Extra Transactions

If a joining member has transactions in its GTID set that are not present on the existing members of the group, it is not allowed to complete the distributed recovery process, and cannot join the group. If a remote cloning operation was carried out, these transactions would be deleted and lost, because the data directory on the joining member is erased. If state transfer from a donor's binary log was carried out, these transactions could conflict with the group's transactions.

Extra transactions might be present on a member if an administrative transaction is carried out on the instance while Group Replication is stopped. To avoid introducing new transactions in that way, always set the value of the `sql_log_bin` system variable to `OFF` before issuing administrative statements, and back to `ON` afterwards:

```
SET SQL_LOG_BIN=0;
<administrator action>
SET SQL_LOG_BIN=1;
```

Setting this system variable to `OFF` means that the transactions that occur from that point until you set it back to `ON` are not written to the binary log and do not have GTIDs assigned to them.

If an extra transaction is present on a joining member, check the binary log for the affected server to see what the extra transaction actually contains. The safest method to reconcile the joining member's data and GTID set with the members currently in the group is to use MySQL's cloning functionality to transfer the content from a server in the group to the affected server. For instructions to do this, see Section 5.6.7.3, "Cloning Remote Data". If the transaction is required, rerun it after the member has successfully rejoined.

18.4.2 Group Replication Server States

The state of a Group Replication group member shows its current role in the group. The Performance Schema table `replication_group_members` shows the state for each member in a group. If the group is fully functional and all members are communicating properly, all members report the same state for all other members. However, a member that has left the group or is part of a network partition cannot report accurate information on the other servers. In this situation, the member does not attempt to guess the status of the other servers, and instead reports them as unreachable.

A group member can be in the following states:

`ONLINE`	The server is an active member of a group and in a fully functioning state. Other group members can connect to it, as can clients if applicable. A member is only fully synchronized with the group, and participating in it, when it is in the `ONLINE` state.
`RECOVERING`	The server has joined a group and is in the process of becoming an active member. Distributed recovery is currently taking place, where the member is receiving state transfer from a donor using a remote cloning operation or the donor's binary log. This state is For more information, see Section 18.5.4, "Distributed Recovery".
`OFFLINE`	The Group Replication plugin is loaded but the member does not belong to any group. This status may briefly occur while a member is joining or rejoining a group.
`ERROR`	The member is in an error state and is not functioning correctly as a group member. A member can enter error state either while applying transactions or during the recovery phase. A member in this state does not participate in the group's transactions. For more information on possible reasons for error states, see Section 18.7.7, "Responses to Failure Detection and Network Partitioning".

	Depending on the exit action set by `group_replication_exit_state_action`, the member is in read-only mode (`super_read_only=ON`) and could also be in offline mode (`offline_mode=ON`). Note that a server in offline mode following the `OFFLINE_MODE` exit action is displayed with `ERROR` status, not `OFFLINE`. A server with the exit action `ABORT_SERVER` shuts down and is removed from the view of the group. For more information, see Section 18.7.7.4, "Exit Action". While a member is joining or rejoining a replication group, its status can be displayed as `ERROR` before the group completes the compatibility checks and accepts it as a member.
`UNREACHABLE`	The local failure detector suspects that the member cannot be contacted, because the group's messages are timing out. This can happen if a member is disconnected involuntarily, for example. If you see this status for other servers, it can also mean that the member where you query this table is part of a partition, where a subset of the group's servers can contact each other but cannot contact the other servers in the group. For more information, see Section 18.7.8, "Handling a Network Partition and Loss of Quorum".

See Section 18.4.3, "The replication_group_members Table" for an example of the Performance Schema table contents.

18.4.3 The replication_group_members Table

The `performance_schema.replication_group_members` table is used for monitoring the status of the different server instances that are members of the group. The information in the table is updated whenever there is a view change, for example when the configuration of the group is dynamically changed when a new member joins. At that point, servers exchange some of their metadata to synchronize themselves and continue to cooperate together. The information is shared between all the server instances that are members of the replication group, so information on all the group members can be queried from any member. This table can be used to get a high level view of the state of a replication group, for example by issuing:

```
SELECT * FROM performance_schema.replication_group_members;
+---------------------------+--------------------------------------+-------------+-------------+-----------
| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST | MEMBER_PORT | MEMBER_STA
+---------------------------+--------------------------------------+-------------+-------------+-----------
| group_replication_applier | d391e9ee-2691-11ec-bf61-00059a3c7a00 | example1    |        4410 | ONLINE
| group_replication_applier | e059ce5c-2691-11ec-8632-00059a3c7a00 | example2    |        4420 | ONLINE
| group_replication_applier | ecd9ad06-2691-11ec-91c7-00059a3c7a00 | example3    |        4430 | ONLINE
+---------------------------+--------------------------------------+-------------+-------------+-----------
3 rows in set (0.0007 sec)
```

Based on this result we can see that the group consists of three members. Shown in the table is each member's `server_uuid`, as well as the member's host name and port number, which clients use to connect to it. The `MEMBER_STATE` column shows one of the Section 18.4.2, "Group Replication Server States", in this case it shows that all three members in this group are `ONLINE`, and the `MEMBER_ROLE` column shows that there are two secondaries, and a single primary. Therefore this group must be running in single-primary mode. The `MEMBER_VERSION` column can be useful when you are upgrading a group and are combining members running different MySQL versions. The `MEMBER_COMMUNICATION_STACK` column shows the communication stack used for the group.

For more information about the `MEMBER_HOST` value and its impact on the distributed recovery process, see Section 18.2.1.3, "User Credentials For Distributed Recovery".

18.4.4 The replication_group_member_stats Table

Each member in a replication group certifies and applies transactions received by the group. Statistics regarding the certifier and applier procedures are useful to understand how the applier queue is growing, how many conflicts have been found, how many transactions were checked, which transactions are committed everywhere, and so on.

The `performance_schema.replication_group_member_stats` table provides group-level information related to the certification process, and also statistics for the transactions received and originated by each individual member of the replication group. The information is shared between all the server instances that are members of the replication group, so information on all the group members can be queried from any member. Note that refreshing of statistics for remote members is controlled by the message period specified in the `group_replication_flow_control_period` option, so these can differ slightly from the locally collected statistics for the member where the query is made. To use this table to monitor a Group Replication member, issue the following statement:

```
mysql> SELECT * FROM performance_schema.replication_group_member_stats\G
```

Beginning with MySQL 8.0.19, you can also use the following statement:

```
mysql> TABLE performance_schema.replication_group_member_stats\G
```

These columns are important for monitoring the performance of the members connected in the group. Suppose that one of the group's members always reports a large number of transactions in its queue compared to other members. This means that the member is delayed and is not able to keep up to date with the other members of the group. Based on this information, you could decide to either remove the member from the group, or delay the processing of transactions on the other members of the group in order to reduce the number of queued transactions. This information can also help you to decide how to adjust the flow control of the Group Replication plugin, see Section 18.7.2, “Flow Control”.

18.5 Group Replication Operations

This section explains common operations for managing groups.

18.5.1 Configuring an Online Group

You can configure an online group while Group Replication is running by using a set of functions, which rely on a group action coordinator. These functions are installed by the Group Replication plugin in version 8.0.13 and higher. This section describes how changes are made to a running group, and the available functions.

Important

For the coordinator to be able to configure group wide actions on a running group, all members must be running MySQL 8.0.13 or higher and have the functions installed.

To use the functions, connect to a member of the running group and invoke the function with the `SELECT` statement. The Group Replication plugin processes the action and its parameters and the coordinator sends it to all members which are visible to the member where you invoked the function. If the action is accepted, all members execute the action and send a termination message when completed. Once all members declare the action as finished, the invoking member returns the result to the client.

When configuring a whole group, the distributed nature of the operations means that they interact with many processes of the Group Replication plugin, and therefore you should observe the following:

You can issue configuration operations everywhere. If you want to make member A the new primary you do not need to invoke the operation on member A. All operations are sent and executed in a coordinated way on all group members. Also, this distributed execution of an operation has a different ramification: if the invoking member dies, any already running configuration process continues to run

on other members. In the unlikely event that the invoking member dies, you can still use the monitoring features to ensure other members complete the operation successfully.

All members must be online. To simplify the migration or election processes and guarantee they are as fast as possible, the group must not contain any member currently in the distributed recovery process, otherwise the configuration action is rejected by the member where you issue the statement.

No members can join a group during a configuration change. Any member that attempts to join the group during a coordinated configuration change leaves the group and cancels its join process.

Only one configuration at once. A group which is executing a configuration change cannot accept any other group configuration change, because concurrent configuration operations could lead to member divergence.

All members must be running MySQL 8.0.13 or higher. Due to the distributed nature of the configuration actions, all members must recognize them in order to execute them. The operation is therefore rejected if any server running MySQL Server version 8.0.12 or lower is present in the group.

18.5.1.1 Changing a Group's Primary Member

This section explains how to change which member of a single-primary group is the primary. The function used to change a group's mode can be run on any member.

Changing which Member is Primary

Use the `group_replication_set_as_primary()` function to change which member is the primary in a single-primary group. The current primary becomes a read-only secondary, and the specified group member becomes the read-write primary. The function can be used on any member of a replication group running in single-primary mode. It replaces the usual primary election process, which is described in Section 18.1.3.1, “Single-Primary Mode”.

If a standard source to replica replication channel is running on the existing primary member in addition to the Group Replication channels, you must stop that replication channel before you can change the primary member. You can identify the current primary using the `MEMBER_ROLE` column in the Performance Schema table `replication_group_members`, or the `group_replication_primary_member` status variable.

If you invoke the function on a member running a MySQL Server version from 8.0.17, and all members are running MySQL Server version 8.0.17 or higher, you can only specify a new primary member that is running the lowest MySQL Server version in the group, based on the patch version. This safeguard is applied to ensure the group maintains compatibility with new functions. If any member is running a MySQL Server version between MySQL 8.0.13 and MySQL 8.0.16, this safeguard is not enforced for the group and you can specify any new primary member, but it is recommended to select a primary that is running the lowest MySQL Server version in the group.

Any uncommitted transactions that the group is waiting on must be committed, rolled back, or terminated before the operation can complete. Before MySQL 8.0.29, the function waits for all active transactions on the existing primary to end, including incoming transactions that are started after the function is used. From MySQL 8.0.29, you can specify a timeout from 0 seconds (immediately) up to 3600 seconds (60 minutes) for transactions that are running when you use the function. For the timeout to work, all members of the group must be at MySQL 8.0.29 or higher. There is no default setting for the timeout, so if you do not set it, there is no upper limit to the wait time, and new transactions can start during that time.

When the timeout expires, for any transactions that did not yet reach their commit phase, the client session is disconnected so that the transaction does not proceed. Transactions that reached their commit phase are allowed to complete. When you set a timeout, it also prevents new transactions starting on the primary from that point on. Explicitly defined transactions (with a `START TRANSACTION` or `BEGIN` statement) are subject to the timeout, disconnection, and incoming transaction blocking

even if they do not modify any data. To allow inspection of the primary while the function is operating, single statements that do not modify data, as listed in Permitted Queries Under Consistency Rules, are permitted to proceed.

Pass in the `server_uuid` of the member which you want to become the new primary of the group by issuing:

```
SELECT group_replication_set_as_primary(member_uuid);
```

From MySQL 8.0.29, you can add a timeout, for example:

```
SELECT group_replication_set_as_primary('00371d66-3c45-11ea-804b-080027337932', 300)
```

To check the status of the timeout, use the `PROCESSLIST_INFO` column in the Performance Schema threads table:

```
SELECT NAME, PROCESSLIST_INFO FROM performance_schema.threads WHERE NAME=\"thread/group_rpl/THD_transac
+-----------------------------------------+-----------------------------------------------------------
| NAME                                    | PROCESSLIST_INFO
+-----------------------------------------+-----------------------------------------------------------
| thread/group_rpl/THD_transaction_monitor | Group replication transaction monitor: Stopped client c
+-----------------------------------------+-----------------------------------------------------------
```

The status shows when the transaction monitoring thread has been created, when new transactions have been stopped, when the client connections with uncommitted transactions have been disconnected, and finally, when the process is complete and new transactions are allowed again.

While the action runs, you can check its progress by issuing:

```
SELECT event_name, work_completed, work_estimated FROM performance_schema.events_stages_current WHERE e
+-----------------------------------------------------------------------------+----------------+--
| event_name                                                                  | work_completed | w
+-----------------------------------------------------------------------------+----------------+--
| stage/group_rpl/Primary Election: Waiting for members to turn on super_read_only |              3 |
+-----------------------------------------------------------------------------+----------------+--
```

18.5.1.2 Changing a Group's Mode

This section explains how to change the mode which a group is running in, either single or multi-primary. The functions used to change a group's mode can be run on any member.

Changing to Single-Primary Mode

Use the `group_replication_switch_to_single_primary_mode()` function to change a group running in multi-primary mode to single-primary mode by issuing:

```
SELECT group_replication_switch_to_single_primary_mode()
```

When you change to single-primary mode, strict consistency checks are also disabled on all group members, as required in single-primary mode (`group_replication_enforce_update_everywhere_checks=OFF`).

If no string is passed in, the election of the new primary in the resulting single-primary group follows the election policies described in Section 18.1.3.1, "Single-Primary Mode". To override the election process and configure a specific member of the multi-primary group as the new primary in the process, get the `server_uuid` of the member and pass it to `group_replication_switch_to_single_primary_mode()`. For example, issue:

```
SELECT group_replication_switch_to_single_primary_mode(member_uuid);
```

If you invoke the function on a member running a MySQL Server version from 8.0.17, and all members are running MySQL Server version 8.0.17 or higher, you can only specify a new primary member that is running the lowest MySQL Server version in the group, based on the patch version. This safeguard

is applied to ensure the group maintains compatibility with new functions. If you do not specify a new primary member, the election process considers the patch version of the group members.

If any member is running a MySQL Server version between MySQL 8.0.13 and MySQL 8.0.16, this safeguard is not enforced for the group and you can specify any new primary member, but it is recommended to select a primary that is running the lowest MySQL Server version in the group. If you do not specify a new primary member, the election process considers only the major version of the group members.

While the action runs, you can check its progress by issuing:

```
SELECT event_name, work_completed, work_estimated FROM performance_schema.events_stages_current WHERE event
+-------------------------------------------------------------------------+----------------+------------
| event_name                                                              | work_completed | work_estima
+-------------------------------------------------------------------------+----------------+------------
| stage/group_rpl/Primary Switch: waiting for pending transactions to finish |              4 |
+-------------------------------------------------------------------------+----------------+------------
```

Changing to Multi-Primary Mode

Use the `group_replication_switch_to_multi_primary_mode()` function to change a group running in single-primary mode to multi-primary mode by issuing:

```
SELECT group_replication_switch_to_multi_primary_mode()
```

After some coordinated group operations to ensure the safety and consistency of your data, all members which belong to the group become primaries.

When you change a group that was running in single-primary mode to run in multi-primary mode, members running MySQL 8.0.17 or higher are automatically placed in read-only mode if they are running a higher MySQL server version than the lowest version present in the group. Members running MySQL 8.0.16 or lower do not carry out this check, and are always placed in read-write mode.

While the action runs, you can check its progress by issuing:

```
SELECT event_name, work_completed, work_estimated FROM performance_schema.events_stages_current WHERE event
+-------------------------------------------------------------------------+----------------+----------------+
| event_name                                                              | work_completed | work_estimated |
+-------------------------------------------------------------------------+----------------+----------------+
| stage/group_rpl/Multi-primary Switch: applying buffered transactions |              0 |              1 |
+-------------------------------------------------------------------------+----------------+----------------+
```

18.5.1.3 Using Group Replication Group Write Consensus

This section explains how to inspect and configure the maximum number of consensus instances at any time for a group. This maximum is referred to as the event horizon for a group, and is the maximum number of consensus instances that the group can execute in parallel. This enables you to fine tune the performance of your Group Replication deployment. For example, the default value of 10 is suitable for a group running on a LAN, but for groups operating over a slower network such as a WAN, increase this number to improve performance.

Inspecting a Group's Write Concurrency

Use the `group_replication_get_write_concurrency()` function to inspect a group's event horizon value at runtime by issuing:

```
SELECT group_replication_get_write_concurrency();
```

Configuring a Group's Write Concurrency

Use the `group_replication_set_write_concurrency()` function to set the maximum number of consensus instances that the system can execute in parallel by issuing:

```
SELECT group_replication_set_write_concurrency(instances);
```

where *`instances`* is the new maximum number of consensus instances. The `GROUP_REPLICATION_ADMIN` privilege is required to use this function.

18.5.1.4 Setting a Group's Communication Protocol Version

From MySQL 8.0.16, Group Replication has the concept of a communication protocol for the group. The Group Replication communication protocol version can be managed explicitly, and set to accommodate the oldest MySQL Server version that you want the group to support. This enables groups to be formed from members at different MySQL Server versions while ensuring backward compatibility.

- Versions from MySQL 5.7.14 allow compression of messages (see Section 18.7.4, “Message Compression”).
- Versions from MySQL 8.0.16 also allow fragmentation of messages (see Section 18.7.5, “Message Fragmentation”).
- Versions from MySQL 8.0.27 also allow the group communication engine to operate with a single consensus leader when the group is in single-primary mode and `group_replication_paxos_single_leader` is set to true (see Section 18.7.3, “Single Consensus Leader”).

All members of the group must use the same communication protocol version, so that group members can be at different MySQL Server releases but only send messages that can be understood by all group members.

A MySQL server at version X can only join and reach `ONLINE` status in a replication group if the group's communication protocol version is less than or equal to X. When a new member joins a replication group, it checks the communication protocol version that is announced by the existing members of the group. If the joining member supports that version, it joins the group and uses the communication protocol that the group has announced, even if the member supports additional communication capabilities. If the joining member does not support the communication protocol version, it is expelled from the group.

If two members attempt to join in the same membership change event, they can only join if the communication protocol version for both members is already compatible with the group's communication protocol version. Members with different communication protocol versions from the group must join in isolation. For example:

- One MySQL Server 8.0.16 instance can successfully join a group that uses the communication protocol version 5.7.24.
- One MySQL Server 5.7.24 instance cannot successfully join a group that uses the communication protocol version 8.0.16.
- Two MySQL Server 8.0.16 instances cannot simultaneously join a group that uses the communication protocol version 5.7.24.
- Two MySQL Server 8.0.16 instances can simultaneously join a group that uses the communication protocol version 8.0.16.

You can inspect the communication protocol in use by a group by using the `group_replication_get_communication_protocol()` function, which returns the oldest MySQL Server version that the group supports. All existing members of the group return the same communication protocol version. For example:

```
SELECT group_replication_get_communication_protocol();
+-----------------------------------------------+
| group_replication_get_communication_protocol() |
+-----------------------------------------------+
| 8.0.16                                        |
```

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

Note that the `group_replication_get_communication_protocol()` function returns the minimum MySQL version that the group supports, which might differ from the version number that was passed to the `group_replication_set_communication_protocol()` function, and from the MySQL Server version that is installed on the member where you use the function.

