the component or plugin that installed it is uninstalled.

Reinstalling or Upgrading Loadable Functions

To reinstall or upgrade the shared library associated with a loadable function, issue a `DROP FUNCTION` statement, upgrade the shared library, and then issue a `CREATE FUNCTION` statement. If you upgrade the shared library first and then use `DROP FUNCTION`, the server may unexpectedly shut down.

5.7.2 Obtaining Information About Loadable Functions

The Performance Schema `user_defined_functions` table contains information about the currently installed loadable functions:

```
SELECT * FROM performance_schema.user_defined_functions;
```

The `mysql.func` system table also lists installed loadable functions, but only those installed using `CREATE FUNCTION`. The `user_defined_functions` table lists loadable functions installed using `CREATE FUNCTION` as well as loadable functions installed automatically by components or plugins. This difference makes `user_defined_functions` preferable to `mysql.func` for checking which loadable functions are installed. See Section 27.12.21.9, “The user_defined_functions Table”.

5.8 Running Multiple MySQL Instances on One Machine

In some cases, you might want to run multiple instances of MySQL on a single machine. You might want to test a new MySQL release while leaving an existing production setup undisturbed. Or you might want to give different users access to different `mysqld` servers that they manage themselves. (For example, you might be an Internet Service Provider that wants to provide independent MySQL installations for different customers.)

It is possible to use a different MySQL server binary per instance, or use the same binary for multiple instances, or any combination of the two approaches. For example, you might run a server from MySQL 5.7 and one from MySQL 8.0, to see how different versions handle a given workload. Or you might run multiple instances of the current production version, each managing a different set of databases.

Whether or not you use distinct server binaries, each instance that you run must be configured with unique values for several operating parameters. This eliminates the potential for conflict between instances. Parameters can be set on the command line, in option files, or by setting environment variables. See Section 4.2.2, “Specifying Program Options”. To see the values used by a given instance, connect to it and execute a `SHOW VARIABLES` statement.

The primary resource managed by a MySQL instance is the data directory. Each instance should use a different data directory, the location of which is specified using the `--datadir=dir_name` option. For methods of configuring each instance with its own data directory, and warnings about the dangers of failing to do so, see Section 5.8.1, “Setting Up Multiple Data Directories”.

In addition to using different data directories, several other options must have different values for each server instance:

- `--port=port_num`

  `--port` controls the port number for TCP/IP connections. Alternatively, if the host has multiple network addresses, you can set the `bind_address` system variable to cause each server to listen to a different address.

- `--socket={file_name|pipe_name}`

  `--socket` controls the Unix socket file path on Unix or the named-pipe name on Windows. On Windows, it is necessary to specify distinct pipe names only for those servers configured to permit named-pipe connections.

- `--shared-memory-base-name=name`

  This option is used only on Windows. It designates the shared-memory name used by a Windows server to permit clients to connect using shared memory. It is necessary to specify distinct shared-memory names only for those servers configured to permit shared-memory connections.

- `--pid-file=file_name`

  This option indicates the path name of the file in which the server writes its process ID.

If you use the following log file options, their values must differ for each server:

- `--general_log_file=file_name`
- `--log-bin[=file_name]`
- `--slow_query_log_file=file_name`
- `--log-error[=file_name]`

For further discussion of log file options, see Section 5.4, “MySQL Server Logs”.

To achieve better performance, you can specify the following option differently for each server, to spread the load between several physical disks:

- `--tmpdir=dir_name`

Having different temporary directories also makes it easier to determine which MySQL server created any given temporary file.

If you have multiple MySQL installations in different locations, you can specify the base directory for each installation with the `--basedir=dir_name` option. This causes each instance to automatically use a different data directory, log files, and PID file because the default for each of those parameters is relative to the base directory. In that case, the only other options you need to specify are the `--socket` and `--port` options. Suppose that you install different versions of MySQL using `tar` file binary distributions. These install in different locations, so you can start the server for each installation using the command `bin/mysqld_safe` under its corresponding base directory. `mysqld_safe` determines the proper `--basedir` option to pass to `mysqld`, and you need specify only the `--socket` and `--port` options to `mysqld_safe`.

As discussed in the following sections, it is possible to start additional servers by specifying appropriate command options or by setting environment variables. However, if you need to run multiple servers

on a more permanent basis, it is more convenient to use option files to specify for each server those option values that must be unique to it. The `--defaults-file` option is useful for this purpose.

5.8.1 Setting Up Multiple Data Directories

Each MySQL Instance on a machine should have its own data directory. The location is specified using the `--datadir=dir_name` option.

There are different methods of setting up a data directory for a new instance:

- Create a new data directory.
- Copy an existing data directory.