If you need to change the communication protocol version of a group so that members at earlier releases can join, use the `group_replication_set_communication_protocol()` function to specify the MySQL Server version of the oldest member that you want to allow. This makes the group fall back to a compatible communication protocol version if possible. The `GROUP_REPLICATION_ADMIN` privilege is required to use this function, and all existing group members must be online when you issue the statement, with no loss of majority. For example:

```
SELECT group_replication_set_communication_protocol("5.7.25");
```

If you upgrade all the members of a replication group to a new MySQL Server release, the group's communication protocol version is not automatically upgraded to match. If you no longer need to support members at earlier releases, you can use the `group_replication_set_communication_protocol()` function to set the communication protocol version to the new MySQL Server version to which you have upgraded the members. For example:

```
SELECT group_replication_set_communication_protocol("8.0.16");
```

The `group_replication_set_communication_protocol()` function is implemented as a group action, so it is executed at the same time on all members of the group. The group action starts buffering messages and waits for delivery of any outgoing messages that were already in progress to complete, then changes the communication protocol version and sends the buffered messages. If a member attempts to join the group at any time after you change the communication protocol version, the group members announce the new protocol version.

MySQL InnoDB cluster automatically and transparently manages the communication protocol versions of its members, whenever the cluster topology is changed using AdminAPI operations. An InnoDB cluster always uses the most recent communication protocol version that is supported by all the instances that are currently part of the cluster or joining it. For details, see InnoDB Cluster and Group Replication Protocol.

18.5.1.5 Configuring Member Actions

From MySQL 8.0.26, Group Replication has the capability to set actions for the members of a group to take in specified situations. Member actions can be enabled and disabled individually using functions. The member actions configuration for a server can also be reset to the default after it has left the group.

Administrators (with the `GROUP_REPLICATION_ADMIN` privilege) can configure a member action on the group's primary using the `group_replication_enable_member_action` or `group_replication_disable_member_action` function. The member actions configuration, consisting of all the member actions and whether they are enabled or disabled, is then propagated to other group members and joining members using Group Replication's group messages. All group members therefore have the same member actions configuration. You can also configure member actions on a server that is not part of a group, as long as the Group Replication plugin is installed. In that case, the member actions configuration is not propagated to any other servers.

If the server where you use the functions to configure a member action is part of a group, it must be the current primary in a group in single-primary mode, and it must be part of the majority. The configuration change is tracked internally by Group Replication, but it is not given a GTID and is not written to the binary log, so it is not propagated to any servers outside the group, such as downstream replicas. Group Replication increments the version number for its member actions configuration each time a member action is enabled or disabled.

The member actions configuration is propagated to members as follows:

- When starting a group, the member actions configuration of the server that bootstraps the group becomes the configuration for the group.
- If a group's lowest MySQL Server version supports member actions, joining members receive the group's member actions configuration during the state exchange process that takes place when they join. In that case, the joining member replaces its own member actions configuration with the group's.
- If a joining member that supports member actions joins a group where the lowest MySQL Server version does not support member actions, it does not receive a member actions configuration when it joins. In that case, the joining member resets its own configuration to the default.

A member that does not support member actions cannot join a group that has a member actions configuration, because its MySQL Server version is lower than the lowest version that the existing group members are running.

The Performance Schema table `replication_group_member_actions` lists the member actions that are available in the configuration, the events that trigger them, and whether or not they are currently enabled. Member actions have a priority from 1 to 100, with lower values being actioned first. If an error occurs when the member action is being carried out, the failure of the member action can be logged but otherwise ignored. If the failure of the member action is considered critical, it can be handled according to the policy specified by the `group_replication_exit_state_action` system variable.

The `mysql.replication_group_configuration_version` table, which can be viewed using the Performance Schema table `replication_group_configuration_version`, records the current version of the member actions configuration. Whenever a member action is enabled or disabled using the functions, the version number is incremented.

The `group_replication_reset_member_actions` function can only be used on a server that is not part of a group. It resets the member actions configuration to the default settings, and resets its version number to 1. The server must be writeable (with the `read_only` system variable set to `OFF`) and have the Group Replication plugin installed. You can use this function to remove the member actions configuration that a server used when it was part of a group, if you intend to use it as a standalone server with no member actions or different member actions.

Member action: `mysql_disable_super_read_only_if_primary`

The member action `mysql_disable_super_read_only_if_primary` can be configured to make a group in single-primary mode stay in super read-only mode when a new primary is elected, so that the group only accepts replicated transactions and does not accept any direct writes from clients. This setup means that when a group's purpose is to provide a secondary backup to another group for disaster tolerance, you can ensure that the secondary group remains synchronized with the first.

By default, super read-only mode is disabled on the primary when it is elected, so that the primary becomes read-write, and accepts updates from a replication source server and from clients. This is the situation when the member action `mysql_disable_super_read_only_if_primary` is enabled, which is its default setting. If you set the action to disabled using the `group_replication_disable_member_action` function, the primary remains in super read-only mode after election. In this state, it does not accept updates from any clients, even users who have the `CONNECTION_ADMIN` or `SUPER` privilege. It does continue to accept updates performed by replication threads.

18.5.2 Restarting a Group

Group Replication is designed to ensure that the database service is continuously available, even if some of the servers that form the group are currently unable to participate in it due to planned

maintenance or unplanned issues. As long as the remaining members are a majority of the group they can elect a new primary and continue to function as a group. However, if every member of a replication group leaves the group, and Group Replication is stopped on every member by a `STOP GROUP_REPLICATION` statement or system shutdown, the group now only exists in theory, as a configuration on the members. In that situation, to re-create the group, it must be started by bootstrapping as if it was being started for the first time.

The difference between bootstrapping a group for the first time and doing it for the second or subsequent times is that in the latter situation, the members of a group that was shut down might have different transaction sets from each other, depending on the order in which they were stopped or failed. A member cannot join a group if it has transactions that are not present on the other group members. For Group Replication, this includes both transactions that have been committed and applied, which are in the `gtid_executed` GTID set, and transactions that have been certified but not yet applied, which are in the `group_replication_applier` channel. The exact point at which a transaction is committed depends on the transaction consistency level that is set for the group (see Section 18.5.3, “Transaction Consistency Guarantees”). However, a Group Replication group member never removes a transaction that has been certified, which is a declaration of the member's intent to commit the transaction.

The replication group must therefore be restarted beginning with the most up to date member, that is, the member that has the most transactions executed and certified. The members with fewer transactions can then join and catch up with the transactions they are missing through distributed recovery. It is not correct to assume that the last known primary member of the group is the most up to date member of the group, because a member that was shut down later than the primary might have more transactions. You must therefore restart each member to check the transactions, compare all the transaction sets, and identify the most up to date member. This member can then be used to bootstrap the group.

Follow this procedure to restart a replication group safely after every member shuts down.

1. For each group member in turn, in any order:

   a. Connect a client to the group member. If Group Replication is not already stopped, issue a `STOP GROUP_REPLICATION` statement and wait for Group Replication to stop.

   b. Edit the MySQL Server configuration file (typically named `my.cnf` on Linux and Unix systems, or `my.ini` on Windows systems) and set the system variable `group_replication_start_on_boot=OFF`. This setting prevents Group Replication from starting when MySQL Server is started, which is the default.

      If you cannot change that setting on the system, you can just allow the server to attempt to start Group Replication, which will fail because the group has been fully shut down and not yet bootstrapped. If you take that approach, do not set `group_replication_bootstrap_group=ON` on any server at this stage.

   c. Start the MySQL Server instance, and verify that Group Replication has not been started (or has failed to start). Do not start Group Replication at this stage.

   d. Collect the following information from the group member:

      - The contents of the `gtid_executed` GTID set. You can get this by issuing the following statement:

        ```
        mysql> SELECT @@GLOBAL.GTID_EXECUTED
        ```

      - The set of certified transactions on the `group_replication_applier` channel. You can get this by issuing the following statement:

        ```
        mysql> SELECT received_transaction_set FROM \
                performance_schema.replication_connection_status WHERE \
                channel_name="group_replication_applier";
        ```

2. When you have collected the transaction sets from all the group members, compare them to find which member has the biggest transaction set overall, including both the executed transactions (`gtid_executed`) and the certified transactions (on the `group_replication_applier` channel). You can do this manually by looking at the GTIDs, or you can compare the GTID sets using stored functions, as described in Section 17.1.3.8, “Stored Function Examples to Manipulate GTIDs”.

3. Use the member that has the biggest transaction set to bootstrap the group, by connecting a client to the group member and issuing the following statements:

```
mysql> SET GLOBAL group_replication_bootstrap_group=ON;
mysql> START GROUP_REPLICATION;
mysql> SET GLOBAL group_replication_bootstrap_group=OFF;
```

   It is important not to store the setting `group_replication_bootstrap_group=ON` in the configuration file, otherwise when the server is restarted again, a second group with the same name is set up.

4. To verify that the group now exists with this founder member in it, issue this statement on the member that bootstrapped it:

```
mysql> SELECT * FROM performance_schema.replication_group_members;
```

5. Add each of the other members back into the group, in any order, by issuing a `START GROUP_REPLICATION` statement on each of them:

```
mysql> START GROUP_REPLICATION;
```

6. To verify that each member has joined the group, issue this statement on any member:

```
mysql> SELECT * FROM performance_schema.replication_group_members;
```

7. When the members have rejoined the group, if you edited their configuration files to set `group_replication_start_on_boot=OFF`, you can edit them again to set `ON` (or remove the system variable, since `ON` is the default).

18.5.3 Transaction Consistency Guarantees

One of the major implications of a distributed system such as Group Replication is the consistency guarantees that it provides as a group. In other words, the consistency of the global synchronization of transactions distributed across the members of the group. This section describes how Group Replication handles consistency guarantees depending on the events that occur in a group, and how to best configure your group's consistency guarantees.

18.5.3.1 Understanding Transaction Consistency Guarantees

In terms of distributed consistency guarantees, either in normal or failure repair operations, Group Replication has always been an eventual consistency system. This means that as soon as the incoming traffic slows down or stops, all group members have the same data content. The events that relate to the consistency of a system can be split into control operations, either manual or automatically triggered by failures; and data flow operations.

For Group Replication, the control operations that can be evaluated in terms of consistency are:

- a member joining or leaving, which is covered by Group Replication's Section 18.5.4, “Distributed Recovery” and write protection.
- network failures, which are covered by the fencing modes.
- in single-primary groups, primary failover, which can also be an operation triggered by `group_replication_set_as_primary()`.

Consistency Guarantees and Primary Failover

In a single-primary group, in the event of a primary failover when a secondary is promoted to primary, the new primary can either be made available to application traffic immediately, regardless of how large the replication backlog is, or alternatively access to it can be restricted until the backlog has been applied.

With the first approach, the group takes the minimum time possible to secure a stable group membership after a primary failure by electing a new primary and then allowing data access immediately while it is still applying any possible backlog from the old primary. Write consistency is ensured, but reads can temporarily retrieve stale data while the new primary applies the backlog. For example, if client C1 wrote `A=2 WHERE A=1` on the old primary just before its failure, when client C1 is reconnected to the new primary it could potentially read `A=1` until the new primary applies its backlog and catches up with the state of the old primary before it left the group.

With the second alternative, the system secures a stable group membership after the primary failure and elects a new primary in the same way as the first alternative, but in this case the group then waits until the new primary applies all backlog and only then does it permit data access. This ensures that in a situation as described previously, when client C1 is reconnected to the new primary it reads `A=2`. However, the trade-off is that the time required to failover is then proportional to the size of the backlog, which on a correctly configured group should be small .

Prior to MySQL 8.0.14 there was no way to configure the failover policy, by default availability was maximized as described in the first approach. In a group with members running MySQL 8.0.14 and higher, you can configure the level of transaction consistency guarantees provided by members during primary failover using the `group_replication_consistency` variable. See Impact of Consistency on Primary Election.

Data Flow Operations

Data flow is relevant to group consistency guarantees due to the reads and writes executed against a group, especially when these operations are distributed across all members. Data flow operations apply to both modes of Group Replication: single-primary and multi-primary, however to make this explanation clearer it is restricted to single-primary mode. The usual way to split incoming read or write transactions across a single-primary group's members is to route writes to the primary and evenly distribute reads to the secondaries. Since the group should behave as a single entity, it is reasonable to expect that writes on the primary are instantaneously available on the secondaries. Although Group Replication is written using Group Communication System (GCS) protocols that implement the Paxos algorithm, some parts of Group Replication are asynchronous, which implies that data is asynchronously applied to secondaries. This means that a client C2 can write `B=2 WHERE B=1` on the primary, immediately connect to a secondary and read `B=1`. This is because the secondary is still applying backlog, and has not applied the transaction which was applied by the primary.

Transaction Synchronization Points

You configure a group's consistency guarantee based on the point at which you want to synchronize transactions across the group. To help you understand the concept, this section simplifies the points of synchronizing transactions across a group to be at the time of a read operation or at the time of a write operation. If data is synchronized at the time of a read, the current client session waits until a given point, which is the point in time that all preceding update transactions have been applied, before it can start executing. With this approach, only this session is affected, all other concurrent data operations are not affected.

If data is synchronized at the time of write, the writing session waits until all secondaries have written their data. Group Replication uses a total order on writes, and therefore this implies waiting for this and all preceding writes that are in secondaries' queues to be applied. Therefore when using this synchronization point, the writing session waits for all secondaries queues to be applied.

Any alternative ensures that in the situation described for client C2 would always read `B=2` even if immediately connected to a secondary. Each alternative has its advantages and disadvantages, which are directly related to your system workload. The following examples describe different types of workloads and advise which point of synchronization is appropriate.

Imagine the following situations:

- you want to load balance your reads without deploying additional restrictions on which server you read from to avoid reading stale data, group writes are much less common than group reads.
- you have a group that has a predominantly read-only data, you want read-write transactions to be applied everywhere once they commit, so that subsequent reads are done on up-to-date data that includes the latest write. This ensures that you do not pay the synchronization cost for every RO transaction, but only on RW ones.

In these cases, you should choose to synchronize on writes.

Imagine the following situations:

- You want to load balance your reads without deploying additional restrictions on which server you read from to avoid reading stale data, group writes are much more common than group reads.
- You want specific transactions in your workload to always read up-to-date data from the group, for example whenever sensitive data is updated (such as credentials for a file or similar data) and you want to enforce that reads retrieve the most up to date value.

In these cases, you should choose to synchronize on reads.

18.5.3.2 Configuring Transaction Consistency Guarantees

Although the Transaction Synchronization Points section explains that conceptually there are two synchronization points from which you can choose: on read or on write, these terms were a simplification and the terms used in Group Replication are: *before* and *after* transaction execution. The consistency level can have a different impact on read-only (RO) and read-write (RW) transactions processed by the group as demonstrated in this section.

- How to Choose a Consistency Level
- Impacts of Consistency Levels
- Impact of Consistency on Primary Election
- Permitted Queries Under Consistency Rules

The following list shows the possible consistency levels that you can configure in Group Replication using the `group_replication_consistency` variable, in order of increasing transaction consistency guarantee:

- `EVENTUAL`

  Both RO and RW transactions do not wait for preceding transactions to be applied before executing. This was the behavior of Group Replication before the `group_replication_consistency` variable was added. A RW transaction does not wait for other members to apply a transaction. This means that a transaction could be externalized on one member before the others. This also means that in the event of a primary failover, the new primary can accept new RO and RW transactions before the previous primary transactions are all applied. RO transactions could result in outdated values, RW transactions could result in a rollback due to conflicts.

- `BEFORE_ON_PRIMARY_FAILOVER`

  New RO or RW transactions with a newly elected primary that is applying backlog from the old primary are held (not applied) until any backlog has been applied. This ensures that when a primary failover happens, intentionally or not, clients always see the latest value on the primary. This guarantees consistency, but means that clients must be able to handle the delay in the event that a backlog is being applied. Usually this delay should be minimal, but it does depend on the size of the backlog.

- `BEFORE`

  A RW transaction waits for all preceding transactions to complete before being applied. A RO transaction waits for all preceding transactions to complete before being executed. This ensures that this transaction reads the latest value by only affecting the latency of the transaction. This reduces the overhead of synchronization on every RW transaction, by ensuring synchronization is used only on RO transactions. This consistency level also includes the consistency guarantees provided by `BEFORE_ON_PRIMARY_FAILOVER`.

- `AFTER`

  A RW transaction waits until its changes have been applied to all of the other members. This value has no effect on RO transactions. This mode ensures that when a transaction is committed on the local member, any subsequent transaction reads the written value or a more recent value on any group member. Use this mode with a group that is used for predominantly RO operations to ensure that applied RW transactions are applied everywhere once they commit. This could be used by your application to ensure that subsequent reads fetch the latest data which includes the latest writes. This reduces the overhead of synchronization on every RO transaction, by ensuring synchronization is used only on RW transactions. This consistency level also includes the consistency guarantees provided by `BEFORE_ON_PRIMARY_FAILOVER`.

- `BEFORE_AND_AFTER`

  A RW transaction waits for 1) all preceding transactions to complete before being applied and 2) until its changes have been applied on other members. A RO transaction waits for all preceding transactions to complete before execution takes place. This consistency level also includes the consistency guarantees provided by `BEFORE_ON_PRIMARY_FAILOVER`.

The `BEFORE` and `BEFORE_AND_AFTER` consistency levels can be both used on RO and RW transactions. The `AFTER` consistency level has no impact on RO transactions, because they do not generate changes.

How to Choose a Consistency Level

The different consistency levels provide flexibility to both DBAs, who can use them to set up their infrastructure; and to developers who can use the consistency level that best suits their application's requirements. The following scenarios show how to choose a consistency guarantee level based on how you use your group:

- *Scenario 1* you want to load balance your reads without worrying about stale reads, your group write operations are considerably fewer than your group read operations. In this case, you should choose `AFTER`.
- *Scenario 2* you have a data set that applies a lot of writes and you want to do occasional reads without having to worry about reading stale data. In this case, you should choose `BEFORE`.
- *Scenario 3* you want specific transactions in your workload to always read up-to-date data from the group, so that whenever that sensitive data is updated (such as credentials for a file or similar data) you want to enforce that reads always read the most up to date value. In this case, you should choose `BEFORE`.
- *Scenario 4* you have a group that has predominantly read-only (RO) data, you want your read-write (RW) transactions to be applied everywhere once they commit, so that subsequent reads are done on up-to-date data that includes your latest writes and you do not pay the synchronization on every RO transaction, but only on RW ones. In this case, you should choose `AFTER`.
- *Scenario 5* you have a group that has predominantly read-only data, you want your read-write (RW) transactions to always read up-to-date data from the group and to be applied everywhere once they commit, so that subsequent reads are done on up-to-date data that includes your latest write and you do not pay the synchronization on every read-only (RO) transaction, but only on RW ones. In this case, you should choose `BEFORE_AND_AFTER`.

You have the freedom to choose the scope at which the consistency level is enforced. This is important because consistency levels could have a negative impact on group performance if you set them at a global scope. Therefore you can configure the consistency level of a group by using the `group_replication_consistency` system variable at different scopes.

To enforce the consistency level on the current session, use the session scope:

```
> SET @@SESSION.group_replication_consistency= 'BEFORE';
```

To enforce the consistency level on all sessions, use the global scope:

```
> SET @@GLOBAL.group_replication_consistency= 'BEFORE';
```

The possibility of setting the consistency level on specific sessions enables you to take advantage of scenarios such as:

- *Scenario 6* A given system handles several instructions that do not require a strong consistency level, but one kind of instruction does require strong consistency: managing access permissions to documents;. In this scenario, the system changes access permissions and it wants to be sure that all clients see the correct permission. You only need to `SET @@SESSION.group_replication_consistency= 'AFTER'`, on those instructions and leave the other instructions to run with `EVENTUAL` set at the global scope.
- *Scenario 7* On the same system as described in Scenario 6, every day an instruction needs to do some analytical processing, and as such it requires to always read the most up-to-date data. To achieve this, you only need to `SET @@SESSION.group_replication_consistency= 'BEFORE'` on that specific instruction.

To summarize, you do not need to run all transactions with a specific consistency level, especially if only some transactions actually require it.

Note that all read-write transactions are totally ordered in Group Replication, so even when you set the consistency level to `AFTER` for the current session this transaction waits until its changes are applied on all members, which means waiting for this and all preceding transactions that could be in the secondaries' queues. In practice, the consistency level `AFTER` waits for everything until and including this transaction.

Impacts of Consistency Levels

Another way to classify the consistency levels is in terms of impact on the group, that is, the repercussions that the consistency levels have on the other members.

The `BEFORE` consistency level, apart from being ordered on the transaction stream, only impacts on the local member. That is, it does not require coordination with the other members and does not have repercussions on their transactions. In other words, `BEFORE` only impacts the transactions on which it is used.

The `AFTER` and `BEFORE_AND_AFTER` consistency levels do have side-effects on concurrent transactions executed on other members. These consistency levels make the other members transactions wait if transactions with the `EVENTUAL` consistency level start while a transaction with `AFTER` or `BEFORE_AND_AFTER` is executing. The other members wait until the `AFTER` transaction is committed on that member, even if the other member's transactions have the `EVENTUAL` consistency level. In other words, `AFTER` and `BEFORE_AND_AFTER` impact *all* `ONLINE` group members.

To illustrate this further, imagine a group with 3 members, M1, M2 and M3. On member M1 a client issues:

```
> SET @@SESSION.group_replication_consistency= AFTER;
> BEGIN;
> INSERT INTO t1 VALUES (1);
> COMMIT;
```

Then, while the above transaction is being applied, on member M2 a client issues:

```
> SET SESSION group_replication_consistency= EVENTUAL;
```

In this situation, even though the second transaction's consistency level is `EVENTUAL`, because it starts executing while the first transaction is already in the commit phase on M2, the second transaction has to wait for the first transaction to finish the commit and only then can it execute.

You can only use the consistency levels `BEFORE`, `AFTER` and `BEFORE_AND_AFTER` on `ONLINE` members, attempting to use them on members in other states causes a session error.

Transactions whose consistency level is not `EVENTUAL` hold execution until a timeout, configured by `wait_timeout` value is reached, which defaults to 8 hours. If the timeout is reached an `ER_GR_HOLD_WAIT_TIMEOUT` error is thrown.

Impact of Consistency on Primary Election

This section describes how a group's consistency level impacts on a single-primary group that has elected a new primary. Such a group automatically detects failures and adjusts the view of the members that are active, in other words the membership configuration. Furthermore, if a group is deployed in single-primary mode, whenever the group's membership changes there is a check performed to detect if there is still a primary member in the group. If there is none, a new one is selected from the list of secondary members. Typically, this is known as the secondary promotion.

Given the fact that the system detects failures and reconfigures itself automatically, the user may also expect that once the promotion takes place, the new primary is in the exact state, data-wise, as that of the old one. In other words, the user may expect that there is no backlog of replicated transactions to be applied on the new primary once he is able to read from and write to it. In practical terms, the user may expect that once his application fails-over to the new primary, there would be no chance, even if temporarily, to read old data or write into old data records.

When flow control is activated and properly tuned on a group, there is only a small chance of transiently reading stale data from a newly elected primary immediately after the promotion, as there should not be a backlog, or if there is one it should be small. Moreover, you might have a proxy or middleware layers that govern application accesses to the primary after a promotion and enforce the consistency criteria at that level. If your group members are using MySQL 8.0.14 or higher, you can specify the behavior of the new primary once it is promoted using the `group_replication_consistency` variable, which controls whether a newly elected primary blocks both reads and writes until after the backlog is fully applied or if it behaves in the manner of members running MySQL 8.0.13 or earlier. If the `group_replication_consistency` option was set to `BEFORE_ON_PRIMARY_FAILOVER` on a newly elected primary which has backlog to apply, and transactions are issued against the new primary while it is still applying the backlog, incoming transactions are blocked until the backlog is fully applied. Thus, the following anomalies are prevented:

- No stale reads for read-only and read-write transactions. This prevents stale reads from being externalized to the application by the new primary.
- No spurious roll backs for read-write transactions, due to write-write conflicts with replicated read-write transactions still in the backlog waiting to be applied.
- No read skew on read-write transactions, such as:

  ```
  > BEGIN;
  > SELECT x FROM t1; -- x=1 because x=2 is in the backlog;
  > INSERT x INTO t2;
  > COMMIT;
  ```

  This query should not cause a conflict but writes outdated values.

To summarize, when `group_replication_consistency` is set to `BEFORE_ON_PRIMARY_FAILOVER` you are choosing to prioritize consistency over availability,

because reads and writes are held whenever a new primary is elected. This is the trade-off you have to consider when configuring your group. It should also be remembered that if flow control is working correctly, backlog should be minimal. Note that the higher consistency levels `BEFORE`, `AFTER`, and `BEFORE_AND_AFTER` also include the consistency guarantees provided by `BEFORE_ON_PRIMARY_FAILOVER`.

To guarantee that the group provides the same consistency level regardless of which member is promoted to primary, all members of the group should have `BEFORE_ON_PRIMARY_FAILOVER` (or a higher consistency level) persisted to their configuration. For example, on each member issue:

```
> SET PERSIST group_replication_consistency='BEFORE_ON_PRIMARY_FAILOVER';
```

This ensures that the members all behave in the same way, and that the configuration is persisted after a restart of the member.

A transaction cannot be on-hold forever, and if the time held exceeds `wait_timeout` it returns an `ER_GR_HOLD_WAIT_TIMEOUT` error.

Permitted Queries Under Consistency Rules

Although all writes are held when using `BEFORE_ON_PRIMARY_FAILOVER` consistency level, not all reads are blocked to ensure that you can still inspect the server while it is applying backlog after a promotion took place. This is useful for debugging, monitoring, observability and troubleshooting. Some queries that do not modify data are allowed, such as the following:

- `SHOW` statements - from MySQL 8.0.27 this is restricted to those that do not depend on data, only on status and configuration, as listed below
- `SET` statements
- `DO` statements that do not use tables or loadable functions
- `EMPTY` statements
- `USE` statements
- Using `SELECT` statements against the `performance_schema` and `sys` databases
- Using `SELECT` statements against the `PROCESSLIST` table from the `infoschema` database
- `SELECT` statements that do not use tables or loadable functions
- `STOP GROUP_REPLICATION` statements
- `SHUTDOWN` statements
- `RESET PERSIST` statements

The `SHOW` statements that are allowed from MySQL 8.0.27 are `SHOW VARIABLES`, `SHOW PROCESSLIST`, `SHOW STATUS`, `SHOW ENGINE INNODB LOGS`, `SHOW ENGINE INNODB STATUS`, `SHOW ENGINE INNODB MUTEX`, `SHOW MASTER STATUS`, `SHOW REPLICA STATUS`, `SHOW CHARACTER SET`, `SHOW COLLATION`, `SHOW BINARY LOGS`, `SHOW OPEN TABLES`, `SHOW REPLICAS`, `SHOW BINLOG EVENTS`, `SHOW WARNINGS`, `SHOW ERRORS`, `SHOW ENGINES`, `SHOW PRIVILEGES`, `SHOW PROCEDURE STATUS`, `SHOW FUNCTION STATUS`, `SHOW PLUGINS`, `SHOW EVENTS`, `SHOW PROFILE`, `SHOW PROFILES`, and `SHOW RELAYLOG EVENTS`.

18.5.4 Distributed Recovery

Whenever a member joins or rejoins a replication group, it must catch up with the transactions that were applied by the group members before it joined, or while it was away. This process is called distributed recovery.

The joining member begins by checking the relay log for its `group_replication_applier` channel for any transactions that it already received from the group but did not yet apply. If the joining member was in the group previously, it might find unapplied transactions from before it left, in which case it applies these as a first step. A member that is new to the group does not have anything to apply.

After this, the joining member connects to an online existing member to carry out state transfer. The joining member transfers all the transactions that took place in the group before it joined or while it was away, which are provided by the existing member (called the *donor*). Next, the joining member applies the transactions that took place in the group while this state transfer was in progress. When this process is complete, the joining member has caught up with the remaining servers in the group, and it begins to participate normally in the group.

Group Replication uses a combination of these methods for state transfer during distributed recovery:

- A remote cloning operation using the clone plugin's function, which is available from MySQL 8.0.17. To enable this method of state transfer, you must install the clone plugin on the group members and the joining member. Group Replication automatically configures the required clone plugin settings and manages the remote cloning operation.
- Replicating from a donor's binary log and applying the transactions on the joining member. This method uses a standard asynchronous replication channel named `group_replication_recovery` that is established between the donor and the joining member.

Group Replication automatically selects the best combination of these methods for state transfer after you issue `START GROUP_REPLICATION` on the joining member. To do this, Group Replication checks which existing members are suitable as donors, how many transactions the joining member needs from a donor, and whether any required transactions are no longer present in the binary log files on any group member. If the transaction gap between the joining member and a suitable donor is large, or if some required transactions are not in any donor's binary log files, Group Replication begins distributed recovery with a remote cloning operation. If there is not a large transaction gap, or if the clone plugin is not installed, Group Replication proceeds directly to state transfer from a donor's binary log.

- During a remote cloning operation, the existing data on the joining member is removed, and replaced with a copy of the donor's data. When the remote cloning operation is complete and the joining member has restarted, state transfer from a donor's binary log is carried out to get the transactions that the group applied while the remote cloning operation was in progress.
- During state transfer from a donor's binary log, the joining member replicates and applies the required transactions from the donor's binary log, applying the transactions as they are received, up to the point where the binary log records that the joining member joined the group (a view change event). While this is in progress, the joining member buffers the new transactions that the group applies. When state transfer from the binary log is complete, the joining member applies the buffered transactions.

When the joining member is up to date with all the group's transactions, it is declared online and can participate in the group as a normal member, and distributed recovery is complete.

Tip

State transfer from the binary log is Group Replication's base mechanism for distributed recovery, and if the donors and joining members in your replication group are not set up to support cloning, this is the only available option. As state transfer from the binary log is based on classic asynchronous replication, it might take a very long time if the server joining the group does not have the group's data at all, or has data taken from a very old backup image. In this situation, it is therefore recommended that before adding a server to the group, you should set it up with the group's data by transferring a fairly recent snapshot of a server already in the group. This minimizes the time taken for distributed recovery, and reduces the impact on donor servers, since they have to retain and transfer fewer binary log files.

18.5.4.1 Connections for Distributed Recovery

When a joining member connects to an online existing member for state transfer during distributed recovery, the joining member acts as a client on the connection and the existing member acts as a server. When state transfer from the donor's binary log is in progress over this connection (using the asynchronous replication channel `group_replication_recovery`), the joining member acts as the replica and the existing member acts as the source. When a remote cloning operation is in progress over this connection, the joining member acts as a recipient and the existing member acts as a donor. Configuration settings that apply to those roles outside the Group Replication context can apply for Group Replication also, unless they are overridden by a Group Replication-specific configuration setting or behavior.

The connection that an existing member offers to a joining member for distributed recovery is not the same connection that is used by Group Replication for communication between online members of the group.

- The connection used by the group communication engine for Group Replication (XCom, a Paxos variant) for TCP communication between remote XCom instances is specified by the `group_replication_local_address` system variable. This connection is used for TCP/IP messages between online members. Communication with the local instance is over an input channel using shared memory.
- For distributed recovery, up to MySQL 8.0.20, group members offer their standard SQL client connection to joining members, as specified by MySQL Server's `hostname` and `port` system variables. If an alternative port number is specified by the `report_port` system variable, that one is used instead.
- From MySQL 8.0.21, group members may advertise an alternative list of distributed recovery endpoints as dedicated client connections for joining members, allowing you to control distributed recovery traffic separately from connections by regular client users of the member. You specify this list using the `group_replication_advertise_recovery_endpoints` system variable, and the member transmits their list of distributed recovery endpoints to the group when they join the group. The default is that the member continues to offer the standard SQL client connection as in earlier releases.

Important

Distributed recovery can fail if a joining member cannot correctly identify the other members using the host name as defined by MySQL Server's `hostname` system variable. It is recommended that operating systems running MySQL have a properly configured unique host name, either using DNS or local settings. The host name that the server is using for SQL client connections can be verified in the `Member_host` column of the Performance Schema table `replication_group_members`. If multiple group members externalize a default host name set by the operating system, there is a chance of the joining member not resolving it to the correct member address and not being able to connect for distributed recovery. In this situation you can use MySQL Server's `report_host` system variable to configure a unique host name to be externalized by each of the servers.

The steps for a joining member to establish a connection for distributed recovery are as follows:

1. When the member joins the group, it connects with one of the seed members included in the list in its `group_replication_group_seeds` system variable, initially using the `group_replication_local_address` connection as specified in that list. The seed members might be a subset of the group.
2. Over this connection, the seed member uses Group Replication's membership service to provide the joining member with a list of all the members that are online in the group, in the form of a view. The membership information includes the details of the distributed recovery endpoints or standard SQL client connection offered by each member for distributed recovery.

3. The joining member selects a suitable group member from this list to be its donor for distributed recovery, following the behaviors described in Section 18.5.4.4, “Fault Tolerance for Distributed Recovery”.

4. The joining member then attempts to connect to the donor using the donor's advertised distributed recovery endpoints, trying each in turn in the order they are specified in the list. If the donor provides no endpoints, the joining member attempts to connect using the donor's standard SQL client connection. The SSL requirements for the connection are as specified by the `group_replication_recovery_ssl_*` options described in SSL and Authentication for Distributed Recovery.

5. If the joining member is not able to connect to the selected donor, it retries with other suitable donors, following the behaviors described in Section 18.5.4.4, “Fault Tolerance for Distributed Recovery”. Note that if the joining member exhausts the list of advertised endpoints without making a connection, it does not fall back to the donor's standard SQL client connection, but switches to another donor.

6. When the joining member establishes a distributed recovery connection with a donor, it uses that connection for state transfer as described in Section 18.5.4, “Distributed Recovery”. The host and port for the connection that is used are shown in the joining member's log. Note that if a remote cloning operation is used, when the joining member has restarted at the end of the operation, it establishes a connection with a new donor for state transfer from the binary log. This might be a connection to a different member from the original donor used for the remote cloning operation, or it might be a different connection to the original donor. In any case, the distributed recovery process continues in the same way as it would have with the original donor.

Selecting addresses for distributed recovery endpoints

IP addresses supplied by the `group_replication_advertise_recovery_endpoints` system variable as distributed recovery endpoints do not have to be configured for MySQL Server (that is, they do not have to be specified by the `admin_address` system variable or in the list for the `bind_address` system variable). They do have to be assigned to the server. Any host names used must resolve to a local IP address. IPv4 and IPv6 addresses can be used.

The ports supplied for the distributed recovery endpoints do have to be configured for MySQL Server, so they must be specified by the `port`, `report_port`, or `admin_port` system variable. The server must listen for TCP/IP connections on these ports. If you specify the `admin_port`, the replication user for distributed recovery needs the `SERVICE_CONNECTION_ADMIN` privilege to connect. Selecting the `admin_port` keeps distributed recovery connections separate from regular MySQL client connections.

Joining members try each of the endpoints in turn in the order they are specified on the list. If `group_replication_advertise_recovery_endpoints` is set to `DEFAULT` rather than a list of endpoints, the standard SQL client connection is offered. Note that the standard SQL client connection is not automatically included on a list of distributed recovery endpoints, and is not offered as a fallback if the donor's list of endpoints is exhausted without a connection. If you want to offer the standard SQL client connection as one of a number of distributed recovery endpoints, you must include it explicitly in the list specified by `group_replication_advertise_recovery_endpoints`. You can put it in the last place so that it acts as a last resort for connection.

A group member's distributed recovery endpoints (or standard SQL client connection if endpoints are not provided) do not need to be added to the Group Replication allowlist specified by the `group_replication_ip_allowlist` (from MySQL 8.0.22) or `group_replication_ip_whitelist` system variable. The allowlist is only for the address specified by `group_replication_local_address` for each member. A joining member must have its initial connection to the group permitted by the allowlist in order to retrieve the address or addresses for distributed recovery.

The distributed recovery endpoints that you list are validated when the system variable is set and when a `START GROUP_REPLICATION` statement has been issued. If the list cannot be parsed correctly, or if any of the endpoints cannot be accessed on the host because the server is not listening on them, Group Replication logs an error and does not start.

Compression for Distributed Recovery

From MySQL 8.0.18, you can optionally configure compression for distributed recovery by the method of state transfer from a donor's binary log. Compression can benefit distributed recovery where network bandwidth is limited and the donor has to transfer many transactions to the joining member. The `group_replication_recovery_compression_algorithms` and `group_replication_recovery_zstd_compression_level` system variables configure permitted compression algorithms, and the `zstd` compression level, used when carrying out state transfer from a donor's binary log. For more information, see Section 4.2.8, "Connection Compression Control".

Note that these compression settings do not apply for remote cloning operations. When a remote cloning operation is used for distributed recovery, the clone plugin's `clone_enable_compression` setting applies.