The following discussion provides more detail about each method.

Warning

Normally, you should never have two servers that update data in the same databases. This may lead to unpleasant surprises if your operating system does not support fault-free system locking. If (despite this warning) you run multiple servers using the same data directory and they have logging enabled, you must use the appropriate options to specify log file names that are unique to each server. Otherwise, the servers try to log to the same files.

Even when the preceding precautions are observed, this kind of setup works only with `MyISAM` and `MERGE` tables, and not with any of the other storage engines. Also, this warning against sharing a data directory among servers always applies in an NFS environment. Permitting multiple MySQL servers to access a common data directory over NFS is a *very bad idea*. The primary problem is that NFS is the speed bottleneck. It is not meant for such use. Another risk with NFS is that you must devise a way to ensure that two or more servers do not interfere with each other. Usually NFS file locking is handled by the `lockd` daemon, but at the moment there is no platform that performs locking 100% reliably in every situation.

Create a New Data Directory

With this method, the data directory is in the same state as when you first install MySQL, and has the default set of MySQL accounts and no user data.

On Unix, initialize the data directory. See Section 2.9, “Postinstallation Setup and Testing”.

On Windows, the data directory is included in the MySQL distribution:

- MySQL Zip archive distributions for Windows contain an unmodified data directory. You can unpack such a distribution into a temporary location, then copy it `data` directory to where you are setting up the new instance.
- Windows MSI package installers create and set up the data directory that the installed server uses, but also create a pristine “template” data directory named `data` under the installation directory. After an installation has been performed using an MSI package, the template data directory can be copied to set up additional MySQL instances.

Copy an Existing Data Directory

With this method, any MySQL accounts or user data present in the data directory are carried over to the new data directory.

1. Stop the existing MySQL instance using the data directory. This must be a clean shutdown so that the instance flushes any pending changes to disk.

2. Copy the data directory to the location where the new data directory should be.

3. Copy the `my.cnf` or `my.ini` option file used by the existing instance. This serves as a basis for the new instance.

4. Modify the new option file so that any pathnames referring to the original data directory refer to the new data directory. Also, modify any other options that must be unique per instance, such as the TCP/IP port number and the log files. For a list of parameters that must be unique per instance, see Section 5.8, "Running Multiple MySQL Instances on One Machine".

5. Start the new instance, telling it to use the new option file.

5.8.2 Running Multiple MySQL Instances on Windows

You can run multiple servers on Windows by starting them manually from the command line, each with appropriate operating parameters, or by installing several servers as Windows services and running them that way. General instructions for running MySQL from the command line or as a service are given in Section 2.3, "Installing MySQL on Microsoft Windows". The following sections describe how to start each server with different values for those options that must be unique per server, such as the data directory. These options are listed in Section 5.8, "Running Multiple MySQL Instances on One Machine".

5.8.2.1 Starting Multiple MySQL Instances at the Windows Command Line

The procedure for starting a single MySQL server manually from the command line is described in Section 2.3.4.6, "Starting MySQL from the Windows Command Line". To start multiple servers this way, you can specify the appropriate options on the command line or in an option file. It is more convenient to place the options in an option file, but it is necessary to make sure that each server gets its own set of options. To do this, create an option file for each server and tell the server the file name with a `--defaults-file` option when you run it.

Suppose that you want to run one instance of `mysqld` on port 3307 with a data directory of `C:\mydata1`, and another instance on port 3308 with a data directory of `C:\mydata2`. Use this procedure:

1. Make sure that each data directory exists, including its own copy of the `mysql` database that contains the grant tables.

2. Create two option files. For example, create one file named `C:\my-opts1.cnf` that looks like this:

   ```
   [mysqld]
   datadir = C:/mydata1
   port = 3307
   ```

   Create a second file named `C:\my-opts2.cnf` that looks like this:

   ```
   [mysqld]
   datadir = C:/mydata2
   port = 3308
   ```

3. Use the `--defaults-file` option to start each server with its own option file:

   ```
   C:\> C:\mysql\bin\mysqld --defaults-file=C:\my-opts1.cnf
   C:\> C:\mysql\bin\mysqld --defaults-file=C:\my-opts2.cnf
   ```

   Each server starts in the foreground (no new prompt appears until the server exits later), so you need to issue those two commands in separate console windows.

To shut down the servers, connect to each using the appropriate port number:

```
C:\> C:\mysql\bin\mysqladmin --port=3307 --host=127.0.0.1 --user=root --password shutdown
C:\> C:\mysql\bin\mysqladmin --port=3308 --host=127.0.0.1 --user=root --password shutdown
```