Replication User for Distributed Recovery

Distributed recovery requires a replication user that has the correct permissions so that Group Replication can establish direct member-to-member replication channels. The replication user must also have the correct permissions to act as the clone user on the donor for a remote cloning operation. The same replication user must be used for distributed recovery on every group member. For instructions to set up this replication user, see Section 18.2.1.3, "User Credentials For Distributed Recovery". For instructions to secure the replication user credentials, see Section 18.6.3.1, "Secure User Credentials for Distributed Recovery".

SSL and Authentication for Distributed Recovery

SSL for distributed recovery is configured separately from SSL for normal group communications, which is determined by the server's SSL settings and the `group_replication_ssl_mode` system variable. For distributed recovery connections, dedicated Group Replication distributed recovery SSL system variables are available to configure the use of certificates and ciphers specifically for distributed recovery.

By default, SSL is not used for distributed recovery connections. To activate it, set `group_replication_recovery_use_ssl=ON`, and configure the Group Replication distributed recovery SSL system variables as described in Section 18.6.3, "Securing Distributed Recovery Connections". You need a replication user that is set up to use SSL.

When distributed recovery is configured to use SSL, Group Replication applies this setting for remote cloning operations, as well as for state transfer from a donor's binary log. Group Replication automatically configures the settings for the clone SSL options (`clone_ssl_ca`, `clone_ssl_cert`, and `clone_ssl_key`) to match your settings for the corresponding Group Replication distributed recovery options (`group_replication_recovery_ssl_ca`, `group_replication_recovery_ssl_cert`, and `group_replication_recovery_ssl_key`).

If you are not using SSL for distributed recovery (so `group_replication_recovery_use_ssl` is set to `OFF`), and the replication user account for Group Replication authenticates with the `caching_sha2_password` plugin (which is the default in MySQL 8.0) or the `sha256_password` plugin, RSA key-pairs are used for password exchange. In this case, either use the `group_replication_recovery_public_key_path` system variable to specify the RSA public key file, or use the `group_replication_recovery_get_public_key` system variable to request the public key from the source, as described in Replication User With The Caching SHA-2 Authentication Plugin.

18.5.4.2 Cloning for Distributed Recovery

MySQL Server's clone plugin is available from MySQL 8.0.17. If you want to use remote cloning operations for distributed recovery in a group, you must set up existing members and joining members beforehand to support this function. If you do not want to use this function in a group, do not set it up, in which case Group Replication only uses state transfer from the binary log.

To use cloning, at least one existing group member and the joining member must be set up beforehand to support remote cloning operations. As a minimum, you must install the clone plugin on the donor and joining member, grant the `BACKUP_ADMIN` permission to the replication user for distributed recovery, and set the `group_replication_clone_threshold` system variable to an appropriate level. To ensure the maximum availability of donors, it is advisable to set up all current and future group members to support remote cloning operations.

Be aware that a remote cloning operation removes user-created tablespaces and data from the joining member before transferring the data from the donor. If the operation is stopped while in progress, the joining member might be left with partial data or no data. This can be repaired by retrying the remote cloning operation, which Group Replication does automatically.

Prerequisites for Cloning

For full instructions to set up and configure the clone plugin, see Section 5.6.7, “The Clone Plugin” . Detailed prerequisites for a remote cloning operation are covered in Section 5.6.7.3, “Cloning Remote Data” . For Group Replication, note the following key points and differences:

- The donor (an existing group member) and the recipient (the joining member) must have the clone plugin installed and active. For instructions to do this, see Section 5.6.7.1, “Installing the Clone Plugin” .
- The donor and the recipient must run on the same operating system, and must have the same MySQL Server version (which must be MySQL 8.0.17 or above to support the clone plugin). Cloning is therefore not suitable for groups where members run different MySQL Server versions.
- The donor and the recipient must have the Group Replication plugin installed and active, and any other plugins that are active on the donor (such as a keyring plugin) must also be active on the recipient.
- If distributed recovery is configured to use SSL (`group_replication_recovery_use_ssl=ON`), Group Replication applies this setting for remote cloning operations. Group Replication automatically configures the settings for the clone SSL options (`clone_ssl_ca`, `clone_ssl_cert`, and `clone_ssl_key`) to match your settings for the corresponding Group Replication distributed recovery options (`group_replication_recovery_ssl_ca`, `group_replication_recovery_ssl_cert`, and `group_replication_recovery_ssl_key`).
- You do not need to set up a list of valid donors in the `clone_valid_donor_list` system variable for the purpose of joining a replication group. Group Replication configures this setting automatically for you after it selects a donor from the existing group members. Note that remote cloning operations use the server's SQL protocol hostname and port.
- The clone plugin has a number of system variables to manage the network load and performance impact of the remote cloning operation. Group Replication does not configure these settings, so you can review them and set them if you want to, or allow them to default. Note that when a remote cloning operation is used for distributed recovery, the clone plugin's `clone_enable_compression` setting applies to the operation, rather than the Group Replication compression setting.
- To invoke the remote cloning operation on the recipient, Group Replication uses the internal `mysql.session` user, which already has the `CLONE_ADMIN` privilege, so you do not need to set this up.
- As the clone user on the donor for the remote cloning operation, Group Replication uses the replication user that you set up for distributed recovery (which is covered in Section 18.2.1.3, “User Credentials For Distributed Recovery”). You must therefore give the `BACKUP_ADMIN` privilege to this replication user on all group members that support cloning. Also give the privilege to the replication user on joining members when you are configuring them for Group Replication, because they can act as donors after they join the group. The same replication user is used for distributed recovery on every group member. To give this privilege to the replication user on existing members, you can issue this statement on each group member individually with binary logging disabled, or on one group member with binary logging enabled:

```
GRANT BACKUP_ADMIN ON *.* TO rpl_user@'%';
```

- If you use `START GROUP_REPLICATION` to specify the replication user credentials on a server that previously supplied the user credentials using `CHANGE REPLICATION SOURCE TO` | `CHANGE MASTER TO`, ensure that you remove the user credentials from the replication metadata repositories before any remote cloning operations take place. Also ensure that `group_replication_start_on_boot=OFF` is set on the joining member. For instructions, see Section 18.6.3, “Securing Distributed Recovery Connections”. If you do not unset the user credentials, they are transferred to the joining member during remote cloning operations. The `group_replication_recovery` channel could then be inadvertently started with the stored credentials, on either the original member or members that were cloned from it. An automatic start of Group Replication on server boot (including after a remote cloning operation) would use the stored user credentials, and they would also be used if an operator did not specify the distributed recovery credentials on a `START GROUP_REPLICATION` command.

Threshold for Cloning

When group members have been set up to support cloning, the `group_replication_clone_threshold` system variable specifies a threshold, expressed as a number of transactions, for the use of a remote cloning operation in distributed recovery. If the gap between the transactions on the donor and the transactions on the joining member is larger than this number, a remote cloning operation is used for state transfer to the joining member when this is technically possible. Group Replication calculates whether the threshold has been exceeded based on the `gtid_executed` sets of the existing group members. Using a remote cloning operation in the event of a large transaction gap lets you add new members to the group without transferring the group's data to the server manually beforehand, and also enables a member that is very out of date to catch up more efficiently.

The default setting for the `group_replication_clone_threshold` Group Replication system variable is extremely high (the maximum permitted sequence number for a transaction in a GTID), so it effectively deactivates cloning wherever state transfer from the binary log is possible. To enable Group Replication to select a remote cloning operation for state transfer where this is more appropriate, set the system variable to specify a number of transactions as the transaction gap above which you want cloning to take place.

Warning

Do not use a low setting for `group_replication_clone_threshold` in an active group. If a number of transactions above the threshold takes place in the group while the remote cloning operation is in progress, the joining member triggers a remote cloning operation again after restarting, and could continue this indefinitely. To avoid this situation, ensure that you set the threshold to a number higher than the number of transactions that you would expect to occur in the group during the time taken for the remote cloning operation.

Group Replication attempts to execute a remote cloning operation regardless of your threshold when state transfer from a donor's binary log is impossible, for example because the transactions needed by the joining member are not available in the binary log on any existing group member. Group Replication identifies this based on the `gtid_purged` sets of the existing group members. You cannot use the `group_replication_clone_threshold` system variable to deactivate cloning when the required transactions are not available in any member's binary log files, because in that situation cloning is the only alternative to transferring data to the joining member manually.

Cloning Operations

When group members and joining members are set up for cloning, Group Replication manages remote cloning operations for you. A remote cloning operation might take some time to complete, depending on the size of the data. See Section 5.6.7.10, “Monitoring Cloning Operations” for information on monitoring the process.

Note

When state transfer is complete, Group Replication restarts the joining member to complete the process. If `group_replication_start_on_boot=OFF` is set on the joining member, for example because you specify the replication user credentials on the `START GROUP_REPLICATION` statement, you must issue `START GROUP_REPLICATION` manually again following this restart. If `group_replication_start_on_boot=ON` and other settings required to start Group Replication were set in a configuration file or using a `SET PERSIST` statement, you do not need to intervene and the process continues automatically to bring the joining member online.

If the remote cloning procedure takes a long time, in releases before MySQL 8.0.22, it is possible for the set of certification information that accumulates for the group during that time to become too large to transmit to the joining member. In that case, the joining member logs an error message and does not join the group. From MySQL 8.0.22, Group Replication manages the garbage collection process for applied transactions differently to avoid this scenario. In earlier releases, if you do see this error, after the remote cloning operation completes, wait two minutes to allow a round of garbage collection to take place to reduce the size of the group's certification information. Then issue the following statement on the joining member, so that it stops trying to apply the previous set of certification information:

```
RESET SLAVE FOR CHANNEL group_replication_recovery;
Or from MySQL 8.0.22:
RESET REPLICA FOR CHANNEL group_replication_recovery;
```

A remote cloning operation clones settings that are persisted in tables from the donor to the recipient, as well as the data. Group Replication manages the settings that relate specifically to Group Replication channels. Group Replication member settings that are persisted in configuration files, such as the group replication local address, are not cloned and are not changed on the joining member. Group Replication also preserves the channel settings that relate to the use of SSL, so these are unique to the individual member.

If the replication user credentials used by the donor for the `group_replication_recovery` replication channel have been stored in the replication metadata repositories using a `CHANGE REPLICATION SOURCE TO` | `CHANGE MASTER TO` statement, they are transferred to and used by the joining member after cloning, and they must be valid there. With stored credentials, all group members that received state transfer by a remote cloning operation therefore automatically receive the replication user and password for distributed recovery. If you specify the replication user credentials on the `START GROUP_REPLICATION` statement, these are used to start the remote cloning operation, but they are not transferred to and used by the joining member after cloning. If you do not want the credentials transferred to new joiners and recorded there, ensure that you unset them before remote cloning operations take place, as described in Section 18.6.3, “Securing Distributed Recovery Connections”, and use `START GROUP_REPLICATION` to supply them instead.

If a `PRIVILEGE_CHECKS_USER` account has been used to help secure the replication appliers (see Section 17.3.3.2, “Privilege Checks For Group Replication Channels”), from MySQL 8.0.19, the `PRIVILEGE_CHECKS_USER` account and related settings from the donor are cloned to the joining member. If the joining member is set to start Group Replication on boot, it automatically uses the account for privilege checks on the appropriate replication channels. (In MySQL 8.0.18, due to a number of limitations, it is recommended that you do not use a `PRIVILEGE_CHECKS_USER` account with Group Replication channels.)

Cloning for Other Purposes

Group Replication initiates and manages cloning operations for distributed recovery. Group members that have been set up to support cloning may also participate in cloning operations that a user initiates manually. For example, you might want to create a new server instance by cloning from a group member as the donor, but you do not want the new server instance to join the group immediately, or maybe not ever.

In all releases that support cloning, you can initiate a cloning operation manually involving a group member on which Group Replication is stopped. Note that because cloning requires that the active plugins on a donor and recipient must match, the Group Replication plugin must be installed and active on the other server instance, even if you do not intend that server instance to join a group. You can install the plugin by issuing this statement:

```
INSTALL PLUGIN group_replication SONAME 'group_replication.so';
```

In releases before MySQL 8.0.20, you cannot initiate a cloning operation manually if the operation involves a group member on which Group Replication is running. From MySQL 8.0.20, you can do this, provided that the cloning operation does not remove and replace the data on the recipient. The statement to initiate the cloning operation must therefore include the `DATA DIRECTORY` clause if Group Replication is running.

18.5.4.3 Configuring Distributed Recovery

Several aspects of Group Replication's distributed recovery process can be configured to suit your system.

Number of Connection Attempts

For state transfer from the binary log, Group Replication limits the number of attempts a joining member makes when trying to connect to a donor from the pool of donors. If the connection retry limit is reached without a successful connection, the distributed recovery procedure terminates with an error. Note that this limit specifies the total number of attempts that the joining member makes to connect to a donor. For example, if 2 group members are suitable donors, and the connection retry limit is set to 4, the joining member makes 2 attempts to connect to each of the donors before reaching the limit.

The default connection retry limit is 10. You can configure this setting using the `group_replication_recovery_retry_count` system variable. The following command sets the maximum number of attempts to connect to a donor to 5:

```
mysql> SET GLOBAL group_replication_recovery_retry_count= 5;
```

For remote cloning operations, this limit does not apply. Group Replication makes only one connection attempt to each suitable donor for cloning, before starting to attempt state transfer from the binary log.

Sleep Interval for Connection Attempts

For state transfer from the binary log, the `group_replication_recovery_reconnect_interval` system variable defines how much time the distributed recovery process should sleep between donor connection attempts. Note that distributed recovery does not sleep after every donor connection attempt. As the joining member is connecting to different servers and not to the same one repeatedly, it can assume that the problem that affects server A does not affect server B. Distributed recovery therefore suspends only when it has gone through all the possible donors. Once the server joining the group has made one attempt to connect to each of the suitable donors in the group, the distributed recovery process sleeps for the number of seconds configured by the `group_replication_recovery_reconnect_interval` system variable. For example, if 2 group members are suitable donors, and the connection retry limit is set to 4, the joining member makes one attempt to connect to each of the donors, then sleeps for the connection retry interval, then makes one further attempt to connect to each of the donors before reaching the limit.

The default connection retry interval is 60 seconds, and you can change this value dynamically. The following command sets the distributed recovery donor connection retry interval to 120 seconds:

```
mysql> SET GLOBAL group_replication_recovery_reconnect_interval= 120;
```

For remote cloning operations, this interval does not apply. Group Replication makes only one connection attempt to each suitable donor for cloning, before starting to attempt state transfer from the binary log.

Marking the Joining Member Online

When distributed recovery has successfully completed state transfer from the donor to the joining member, the joining member can be marked as online in the group and ready to participate. By default, this is done after the joining member has received and applied all the transactions that it was missing. Optionally, you can allow a joining member to be marked as online when it has received and certified (that is, completed conflict detection for) all the transactions that it was missing, but before it has applied them. If you want to do this, use the `group_replication_recovery_complete_at` system variable to specify the alternative setting `TRANSACTIONS_CERTIFIED`.

18.5.4.4 Fault Tolerance for Distributed Recovery

Group Replication's distributed recovery process has a number of built-in measures to ensure fault tolerance in the event of any problems during the process.

The donor for distributed recovery is selected randomly from the existing list of suitable online group members in the current view. Selecting a random donor means that there is a good chance that the same server is not selected more than once when multiple members enter the group. From MySQL 8.0.17, for state transfer from the binary log, the joiner only selects a donor that is running a lower or equal patch version of MySQL Server compared to itself. For earlier releases, all of the online members are allowed to be a donor. For a remote cloning operation, the joiner only selects a donor that is running the same patch version as itself. Note that when the joining member has restarted at the end of the operation, it establishes a connection with a new donor for state transfer from the binary log, which might be a different member from the original donor used for the remote cloning operation.

In the following situations, Group Replication detects an error in distributed recovery, automatically switches over to a new donor, and retries the state transfer:

- *Connection error* - There is an authentication issue or another problem with making the connection to a candidate donor.
- *Replication errors* - One of the replication threads (the receiver or applier threads) being used for state transfer from the binary log fails. Because this method of state transfer uses the existing MySQL replication framework, it is possible that some transient errors could cause errors in the receiver or applier threads.
- *Remote cloning operation errors* - A remote cloning operation fails or is stopped before it completes.
- *Donor leaves the group* - The donor leaves the group, or Group Replication is stopped on the donor, while state transfer is in progress.

The Performance Schema table `replication_applier_status_by_worker` displays the error that caused the last retry. In these situations, the new connection following the error is attempted with a new candidate donor. Selecting a different donor in the event of an error means that there is a chance the new candidate donor does not have the same error. If the clone plugin is installed, Group Replication attempts a remote cloning operation with each of the suitable online clone-supporting donors first. If all those attempts fail, Group Replication attempts state transfer from the binary log with all the suitable donors in turn, if that is possible.

Warning

For a remote cloning operation, user-created tablespaces and data on the recipient (the joining member) are dropped before the remote cloning operation begins to transfer the data from the donor. If the remote cloning operation starts but does not complete, the joining member might be left with a partial set of its original data files, or with no user data. Data transferred by the donor is removed from the recipient if the cloning operation is stopped before the data is fully cloned. This situation can be repaired by retrying the cloning operation, which Group Replication does automatically.

In the following situations, the distributed recovery process cannot be completed, and the joining member leaves the group:

- *Purged transactions* - Transactions that are required by the joining member are not present in any online group member's binary log files, and the data cannot be obtained by a remote cloning operation (because the clone plugin is not installed, or because cloning was attempted with all possible donors but failed). The joining member is therefore unable to catch up with the group.
- *Extra transactions* - The joining member already contains some transactions that are not present in the group. If a remote cloning operation was carried out, these transactions would be deleted and lost, because the data directory on the joining member is erased. If state transfer from a donor's binary log was carried out, these transactions could conflict with the group's transactions. For advice on dealing with this situation, see Extra Transactions.
- *Connection retry limit reached* - The joining member has made all the connection attempts allowed by the connection retry limit. You can configure this using the `group_replication_recovery_retry_count` system variable (see Section 18.5.4.3, “Configuring Distributed Recovery”).
- *No more donors* - The joining member has unsuccessfully attempted a remote cloning operation with each of the online clone-supporting donors in turn (if the clone plugin is installed), then has unsuccessfully attempted state transfer from the binary log with each of the suitable online donors in turn, if possible.
- *Joining member leaves the group* - The joining member leaves the group or Group Replication is stopped on the joining member while state transfer is in progress.

If the joining member left the group unintentionally, so in any situation listed above except the last, it proceeds to take the action specified by the `group_replication_exit_state_action` system variable.

18.5.4.5 How Distributed Recovery Works

When Group Replication's distributed recovery process is carrying out state transfer from the binary log, to synchronize the joining member with the donor up to a specific point in time, the joining member and donor make use of GTIDs (see Section 17.1.3, “Replication with Global Transaction Identifiers”). However, GTIDs only provide a means to realize which transactions the joining member is missing. They do not help marking a specific point in time to which the server joining the group must catch up, nor do they convey certification information. This is the job of binary log view markers, which mark view changes in the binary log stream, and also contain additional metadata information, supplying the joining member with missing certification-related data.

This topic explains the role of view changes and the view change identifier, and the steps to carry out state transfer from the binary log.

View and View Changes

A *view* corresponds to a group of members participating actively in the current configuration, in other words at a specific point in time. They are functioning correctly and online in the group.

A *view change* occurs when a modification to the group configuration happens, such as a member joining or leaving. Any group membership change results in an independent view change communicated to all members at the same logical point in time.

A *view identifier* uniquely identifies a view. It is generated whenever a view change happens.

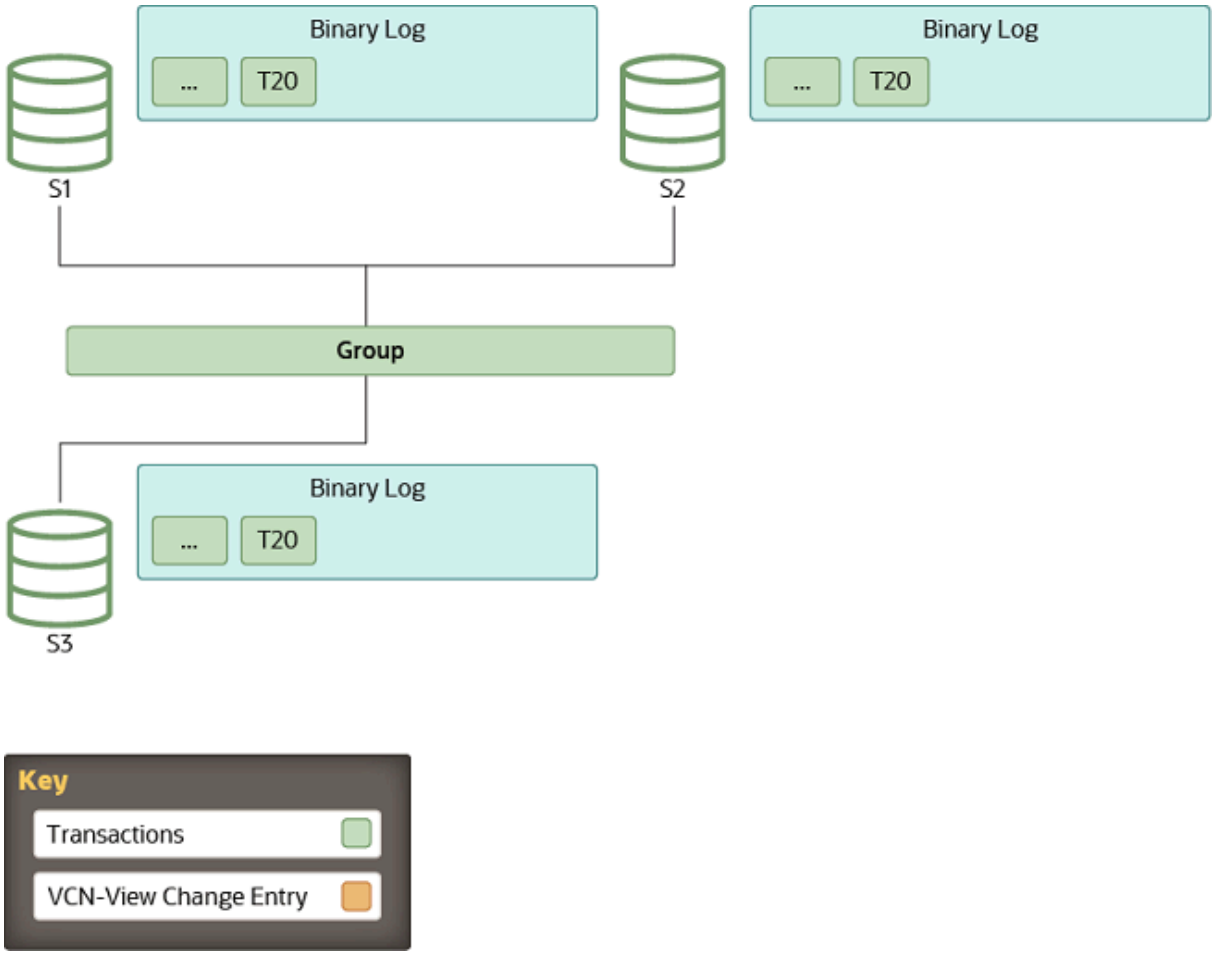
At the group communication layer, view changes with their associated view identifiers mark boundaries between the data exchanged before and after a member joins. This concept is implemented through a binary log event: the "view change log event" (VCLE). The view identifier is recorded to demarcate transactions transmitted before and after changes happen in the group membership.

The view identifier itself is built from two parts: a randomly generated part, and a monotonically increasing integer. The randomly generated part is generated when the group is created, and remains unchanged while there is at least one member in the group. The integer is incremented every time a view change happens. Using these two different parts enables the view identifier to identify incremental group changes caused by members joining or leaving, and also to identify the situation where all members leave the group in a full group shutdown, so no information remains of what view the group was in. Randomly generating part of the identifier when the group is started from the beginning ensures that the data markers in the binary log remain unique, and an identical identifier is not reused after a full group shutdown, as this would cause issues with distributed recovery in the future.

Begin: Stable Group

All servers are online and processing incoming transactions from the group. Some servers may be a little behind in terms of transactions replicated, but eventually they converge. The group acts as one distributed and replicated database.

Figure 18.8 Stable Group

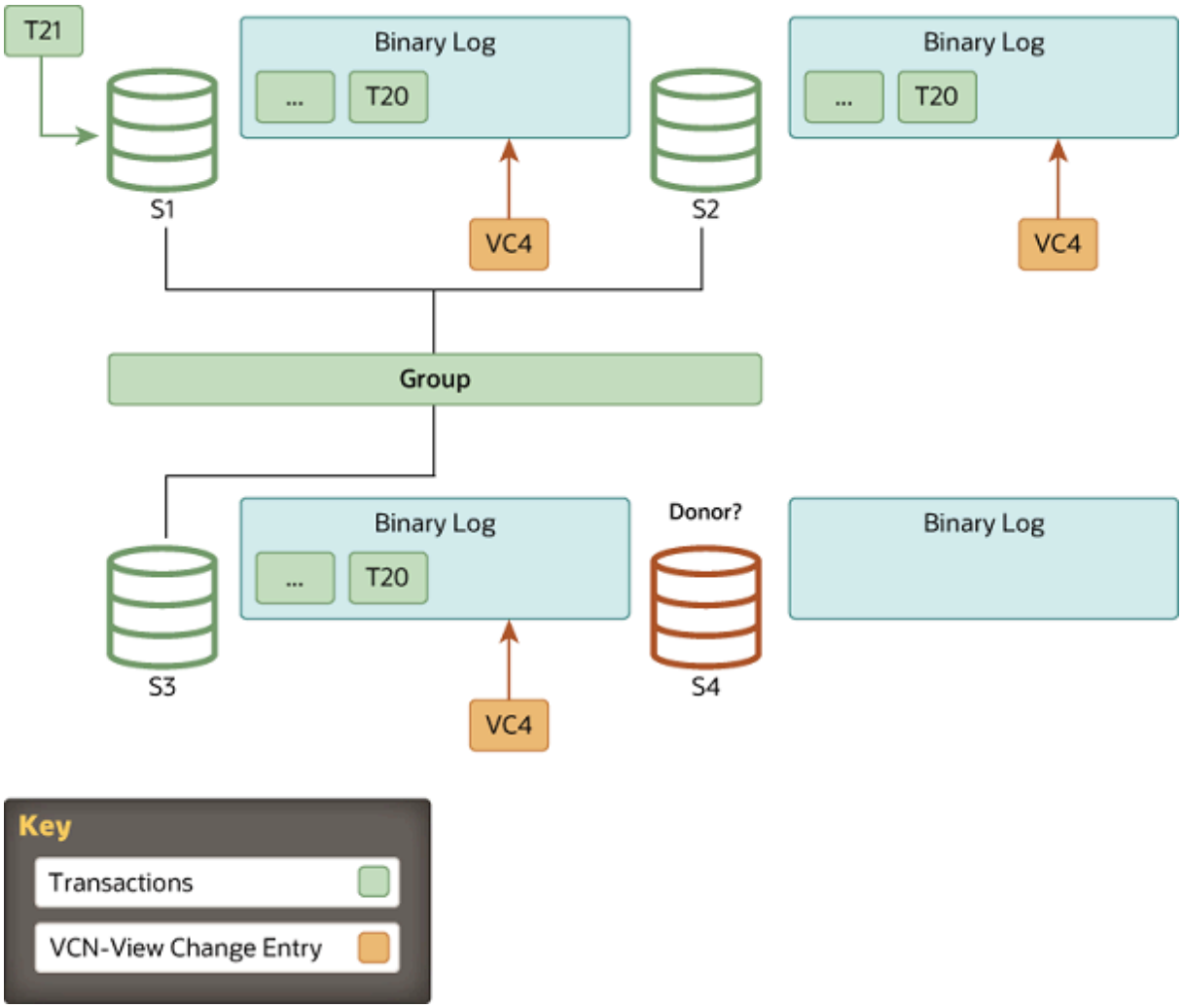


View Change: a Member Joins

Whenever a new member joins the group and therefore a view change is performed, every online server queues a view change log event for execution. This is queued because before the view change, several transactions can be queued on the server to be applied and as such, these belong to the old view. Queuing the view change event after them guarantees a correct marking of when this happened.

Meanwhile, the joining member selects a suitable donor from the list of online servers as stated by the membership service through the view abstraction. A member joins on view 4 and the online members write a view change event to the binary log.

Figure 18.9 A Member Joins

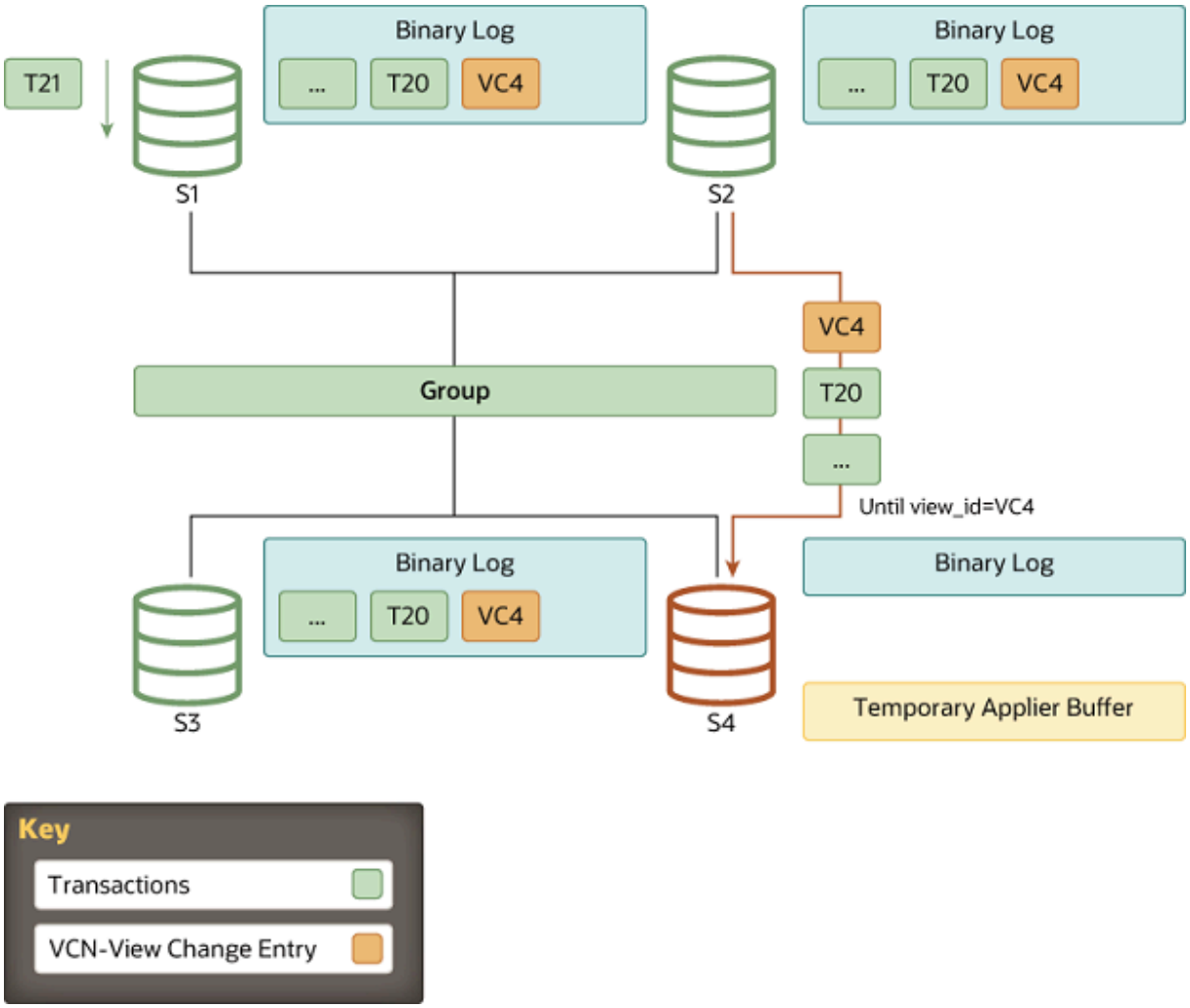


State Transfer: Catching Up

If group members and the joining member are set up with the clone plugin (see Section 18.5.4.2, "Cloning for Distributed Recovery"), and the difference in transactions between the joining member and the group exceeds the threshold set for a remote cloning operation (`group_replication_clone_threshold`), Group Replication begins distributed recovery with a remote cloning operation. A remote cloning operation is also carried out if required transactions are no longer present in any group member's binary log files. During a remote cloning operation, the existing data on the joining member is removed, and replaced with a copy of the donor's data. When the remote cloning operation is complete and the joining member has restarted, state transfer from a donor's binary log is carried out to get the transactions that the group applied while the remote cloning operation was in progress. If there is not a large transaction gap, or if the clone plugin is not installed, Group Replication proceeds directly to state transfer from a donor's binary log.

For state transfer from a donor's binary log, a connection is established between the joining member and the donor and state transfer begins. This interaction with the donor continues until the server joining the group's applier thread processes the view change log event that corresponds to the view change triggered when the server joining the group came into the group. In other words, the server joining the group replicates from the donor, until it gets to the marker with the view identifier which matches the view marker it is already in.

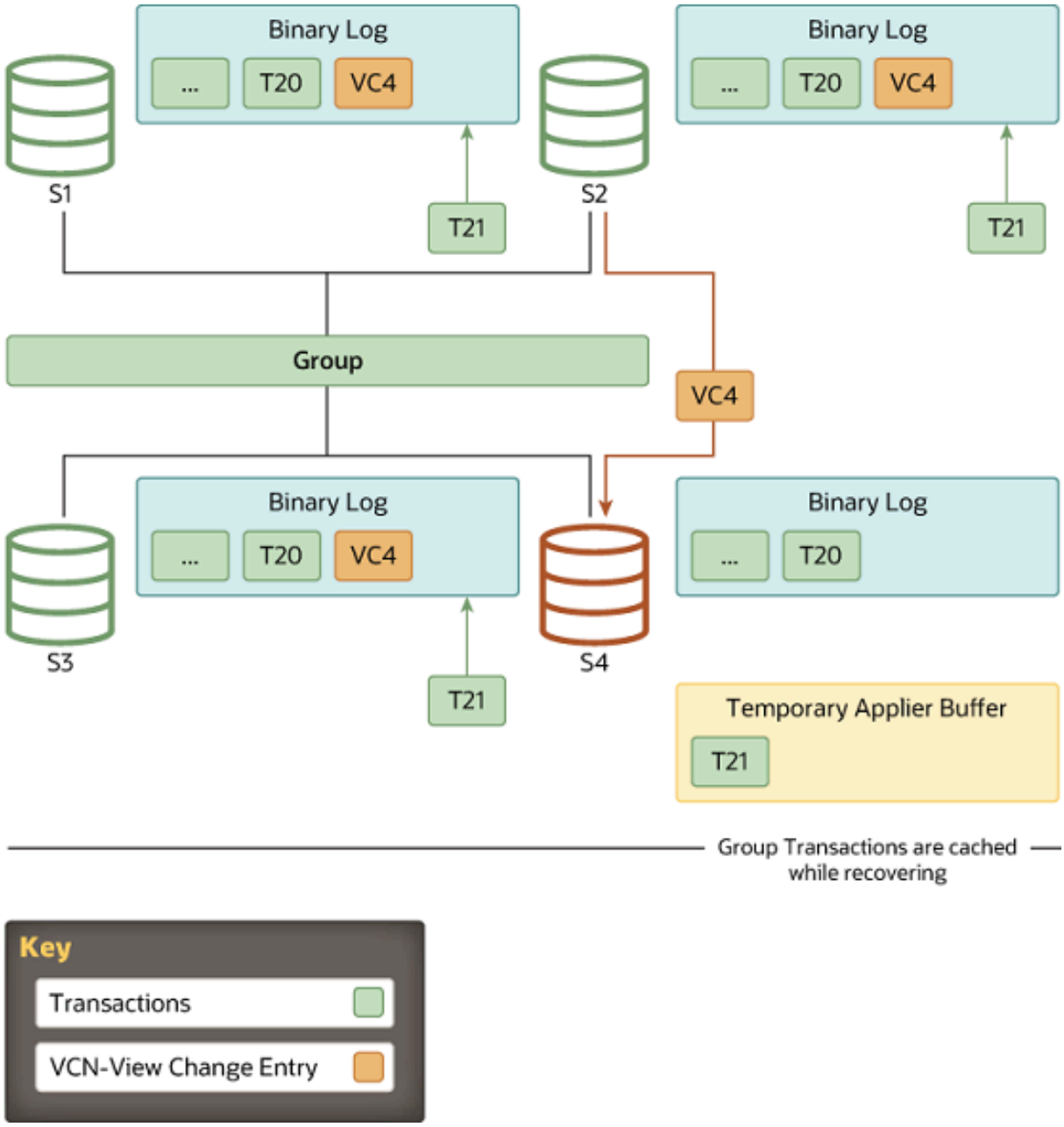
Figure 18.10 State Transfer: Catching Up



As view identifiers are transmitted to all members in the group at the same logical time, the server joining the group knows at which view identifier it should stop replicating. This avoids complex GTID set calculations because the view identifier clearly marks which data belongs to each group view.

While the server joining the group is replicating from the donor, it is also caching incoming transactions from the group. Eventually, it stops replicating from the donor and switches to applying those that are cached.

Figure 18.11 Queued Transactions



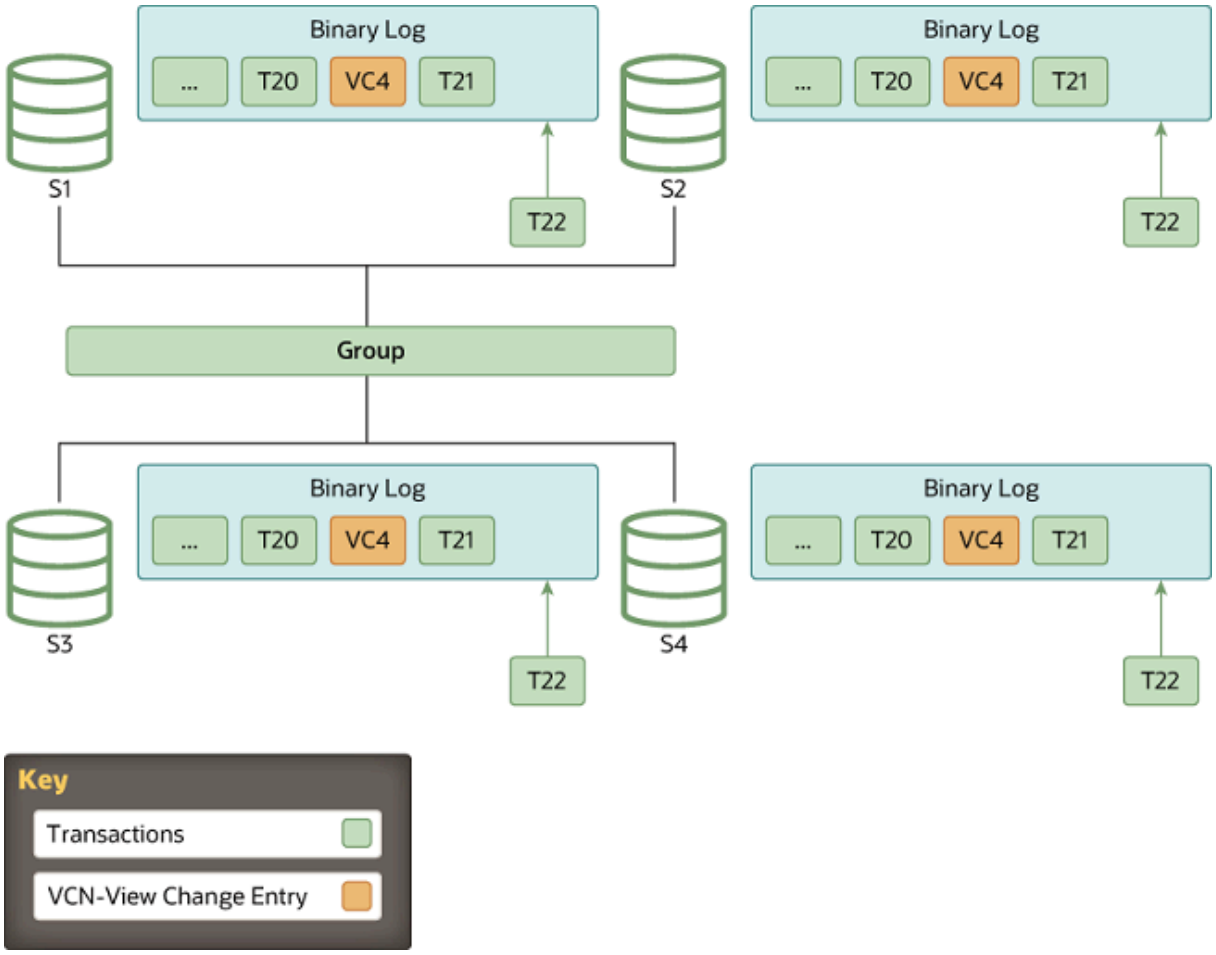
Finish: Caught Up

When the server joining the group recognizes a view change log event with the expected view identifier, the connection to the donor is terminated and it starts applying the cached transactions. Although it acts as a marker in the binary log, delimiting view changes, the view change log event also plays another role. It conveys the certification information as perceived by all servers when the server joining the group entered the group, in other words the last view change. Without it, the server joining the group would not have the necessary information to be able to certify (detect conflicts) subsequent transactions.

The duration of the catch up is not deterministic, because it depends on the workload and the rate of incoming transactions to the group. This process is completely online and the server joining the group does not block any other server in the group while it is catching up. Therefore the number of transactions the server joining the group is behind when it moves to this stage can, for this reason, vary and thus increase or decrease according to the workload.

When the server joining the group reaches zero queued transactions and its stored data is equal to the other members, its public state changes to online.

Figure 18.12 Instance Online



18.5.5 Support For IPv6 And For Mixed IPv6 And IPv4 Groups

From MySQL 8.0.14, Group Replication group members can use IPv6 addresses as an alternative to IPv4 addresses for communications within the group. To use IPv6 addresses, the operating system on the server host and the MySQL Server instance must both be configured to support IPv6. For instructions to set up IPv6 support for a server instance, see Section 5.1.13, “IPv6 Support”.

IPv6 addresses, or host names that resolve to them, can be specified as the network address that the member provides in the `group_replication_local_address` option for connections from other members. When specified with a port number, an IPv6 address must be specified in square brackets, for example:

```
group_replication_local_address= "[2001:db8:85a3:8d3:1319:8a2e:370:7348]:33061"
```

The network address or host name specified in `group_replication_local_address` is used by Group Replication as the unique identifier for a group member within the replication group. If a host name specified as the Group Replication local address for a server instance resolves to both an IPv4 and an IPv6 address, the IPv4 address is always used for Group Replication connections. The address or host name specified as the Group Replication local address is not the same as the MySQL server SQL protocol host and port, and is not specified in the `bind_address` system variable for the server instance. For the purpose of IP address permissions for Group Replication (see Section 18.6.4, “Group Replication IP Address Permissions”), the address that you specify for each group member in `group_replication_local_address` must

be added to the list for the `group_replication_ip_allowlist` (from MySQL 8.0.22) or `group_replication_ip_whitelist` system variable on the other servers in the replication group.

A replication group can contain a combination of members that present an IPv6 address as their Group Replication local address, and members that present an IPv4 address. When a server joins such a mixed group, it must make the initial contact with the seed member using the protocol that the seed member advertises in the `group_replication_group_seeds` option, whether that is IPv4 or IPv6. If any of the seed members for the group are listed in the `group_replication_group_seeds` option with an IPv6 address when a joining member has an IPv4 Group Replication local address, or the reverse, you must also set up and permit an alternative address for the joining member for the required protocol (or a host name that resolves to an address for that protocol). If a joining member does not have a permitted address for the appropriate protocol, its connection attempt is refused. The alternative address or host name only needs to be added to the `group_replication_ip_allowlist` (from MySQL 8.0.22) or `group_replication_ip_whitelist` system variable on the other servers in the replication group, not to the `group_replication_local_address` value for the joining member (which can only contain a single address).

For example, server A is a seed member for a group, and has the following configuration settings for Group Replication, so that it is advertising an IPv6 address in the `group_replication_group_seeds` option:

```
group_replication_bootstrap_group=on
group_replication_local_address= "[2001:db8:85a3:8d3:1319:8a2e:370:7348]:33061"
group_replication_group_seeds= "[2001:db8:85a3:8d3:1319:8a2e:370:7348]:33061"
```

Server B is a joining member for the group, and has the following configuration settings for Group Replication, so that it has an IPv4 Group Replication local address:

```
group_replication_bootstrap_group=off
group_replication_local_address= "203.0.113.21:33061"
group_replication_group_seeds= "[2001:db8:85a3:8d3:1319:8a2e:370:7348]:33061"
```

Server B also has an alternative IPv6 address `2001:db8:8b0:40:3d9c:cc43:e006:19e8`. For Server B to join the group successfully, both its IPv4 Group Replication local address, and its alternative IPv6 address, must be listed in Server A's allowlist, as in the following example:

```
group_replication_ip_allowlist=
"203.0.113.0/24,2001:db8:85a3:8d3:1319:8a2e:370:7348,
2001:db8:8b0:40:3d9c:cc43:e006:19e8"
```

As a best practice for Group Replication IP address permissions, Server B (and all other group members) should have the same allowlist as Server A, unless security requirements demand otherwise.

If any or all members of a replication group are using an older MySQL Server version that does not support the use of IPv6 addresses for Group Replication, a member cannot participate in the group using an IPv6 address (or a host name that resolves to one) as its Group Replication local address. This applies both in the case where at least one existing member uses an IPv6 address and a new member that does not support this attempts to join, and in the case where a new member attempts to join using an IPv6 address but the group includes at least one member that does not support this. In each situation, the new member cannot join. To make a joining member present an IPv4 address for group communications, you can either change the value of `group_replication_local_address` to an IPv4 address, or configure your DNS to resolve the joining member's existing host name to an IPv4 address. After you have upgraded every group member to a MySQL Server version that supports IPv6 for Group Replication, you can change the `group_replication_local_address` value for each member to an IPv6 address, or configure your DNS to present an IPv6 address. Changing the value of `group_replication_local_address` takes effect only when you stop and restart Group Replication.

IPv6 addresses can also be used as distributed recovery endpoints, which can be specified from MySQL 8.0.21 using the `group_replication_advertise_recovery_endpoints` system

variable. The same rules apply to addresses used in this list. See Section 18.5.4.1, “Connections for Distributed Recovery”.

18.5.6 Using MySQL Enterprise Backup with Group Replication

MySQL Enterprise Backup is a commercially-licensed backup utility for MySQL Server, available with MySQL Enterprise Edition. This section explains how to back up and subsequently restore a Group Replication member using MySQL Enterprise Backup. The same technique can be used to quickly add a new member to a group.

Backing up a Group Replication Member Using MySQL Enterprise Backup

Backing up a Group Replication member is similar to backing up a stand-alone MySQL instance. The following instructions assume that you are already familiar with how to use MySQL Enterprise Backup to perform a backup; if that is not the case, please review the MySQL Enterprise Backup 8.0 User's Guide, especially Backing Up a Database Server. Also note the requirements described in Grant MySQL Privileges to Backup Administrator and Using MySQL Enterprise Backup with Group Replication.

Consider the following group with three members, `s1`, `s2`, and `s3`, running on hosts with the same names:

```
mysql> SELECT member_host, member_port, member_state FROM performance_schema.replication_group_members;
+-------------+-------------+--------------+
| member_host | member_port | member_state |
+-------------+-------------+--------------+
| s1          |        3306 | ONLINE       |
| s2          |        3306 | ONLINE       |
| s3          |        3306 | ONLINE       |
+-------------+-------------+--------------+
```

Using MySQL Enterprise Backup, create a backup of `s2` by issuing on its host, for example, the following command:

```
s2> mysqlbackup --defaults-file=/etc/my.cnf --backup-image=/backups/my.mbi_`date +%d%m_%H%M` \
        --backup-dir=/backups/backup_`date +%d%m_%H%M` --user=root -p \
        --host=127.0.0.1 backup-to-image
```

Notes

- *For MySQL Enterprise Backup 8.0.18 and earlier,* If the system variable `sql_require_primary_key` is set to `ON` for the group, MySQL Enterprise Backup is not able to log the backup progress on the servers. This is because the `backup_progress` table on the server is a CSV table, for which primary keys are not supported. In that case, `mysqlbackup` issues the following warnings during the backup operation:

  ```
  181011 11:17:06 MAIN WARNING: MySQL query 'CREATE TABLE IF NOT EXISTS
  mysql.backup_progress( `backup_id` BIGINT NOT NULL, `tool_name` VARCHAR(4096)
  NOT NULL, `error_code` INT NOT NULL, `error_message` VARCHAR(4096) NOT NULL,
  `current_time` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP                ON
  UPDATE CURRENT_TIMESTAMP,`current_state` VARCHAR(200) NOT NULL ) ENGINE=CSV
  DEFAULT CHARSET=utf8mb3 COLLATE=utf8mb3_bin': 3750, Unable to create a table
  without PK, when system variable 'sql_require_primary_key' is set. Add a PK
  to the table or unset this variable to avoid this message. Note that tables
  without PK can cause performance problems in row-based replication, so please
  consult your DBA before changing this setting.
  181011 11:17:06 MAIN WARNING: This backup operation's progress info cannot be
  logged.
  ```

  This does not prevent `mysqlbackup` from finishing the backup.

- *For MySQL Enterprise Backup 8.0.20 and earlier*, when backing up a secondary member, as MySQL Enterprise Backup cannot write backup status

and metadata to a read-only server instance, it might issue warnings similar to the following one during the backup operation:

```
181113 21:31:08 MAIN WARNING: This backup operation cannot write to backup
progress. The MySQL server is running with the --super-read-only option.
```

You can avoid the warning by using the `--no-history-logging` option with your backup command. This is not an issue for MySQL Enterprise Backup 8.0.21 and higher—see Using MySQL Enterprise Backup with Group Replication for details.

Restoring a Failed Member

Assume one of the members (`s3` in the following example) is irreconcilably corrupted. The most recent backup of group member `s2` can be used to restore `s3`. Here are the steps for performing the restore:

1. *Copy the backup of s2 onto the host for s3.* The exact way to copy the backup depends on the operating system and tools available to you. In this example, we assume the hosts are both Linux servers and use SCP to copy the files between them:

   ```
   s2/backups> scp my.mbi_2206_1429 s3:/backups
   ```

2. *Restore the backup.* Connect to the target host (the host for `s3` in this case), and restore the backup using MySQL Enterprise Backup. Here are the steps:

   a. Stop the corrupted server, if it is still running. For example, on Linux distributions that use systemd:

   ```
   s3> systemctl stop mysqld
   ```

   b. Preserve the two configuration files in the corrupted server's data directory, `auto.cnf` and `mysqld-auto.cnf` (if it exists), by copying them to a safe location outside of the data directory. This is for preserving the server's UUID and Section 5.1.9.3, “Persisted System Variables” (if used), which are needed in the steps below.

   c. Delete all contents in the data directory of `s3`. For example:

   ```
   s3> rm -rf /var/lib/mysql/*
   ```

   If the system variables `innodb_data_home_dir`, `innodb_log_group_home_dir`, and `innodb_undo_directory` point to any directories other than the data directory, they should also be made empty; otherwise, the restore operation fails.

   d. Restore backup of `s2` onto the host for `s3`:

   ```
   s3> mysqlbackup --defaults-file=/etc/my.cnf \
     --datadir=/var/lib/mysql \
     --backup-image=/backups/my.mbi_2206_1429  \
     --backup-dir=/tmp/restore_`date +%d%m_%H%M` copy-back-and-apply-log
   ```

Note

The command above assumes that the binary logs and relay logs on `s2` and `s3` have the same base name and are at the same location on the two servers. If these conditions are not met, you should use the `--log-bin` and `--relay-log` options to restore the binary log and relay log to their original file paths on `s3`. For example, if you know that on `s3` the binary log's base name is `s3-bin` and the relay-log's base name is `s3-relay-bin`, your restore command should look like:

```
mysqlbackup --defaults-file=/etc/my.cnf \
  --datadir=/var/lib/mysql \
  --backup-image=/backups/my.mbi_2206_1429  \
  --log-bin=s3-bin --relay-log=s3-relay-bin \
```

```
  --backup-dir=/tmp/restore_`date +%d%m_%H%M` copy-back-and-apply-log
```

Being able to restore the binary log and relay log to the right file paths makes the restore process easier; if that is impossible for some reason, see Rebuild the Failed Member to Rejoin as a New Member.

3. *Restore the `auto.cnf` file for s3.* To rejoin the replication group, the restored member *must* have the same `server_uuid` it used to join the group before. Supply the old server UUID by copying the `auto.cnf` file preserved in step 2 above into the data directory of the restored member.

Note

If you cannot supply the failed member's original `server_uuid` to the restored member by restoring its old `auto.cnf` file, you must let the restored member join the group as a new member; see instructions in Rebuild the Failed Member to Rejoin as a New Member below on how to do that.

4. *Restore the `mysqld-auto.cnf` file for s3 (only required if s3 used persistent system variables).* The settings for the Section 5.1.9.3, “Persisted System Variables” that were used to configure the failed member must be provided to the restored member. These settings are to be found in the `mysqld-auto.cnf` file of the failed server, which you should have preserved in step 2 above. Restore the file to the data directory of the restored server. See Restoring Persisted System Variables on what to do if you do not have a copy of the file.

5. *Start the restored server.* For example, on Linux distributions that use systemd:

```
systemctl start mysqld
```

Note

If the server you are restoring is a primary member, perform the steps described in Restoring a Primary Member *before starting the restored server*.

6. *Restart Group Replication.* Connect to the restarted `s3` using, for example, a `mysql` client, and issue the following command:

```
mysql> START GROUP_REPLICATION;
```

Before the restored instance can become an online member of the group, it needs to apply any transactions that have happened to the group after the backup was taken; this is achieved using Group Replication's distributed recovery mechanism, and the process starts after the START GROUP_REPLICATION statement has been issued. To check the member status of the restored instance, issue:

```
mysql> SELECT member_host, member_port, member_state FROM performance_schema.replication_group_members;
+-------------+-------------+--------------+
| member_host | member_port | member_state |
+-------------+-------------+--------------+
| s1          |        3306 | ONLINE       |
| s2          |        3306 | ONLINE       |
| s3          |        3306 | RECOVERING   |
+-------------+-------------+--------------+
```

This shows that `s3` is applying transactions to catch up with the group. Once it has caught up with the rest of the group, its `member_state` changes to `ONLINE`:

```
mysql> SELECT member_host, member_port, member_state FROM performance_schema.replication_group_members;
+-------------+-------------+--------------+
| member_host | member_port | member_state |
+-------------+-------------+--------------+
| s1          |        3306 | ONLINE       |
| s2          |        3306 | ONLINE       |
```

```
| s3          |        3306 | ONLINE       |
+-------------+-------------+--------------+
```

Note

If the server you are restoring is a primary member, once it has gained synchrony with the group and become `ONLINE`, perform the steps described at the end of Restoring a Primary Member to revert the configuration changes you had made to the server before you started it.

The member has now been fully restored from the backup and functions as a regular member of the group.

Rebuild the Failed Member to Rejoin as a New Member

Sometimes, the steps outlined above in Restoring a Failed Member cannot be carried out because, for example, the binary log or relay log is corrupted, or it is just missing from the backup. In such a situation, use the backup to rebuild the member, and then add it to the group as a new member. In the steps below, we assume the rebuilt member is named `s3`, like the failed member, and that it runs on the same host as `s3`:

1. *Copy the backup of s2 onto the host for s3 .* The exact way to copy the backup depends on the operating system and tools available to you. In this example we assume the hosts are both Linux servers and use SCP to copy the files between them:

   ```
   s2/backups> scp my.mbi_2206_1429 s3:/backups
   ```

2. *Restore the backup.* Connect to the target host (the host for `s3` in this case), and restore the backup using MySQL Enterprise Backup. Here are the steps:

   a. Stop the corrupted server, if it is still running. For example, on Linux distributions that use systemd:

   ```
   s3> systemctl stop mysqld
   ```

   b. Preserve the configuration file `mysqld-auto.cnf`, if it is found in the corrupted server's data directory, by copying it to a safe location outside of the data directory. This is for preserving the server's Section 5.1.9.3, “Persisted System Variables”, which are needed later.

   c. Delete all contents in the data directory of `s3`. For example:

   ```
   s3> rm -rf /var/lib/mysql/*
   ```

   If the system variables `innodb_data_home_dir`, `innodb_log_group_home_dir`, and `innodb_undo_directory` point to any directories other than the data directory, they should also be made empty; otherwise, the restore operation fails.

   d. Restore the backup of `s2` onto the host of `s3`. With this approach, we are rebuilding `s3` as a new member, for which we do not need or do not want to use the old binary and relay logs in the backup; therefore, if these logs have been included in your backup, exclude them using the `--skip-binlog` and `--skip-relaylog` options:

   ```
   s3> mysqlbackup --defaults-file=/etc/my.cnf \
     --datadir=/var/lib/mysql \
     --backup-image=/backups/my.mbi_2206_1429  \
     --backup-dir=/tmp/restore_`date +%d%m_%H%M` \
     --skip-binlog --skip-relaylog \
     copy-back-and-apply-log
   ```

   Note

   If you have healthy binary log and relay logs in the backup that you can transfer onto the target host with no issues, you are recommended to

follow the easier procedure as described in Restoring a Failed Member above.

3. *Restore the `mysqld-auto.cnf` file for s3 (only required if s3 used persistent system variables).* The settings for the Section 5.1.9.3, “Persisted System Variables” that were used to configure the failed member must be provided to the restored server. These settings are to be found in the `mysqld-auto.cnf` file of the failed server, which you should have preserved in step 2 above. Restore the file to the data directory of the restored server. See Restoring Persisted System Variables on what to do if you do not have a copy of the file.

Note

Do NOT restore the corrupted server's `auto.cnf` file to the data directory of the new member—when the rebuilt `s3` joins the group as a new member, it is going to be assigned a new server UUID.

4. *Start the restored server.* For example, on Linux distributions that use systemd:

```
systemctl start mysqld
```

Note

If the server you are restoring is a primary member, perform the steps described in Restoring a Primary Member *before starting the restored server*.

5. *Reconfigure the restored member to join Group Replication.* Connect to the restored server with a `mysql` client and reset the source and replica information with the following commands:

```
mysql> RESET MASTER;
```

```
mysql> RESET SLAVE ALL;
Or from MySQL 8.0.22:
mysql> RESET REPLICA ALL;
```

For the restored server to be able to recover automatically using Group Replication's built-in mechanism for distributed recovery, configure the server's `gtid_executed` variable. To do this, use the `backup_gtid_executed.sql` file included in the backup of `s2`, which is usually restored under the restored member's data directory. Disable binary logging, use the `backup_gtid_executed.sql` file to configure `gtid_executed`, and then re-enable binary logging by issuing the following statements with your `mysql` client:

```
mysql> SET SQL_LOG_BIN=OFF;
mysql> SOURCE datadir/backup_gtid_executed.sql
mysql> SET SQL_LOG_BIN=ON;
```

Then, configure the Group Replication user credentials on the member:

```
mysql> CHANGE MASTER TO MASTER_USER='rpl_user', MASTER_PASSWORD='password' /
  FOR CHANNEL 'group_replication_recovery';

Or from MySQL 8.0.23:
mysql> CHANGE REPLICATION SOURCE TO SOURCE_USER='rpl_user', SOURCE_PASSWORD='password' /
  FOR CHANNEL 'group_replication_recovery';
```

6. *Restart Group Replication.* Issue the following command to the restored server with your `mysql` client:

```
mysql> START GROUP_REPLICATION;
```

Before the restored instance can become an online member of the group, it needs to apply any transactions that have happened to the group after the backup was taken; this is achieved using Group Replication's distributed recovery mechanism, and the process starts after the START

GROUP_REPLICATION statement has been issued. To check the member status of the restored instance, issue:

```
mysql> SELECT member_host, member_port, member_state FROM performance_schema.replication_group_membe
+-------------+-------------+--------------+
| member_host | member_port | member_state |
+-------------+-------------+--------------+
| s3          |        3306 | RECOVERING   |
| s2          |        3306 | ONLINE       |
| s1          |        3306 | ONLINE       |
+-------------+-------------+--------------+
```

This shows that `s3` is applying transactions to catch up with the group. Once it has caught up with the rest of the group, its `member_state` changes to `ONLINE`:

```
mysql> SELECT member_host, member_port, member_state FROM performance_schema.replication_group_membe
+-------------+-------------+--------------+
| member_host | member_port | member_state |
+-------------+-------------+--------------+
| s3          |        3306 | ONLINE       |
| s2          |        3306 | ONLINE       |
| s1          |        3306 | ONLINE       |
+-------------+-------------+--------------+
```

Note

If the server you are restoring is a primary member, once it has gained synchrony with the group and become `ONLINE`, perform the steps described at the end of Restoring a Primary Member to revert the configuration changes you had made to the server before you started it.

The member has now been restored to the group as a new member.

Restoring Persisted System Variables. `mysqlbackup` does not provide support for backing up or preserving Section 5.1.9.3, “Persisted System Variables”—the file `mysqld-auto.cnf` is not included in a backup. To start the restored member with its persisted variable settings, you need to do one of the following:

- Preserve a copy of the `mysqld-auto.cnf` file from the corrupted server, and copy it to the restored server's data directory.
- Copy the `mysqld-auto.cnf` file from another member of the group into the restored server's data directory, if that member has the same persisted system variable settings as the corrupted member.
- After the restored server is started and before you restart Group Replication, set all the system variables manually to their persisted values through a `mysql` client.

Restoring a Primary Member. If the restored member is a primary in the group, care must be taken to prevent writes to the restored database during the Group Replication distributed recovery process. Depending on how the group is accessed by clients, there is a possibility of DML statements being executed on the restored member once it becomes accessible on the network, prior to the member finishing its catch-up on the activities it has missed while off the group. To avoid this, *before starting the restored server*, configure the following system variables in the server option file:

```
group_replication_start_on_boot=OFF
super_read_only=ON
event_scheduler=OFF
```

These settings ensure that the member becomes read-only at startup and that the event scheduler is turned off while the member is catching up with the group during the distributed recovery process. Adequate error handling must also be configured on the clients, as they are prevented temporarily from performing DML operations during this period on the restored member. Once the restore process is fully completed and the restored member is in-sync with the rest of the group, revert those changes; restart the event scheduler:

```
mysql> SET global event_scheduler=ON;
```

Edit the following system variables in the member's option file, so things are correctly configured for the next startup:

```
group_replication_start_on_boot=ON
super_read_only=OFF
event_scheduler=ON
```

18.6 Group Replication Security

This section explains how to secure a group, securing the connections between members of a group, or by establishing a security perimeter using an IP address allowlist.

18.6.1 Communication Stack for Connection Security Management

From MySQL 8.0.27, Group Replication can secure group communication connections between members by one of the following methods:

- Using its own implementation of the security protocols, including TLS/SSL and the use of an allowlist for incoming Group Communication System (GCS) connections. This is the only option for MySQL 8.0.26 and earlier.
- Using MySQL Server's own connection security in place of Group Replication's implementation. Using the MySQL protocol means that standard methods of user authentication can be used for granting (or revoking) access to the group in place of the allowlist, and the latest functionality of the server's protocol is always available on release. This option is available from MySQL 8.0.27.

The choice is made by setting the system variable `group_replication_communication_stack` to `XCOM` to use Group Replication's own implementation (this is the default choice), or to `MYSQL` to use MySQL Server's connection security.

The following additional configuration is required for a replication group to use the MySQL communication stack. It is especially important to make sure these requirements are all fulfilled when you switch from using the XCom communication stack to the MySQL communication stack for your group.

Group Replication Requirements For The MySQL Communication Stack

- The network address configured by the `group_replication_local_address` system variable for each group member must be set to one of the IP addresses and ports that MySQL Server is listening on, as specified by the `bind_address` system variable for the server. The combination of IP address and port for each member must be unique in the group. It is recommended that the `group_replication_group_seeds` system variable for each group member be configured to contain all the local addresses for all the group members.
- The MySQL communication stack supports network namespaces, which the XCom communication stack does not support. If network namespaces are used with the Group Replication local addresses for the group members (`group_replication_local_address`), these must be configured for each group member using the `CHANGE REPLICATION SOURCE TO` statement. Also, the `report_host` server system variable for each group member must be set to report the namespace. All group members must use the same namespace to avoid possible issues with address resolution during distributed recovery.
- The `group_replication_ssl_mode` system variable must be set to the required setting for group communications. This system variable controls whether TLS/SSL is enabled or disabled for group communications. For MySQL 8.0.26 and earlier, the TLS/SSL configuration is always taken from the server's SSL settings; for MySQL 8.0.27 and later, when the MySQL communication stack is used, the TLS/SSL configuration is taken from Group Replication's distributed recovery settings. This setting should be the same on all the group members, to avoid potential conflicts.
- The settings for the `--ssl` or `--skip-ssl` server option and for the `require_secure_transport` server system variable should be the same on all the group

members, to avoid potential conflicts. If `group_replication_ssl_mode` is set to `REQUIRED`, `VERIFY_CA`, or `VERIFY_IDENTITY`, use `--ssl` and `require_secure_transport=ON`. If `group_replication_ssl_mode` is set to `DISABLED`, use `require_secure_transport=OFF`.

- If TLS/SSL is enabled for group communications, Group Replication's settings for securing distributed recovery must be configured if they are not already in place, or validated if they already are. The MySQL communication stack uses these settings not just for member-to-member distributed recovery connections, but also for TLS/SSL configuration in general group communications. `group_replication_recovery_use_ssl` and the other `group_replication_recovery_*` system variables are explained in Section 18.6.3.2, "Secure Socket Layer (SSL) Connections for Distributed Recovery".

- The Group Replication allowlist is not used when the group is using the MySQL communication stack, so the `group_replication_ip_allowlist` and `group_replication_ip_whitelist` system variables are ignored and need not be configured.

- The replication user account that Group Replication uses for distributed recovery, as configured using the `CHANGE REPLICATION SOURCE TO` statement, is used for authentication by the MySQL communication stack when setting up Group Replication connections. This user account, which is the same on all group members, must be given the following privileges:

  - `GROUP_REPLICATION_STREAM`. This privilege is required for the user account to be able to establish connections for Group Replication using the MySQL communication stack.

  - `CONNECTION_ADMIN`. This privilege is required so that Group Replication connections are not terminated if one of the servers involved is placed in offline mode. If the MySQL communication stack is in use without this privilege, a member that is placed in offline mode is expelled from the group.

  These are in addition to the privileges `REPLICATION SLAVE` and `BACKUP_ADMIN` that all replication user accounts must have (see Section 18.2.1.3, "User Credentials For Distributed Recovery"). When you add the new privileges, ensure that you skip binary logging on each group member by issuing `SET SQL_LOG_BIN=0` before you issue the `GRANT` statements, and `SET SQL_LOG_BIN=1` after them, so that the local transaction does not interfere with restarting Group Replication.

`group_replication_communication_stack` is effectively a group-wide configuration setting, and the setting must be the same on all group members. However, this is not policed by Group Replication's own checks for group-wide configuration settings. A member with a different value from the rest of the group cannot communicate with the other members at all, because the communication protocols are incompatible, so it cannot exchange information about its configuration settings.

This means that although the value of the system variable can be changed while Group Replication is running, and takes effect after you restart Group Replication on the group member, the member still cannot rejoin the group until the setting has been changed on all the members. You must therefore stop Group Replication on all of the members and change the value of the system variable on them all before you can restart the group. Because all of the members are stopped, a full reboot of the group (a bootstrap by a server with `group_replication_bootstrap_group=ON`) is required in order for the value change to take effect. You can make the other required changes to settings on the group members while they are stopped.

For a running group, follow this procedure to change the value of `group_replication_communication_stack` and the other required settings to migrate a group from the XCom communication stack to the MySQL communication stack, or from the MySQL communication stack to the XCom communication stack:

1. Stop Group Replication on each of the group members, using a `STOP GROUP_REPLICATION` statement. Stop the primary member last, so that you do not trigger a new primary election and have to wait for that to complete.

2. On each of the group members, set the system variable `group_replication_communication_stack` to the new communication stack, `MYSQL` or

`XCOM` as appropriate. You can do this by editing the MySQL Server configuration file (typically named `my.cnf` on Linux and Unix systems, or `my.ini` on Windows systems), or by using a `SET` statement. For example:

```
SET PERSIST group_replication_communication_stack="MYSQL";
```

3. If you are migrating the replication group from the XCom communication stack (the default) to the MySQL communication stack, on each of the group members, configure or reconfigure the required system variables to appropriate settings, as described in the listing above. For example, the `group_replication_local_address` system variable must be set to one of the IP addresses and ports that MySQL Server is listening on. Also configure any network namespaces using a `CHANGE REPLICATION SOURCE TO` statement.

4. If you are migrating the replication group from the XCom communication stack (the default) to the MySQL communication stack, on each of the group members, issue `GRANT` statements to give the replication user account the `GROUP_REPLICATION_STREAM` and `CONNECTION_ADMIN` privileges. You will need to take the group members out of the read-only state that is applied when Group Replication is stopped. Also ensure that you skip binary logging on each group member by issuing `SET SQL_LOG_BIN=0` before you issue the `GRANT` statements, and `SET SQL_LOG_BIN=1` after them, so that the local transaction does not interfere with restarting Group Replication. For example:

```
SET GLOBAL SUPER_READ_ONLY=OFF;
SET SQL_LOG_BIN=0;
GRANT GROUP_REPLICATION_STREAM ON *.* TO rpl_user@'%';
GRANT CONNECTION_ADMIN ON *.* TO rpl_user@'%';
SET SQL_LOG_BIN=1;
```

5. If you are migrating the replication group from the MySQL communication stack back to the XCom communication stack, on each of the group members, reconfigure the system variables in the requirements listing above to settings suitable for the XCom communication stack. Section 18.9, “Group Replication System Variables” lists the system variables with their defaults and requirements for the XCom communication stack.

Note

- The XCom communication stack does not support network namespaces, so the Group Replication local address (`group_replication_local_address` system variable) cannot use these. Unset them by issuing a `CHANGE REPLICATION SOURCE TO` statement.
- When you move back to the XCom communication stack, the settings specified by `group_replication_recovery_use_ssl` and the other `group_replication_recovery_*` system variables are not used to secure group communications. Instead, the Group Replication system variable `group_replication_ssl_mode` is used to activate the use of SSL for group communication connections and specify the security mode for the connections, and the remainder of the configuration is taken from the server's SSL configuration. For details, see Section 18.6.2, “Securing Group Communication Connections with Secure Socket Layer (SSL)”.

6. To restart the group, follow the process in Section 18.5.2, “Restarting a Group”, which explains how to safely bootstrap a group where transactions have been executed and certified. A bootstrap by a server with `group_replication_bootstrap_group=ON` is necessary to change the communication stack, because all of the members must be shut down.

7. Members now connect to each other using the new communication stack. Any server that has `group_replication_communication_stack` set (or defaulted, in the case of XCom) to the previous communication stack is no longer able to join the group. It is important to note that because Group Replication cannot even see the joining attempt, it does not check and reject the

joining member with an error message. Instead, the attempted join fails silently when the previous communication stack gives up trying to contact the new one.

18.6.2 Securing Group Communication Connections with Secure Socket Layer (SSL)

Secure sockets can be used for group communication connections between members of a group.

The Group Replication system variable `group_replication_ssl_mode` is used to activate the use of SSL for group communication connections and specify the security mode for the connections. The default setting means that SSL is not used. The option has the following possible values:

Table 18.1 group_replication_ssl_mode configuration values

Value	Description
`DISABLED`	Establish an unencrypted connection (the default).
`REQUIRED`	Establish a secure connection if the server supports secure connections.
`VERIFY_CA`	Like `REQUIRED`, but additionally verify the server TLS certificate against the configured Certificate Authority (CA) certificates.
`VERIFY_IDENTITY`	Like `VERIFY_CA`, but additionally verify that the server certificate matches the host to which the connection is attempted.

If SSL is used, the means for configuring the secure connection depends on whether the XCom or the MySQL communication stack is used for group communication (a choice between the two is available since MySQL 8.0.27).

When using the XCom communication stack (`group_replication_communication_stack=XCOM`): The remainder of the configuration for Group Replication's group communication connections is taken from the server's SSL configuration. For more information on the options for configuring the server SSL, see Command Options for Encrypted Connections. The server SSL options that are applied to Group Replication's group communication connections are as follows:

Table 18.2 SSL Options

Server Configuration	Description
`ssl_key`	The path name of the SSL private key file in PEM format. On the client side, this is the client private key. On the server side, this is the server private key.
`ssl_cert`	The path name of the SSL public key certificate file in PEM format. On the client side, this is the client public key certificate. On the server side, this is the server public key certificate.
`ssl_ca`	The path name of the Certificate Authority (CA) certificate file in PEM format.
`ssl_capath`	The path name of the directory that contains trusted SSL certificate authority (CA) certificate files in PEM format.
`ssl_crl`	The path name of the file containing certificate revocation lists in PEM format.

Server Configuration	Description
`ssl_crlpath`	The path name of the directory that contains certificate revocation list files in PEM format.
`ssl_cipher`	A list of permissible ciphers for encrypted connections.
`tls_version`	A list of the TLS protocols the server permits for encrypted connections.
`tls_ciphersuites`	Which TLSv1.3 ciphersuites the server permits for encrypted connections.

Important

- Support for the TLSv1 and TLSv1.1 connection protocols is removed from MySQL Server as of MySQL 8.0.28. The protocols were deprecated from MySQL 8.0.26, though MySQL Server clients, including Group Replication server instances acting as a client, do not return warnings to the user if a deprecated TLS protocol version is used. See Removal of Support for the TLSv1 and TLSv1.1 Protocols for more information.
- Support for the TLSv1.3 protocol is available in MySQL Server as of MySQL 8.0.16, provided that MySQL Server was compiled using OpenSSL 1.1.1. The server checks the version of OpenSSL at startup, and if it is lower than 1.1.1, TLSv1.3 is removed from the default value for the server system variables relating to TLS versions (including the `group_replication_recovery_tls_version` system variable).
- Group Replication supports TLSv1.3 from MySQL 8.0.18. In MySQL 8.0.16 and MySQL 8.0.17, if the server supports TLSv1.3, the protocol is not supported in the group communication engine and cannot be used by Group Replication.
- In MySQL 8.0.18, TLSv1.3 can be used in Group Replication for the distributed recovery connection, but the `group_replication_recovery_tls_version` and `group_replication_recovery_tls_ciphersuites` system variables are not available. The donor servers must therefore permit the use of at least one TLSv1.3 ciphersuite that is enabled by default, as listed in Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”. From MySQL 8.0.19, you can use the options to configure client support for any selection of ciphersuites, including only non-default ciphersuites if you want.
- In the list of TLS protocols specified in the `tls_version` system variable, ensure the specified versions are contiguous (for example, `TLSv1.2,TLSv1.3`). If there are any gaps in the list of protocols (for example, if you specified `TLSv1,TLSv1.2`, omitting TLS 1.1) Group Replication might be unable to make group communication connections.

In a replication group, OpenSSL negotiates the use of the highest TLS protocol that is supported by all members. A joining member that is configured to use only TLSv1.3 (`tls_version=TLSv1.3`) cannot join a replication group where any existing member does not support TLSv1.3, because the group members in that case are using a lower TLS protocol version. To join the member to the group, you must configure the joining member to also permit the use of lower TLS protocol versions supported by the existing group members. Conversely, if a joining member does not support TLSv1.3, but the existing group members all do and are using that version for connections to each other, the member can join if the existing group members already permit the use of a suitable lower TLS protocol version, or if you configure them to do so. In that situation, OpenSSL uses a lower TLS protocol version for the

connections from each member to the joining member. Each member's connections to other existing members continue to use the highest available protocol that both members support.

From MySQL 8.0.16, you can change the `tls_version` system variable at runtime to alter the list of permitted TLS protocol versions for the server. Note that for Group Replication, the `ALTER INSTANCE RELOAD TLS` statement, which reconfigures the server's TLS context from the current values of the system variables that define the context, does not change the TLS context for Group Replication's group communication connection while Group Replication is running. To apply the reconfiguration to these connections, you must execute `STOP GROUP_REPLICATION` followed by `START GROUP_REPLICATION` to restart Group Replication on the member or members where you changed the `tls_version` system variable. Similarly, if you want to make all members of a group change to using a higher or lower TLS protocol version, you must carry out a rolling restart of Group Replication on the members after changing the list of permitted TLS protocol versions, so that OpenSSL negotiates the use of the higher TLS protocol version when the rolling restart is completed. For instructions to change the list of permitted TLS protocol versions at runtime, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers” and Server-Side Runtime Configuration and Monitoring for Encrypted Connections.

The following example shows a section from a `my.cnf` file that configures SSL on a server, and activates SSL for Group Replication group communication connections:

```
[mysqld]
ssl_ca = "cacert.pem"
ssl_capath = "/.../ca_directory"
ssl_cert = "server-cert.pem"
ssl_cipher = "DHE-RSA-AEs256-SHA"
ssl_crl = "crl-server-revoked.crl"
ssl_crlpath = "/.../crl_directory"
ssl_key = "server-key.pem"
group_replication_ssl_mode= REQUIRED
```

Important

The `ALTER INSTANCE RELOAD TLS` statement, which reconfigures the server's TLS context from the current values of the system variables that define the context, does not change the TLS context for Group Replication's group communication connections while Group Replication is running. To apply the reconfiguration to these connections, you must execute `STOP GROUP_REPLICATION` followed by `START GROUP_REPLICATION` to restart Group Replication.

Connections made between a joining member and an existing member for distributed recovery are not covered by the options described above. These connections use Group Replication's dedicated distributed recovery SSL options, which are described in Section 18.6.3.2, “Secure Socket Layer (SSL) Connections for Distributed Recovery”.

When using the MySQL communication stack (group_replication_communication_stack=MYSQL): The security settings for distributed recovery of the group are applied to the normal communications between group members. See Section 18.6.3, “Securing Distributed Recovery Connections” on how to configure the security settings.

18.6.3 Securing Distributed Recovery Connections

Important

When using the MySQL communication stack (`group_replication_communication_stack=MYSQL`) AND secure connections between members (`group_replication_ssl_mode` is not set to `DISABLED`), the security settings discussed in this section are applied not just to distributed recovery connections, but to group communications between members in general.

When a member joins the group, distributed recovery is carried out using a combination of a remote cloning operation, if available and appropriate, and an asynchronous replication connection. For a full description of distributed recovery, see Section 18.5.4, “Distributed Recovery”.

Up to MySQL 8.0.20, group members offer their standard SQL client connection to joining members for distributed recovery, as specified by MySQL Server's `hostname` and `port` system variables. From MySQL 8.0.21, group members may advertise an alternative list of distributed recovery endpoints as dedicated client connections for joining members. For more details, see Section 18.5.4.1, “Connections for Distributed Recovery”. Notice that such connections offered to a joining member for distributed recovery is *not* the same connections that are used by Group Replication for communication between online members when the XCom communication stack is used for group communications (`group_replication_communication_stack=XCOM`).

To secure distributed recovery connections in the group, ensure that user credentials for the replication user are properly secured, and use SSL for distributed recovery connections if possible.

18.6.3.1 Secure User Credentials for Distributed Recovery

State transfer from the binary log requires a replication user with the correct permissions so that Group Replication can establish direct member-to-member replication channels. The same replication user is used for distributed recovery on all the group members. If group members have been set up to support the use of a remote cloning operation as part of distributed recovery, which is available from MySQL 8.0.17, this replication user is also used as the clone user on the donor, and requires the correct permissions for this role too. For detailed instructions to set up this user, see Section 18.2.1.3, “User Credentials For Distributed Recovery”.

To secure the user credentials, you can require SSL for connections with the user account, and (from MySQL 8.0.21) you can provide the user credentials when Group Replication is started, rather than storing them in the replica status tables. Also, if you are using caching SHA-2 authentication, you must set up RSA key-pairs on the group members.

Important

When using the MySQL communication stack (`group_replication_communication_stack=MYSQL`) AND secure connections between members (`group_replication_ssl_mode` is not set to `DISABLED`), the recovery users must be properly set up, as they are also the users for group communications. Follow the instructions in Replication User With SSL and Providing Replication User Credentials Securely.

Replication User With The Caching SHA-2 Authentication Plugin

By default, users created in MySQL 8 use Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”. If the replication user you configure for distributed recovery uses the caching SHA-2 authentication plugin, and you are *not* using SSL for distributed recovery connections, RSA key-pairs are used for password exchange. For more information on RSA key-pairs, see Section 6.3.3, “Creating SSL and RSA Certificates and Keys”.

In this situation, you can either copy the public key of the `rpl_user` to the joining member, or configure the donors to provide the public key when requested. The more secure approach is to copy the public key of the replication user account to the joining member. Then you need to configure the `group_replication_recovery_public_key_path` system variable on the joining member with the path to the public key for the replication user account.

The less secure approach is to set `group_replication_recovery_get_public_key=ON` on donors so that they provide the public key of the replication user account to joining members. There is no way to verify the identity of a server, therefore only set `group_replication_recovery_get_public_key=ON` when you are sure there is no risk of server identity being compromised, for example by a man-in-the-middle attack.

Replication User With SSL

A replication user that requires an SSL connection must be created *before* the server joining the group (the joining member) connects to the donor. Typically, this is set up at the time you are provisioning a server to join the group. To create a replication user for distributed recovery that requires an SSL connection, issue these statements on all servers that are going to participate in the group:

```
mysql> SET SQL_LOG_BIN=0;
mysql> CREATE USER 'rec_ssl_user'@'%' IDENTIFIED BY 'password' REQUIRE SSL;
mysql> GRANT REPLICATION SLAVE ON *.* TO 'rec_ssl_user'@'%';
mysql> GRANT CONNECTION_ADMIN ON *.* TO 'rec_ssl_user'@'%';
mysql> GRANT BACKUP_ADMIN ON *.* TO 'rec_ssl_user'@'%';
mysql> GRANT GROUP_REPLICATION_STREAM ON *.* TO rpl_user@'%';
mysql> FLUSH PRIVILEGES;
mysql> SET SQL_LOG_BIN=1;
```

Note

The `GROUP_REPLICATION_STREAM` privilege is required when using both the MySQL communication stack (`group_replication_communication_stack=MYSQL`) and secure connections between members (`group_replication_ssl_mode` not set to `DISABLED`). See Section 18.6.1, “Communication Stack for Connection Security Management”.

Providing Replication User Credentials Securely

To supply the user credentials for the replication user, you can set them permanently as the credentials for the `group_replication_recovery` channel, using a `CHANGE REPLICATION SOURCE TO` | `CHANGE MASTER TO` statement. Alternatively, from MySQL 8.0.21, you can specify them on the `START GROUP_REPLICATION` statement each time Group Replication is started. User credentials specified on `START GROUP_REPLICATION` take precedence over any user credentials that have been set using a `CHANGE REPLICATION SOURCE TO` | `CHANGE MASTER TO` statement.

User credentials set using `CHANGE REPLICATION SOURCE TO` | `CHANGE MASTER TO` are stored in plain text in the replication metadata repositories on the server, but user credentials specified on `START GROUP_REPLICATION` are saved in memory only, and are removed by a `STOP GROUP_REPLICATION` statement or server shutdown. Using `START GROUP_REPLICATION` to specify the user credentials therefore helps to secure the Group Replication servers against unauthorized access. However, this method is not compatible with starting Group Replication automatically, as specified by the `group_replication_start_on_boot` system variable.

If you want to set the user credentials permanently using a `CHANGE REPLICATION SOURCE TO` | `CHANGE MASTER TO` statement, issue this statement on the member that is going to join the group:

```
mysql> CHANGE MASTER TO MASTER_USER='rec_ssl_user', MASTER_PASSWORD='password'
            FOR CHANNEL 'group_replication_recovery';

Or from MySQL 8.0.23:
mysql> CHANGE REPLICATION SOURCE TO SOURCE_USER='rec_ssl_user', SOURCE_PASSWORD='password'
            FOR CHANNEL 'group_replication_recovery';
```

To supply the user credentials on `START GROUP_REPLICATION`, issue this statement when starting Group Replication for the first time, or after a server restart:

```
mysql> START GROUP_REPLICATION USER='rec_ssl_user', PASSWORD='password';
```

Important

If you switch to using `START GROUP_REPLICATION` to specify user credentials on a server that previously supplied the credentials using `CHANGE REPLICATION SOURCE TO` | `CHANGE MASTER TO`, you must complete the following steps to get the security benefits of this change.

1. Stop Group Replication on the group member using a `STOP GROUP_REPLICATION` statement. Although it is possible to take the following two steps while Group Replication is running, you need to restart Group Replication to implement the changes.

2. Set the value of the `group_replication_start_on_boot` system variable to `OFF` (the default is `ON`).

3. Remove the distributed recovery credentials from the replica status tables by issuing this statement:

```
mysql> CHANGE MASTER TO MASTER_USER='', MASTER_PASSWORD=''
           FOR CHANNEL 'group_replication_recovery';

Or from MySQL 8.0.23:
mysql> CHANGE REPLICATION SOURCE TO SOURCE_USER='', SOURCE_PASSWORD=''
           FOR CHANNEL 'group_replication_recovery';
```

4. Restart Group Replication on the group member using a `START GROUP_REPLICATION` statement that specifies the distributed recovery user credentials.

Without these steps, the credentials remain stored in the replica status tables, and can also be transferred to other group members during remote cloning operations for distributed recovery. The `group_replication_recovery` channel could then be inadvertently started with the stored credentials, on either the original member or members that were cloned from it. An automatic start of Group Replication on server boot (including after a remote cloning operation) would use the stored user credentials, and they would also be used if an operator did not specify the distributed recovery credentials as part of `START GROUP_REPLICATION`.

18.6.3.2 Secure Socket Layer (SSL) Connections for Distributed Recovery

Important

When using the MySQL communication stack (`group_replication_communication_stack=MYSQL`) AND secure connections between members (`group_replication_ssl_mode` is not set to `DISABLED`), the security settings discussed in this section are applied not just to distributed recovery connections, but to group communications between members in general. See Section 18.6.1, “Communication Stack for Connection Security Management”.

Whether the distributed recovery connection is made using the standard SQL client connection or a distributed recovery endpoint, to configure the connection securely, you can use Group Replication's dedicated distributed recovery SSL options. These options correspond to the server SSL options that are used for group communication connections, but they are only applied for distributed recovery connections. By default, distributed recovery connections do not use SSL, even if you activated SSL for group communication connections, and the server SSL options are not applied for distributed recovery connections. You must configure these connections separately.

If a remote cloning operation is used as part of distributed recovery, Group Replication automatically configures the clone plugin's SSL options to match your settings for the distributed recovery SSL options. (For details of how the clone plugin uses SSL, see Configuring an Encrypted Connection for Cloning.)

The distributed recovery SSL options are as follows:

- `group_replication_recovery_use_ssl`: Set to `ON` to make Group Replication use SSL for distributed recovery connections, including remote cloning operations and state transfer from a donor's binary log. You can just set this option and none of the other distributed recovery SSL options, in which case the server automatically generates certificates to use for the connection, and uses the default cipher suites. If you want to configure the certificates and cipher suites for the connection, use the other distributed recovery SSL options to do this.

- `group_replication_recovery_ssl_ca`: The path name of the Certificate Authority (CA) file to use for distributed recovery connections. Group Replication automatically configures the clone SSL option `clone_ssl_ca` to match this.

  `group_replication_recovery_ssl_capath`: The path name of a directory that contains trusted SSL certificate authority (CA) certificate files.

- `group_replication_recovery_ssl_cert`: The path name of the SSL public key certificate file to use for distributed recovery connections. Group Replication automatically configures the clone SSL option `clone_ssl_cert` to match this.

- `group_replication_recovery_ssl_key`: The path name of the SSL private key file to use for distributed recovery connections. Group Replication automatically configures the clone SSL option `clone_ssl_cert` to match this.

- `group_replication_recovery_ssl_verify_server_cert`: Makes the distributed recovery connection check the server's Common Name value in the donor sent certificate. Setting this option to `ON` is the equivalent for distributed recovery connections of setting `VERIFY_IDENTITY` for the `group_replication_ssl_mode` option for group communication connections.

- `group_replication_recovery_ssl_crl`: The path name of a file containing certificate revocation lists.

- `group_replication_recovery_ssl_crlpath`: The path name of a directory containing certificate revocation lists.

- `group_replication_recovery_ssl_cipher`: A list of permissible ciphers for connection encryption for the distributed recovery connection. Specify a list of one or more cipher names, separated by colons. For information about which encryption ciphers MySQL supports, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

- `group_replication_recovery_tls_version`: A comma-separated list of one or more permitted TLS protocols for connection encryption when this server instance is the client in the distributed recovery connection, that is, the joining member. The default for this system variable depends on the TLS protocol versions supported in the MySQL Server release. The group members involved in each distributed recovery connection as the client (joining member) and server (donor) negotiate the highest protocol version that they are both set up to support. This system variable is available from MySQL 8.0.19.

- `group_replication_recovery_tls_ciphersuites`: A colon-separated list of one or more permitted ciphersuites when TLSv1.3 is used for connection encryption for the distributed recovery connection, and this server instance is the client in the distributed recovery connection, that is, the joining member. If this system variable is set to `NULL` when TLSv1.3 is used (which is the default if you do not set the system variable), the ciphersuites that are enabled by default are allowed, as listed in Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”. If this system variable is set to the empty string, no cipher suites are allowed, and TLSv1.3 is therefore not used. This system variable is available beginning with MySQL 8.0.19.

18.6.4 Group Replication IP Address Permissions

When and only when the XCom communication stack is used for establishing group communications (`group_replication_communication_stack=XCOM`), the Group Replication plugin lets you specify an allowlist of hosts from which an incoming Group Communication System connection can be accepted. If you specify an allowlist on a server s1, then when server s2 is establishing a connection to s1 for the purpose of engaging group communication, s1 first checks the allowlist before accepting the connection from s2. If s2 is in the allowlist, then s1 accepts the connection, otherwise s1 rejects the connection attempt by s2. Beginning with MySQL 8.0.22, the system variable `group_replication_ip_allowlist` is used to specify the allowlist, and for releases before MySQL 8.0.22, the system variable `group_replication_ip_whitelist` is used. The new system variable works in the same way as the old system variable, only the terminology has changed.

Note

When the MySQL communication stack is used for establishing group communications (`group_replication_communication_stack=MYSQL`), the settings for `group_replication_ip_allowlist` and `group_replication_ip_whitelist` are ignored. See Section 18.6.1, "Communication Stack for Connection Security Management".

If you do not specify an allowlist explicitly, the group communication engine (XCom) automatically scans active interfaces on the host, and identifies those with addresses on private subnetworks, together with the subnet mask that is configured for each interface. These addresses, and the `localhost` IP address for IPv4 and (from MySQL 8.0.14) IPv6 are used to create an automatic Group Replication allowlist. The automatic allowlist therefore includes any IP addresses that are found for the host in the following ranges after the appropriate subnet mask has been applied:

```
IPv4 (as defined in RFC 1918)
10/8 prefix       (10.0.0.0 - 10.255.255.255) - Class A
172.16/12 prefix  (172.16.0.0 - 172.31.255.255) - Class B
192.168/16 prefix (192.168.0.0 - 192.168.255.255) - Class C

IPv6 (as defined in RFC 4193 and RFC 5156)
fc00:/7 prefix    - unique-local addresses
fe80::/10 prefix  - link-local unicast addresses

127.0.0.1 - localhost for IPv4
::1       - localhost for IPv6
```

An entry is added to the error log stating the addresses that have been allowed automatically for the host.

The automatic allowlist of private addresses cannot be used for connections from servers outside the private network, so a server, even if it has interfaces on public IPs, does not by default allow Group Replication connections from external hosts. For Group Replication connections between server instances that are on different machines, you must provide public IP addresses and specify these as an explicit allowlist. If you specify any entries for the allowlist, the private and `localhost` addresses are not added automatically, so if you use any of these, you must specify them explicitly.

To specify an allowlist manually, use the `group_replication_ip_allowlist` (MySQL 8.0.22 and later) or `group_replication_ip_whitelist` system variable. Before MySQL 8.0.24, you cannot change the allowlist on a server while it is an active member of a replication group. If the member is active, you must execute `STOP GROUP_REPLICATION` before changing the allowlist, and `START GROUP_REPLICATION` afterwards. From MySQL 8.0.24, you can change the allowlist while Group Replication is running.

The allowlist must contain the IP address or host name that is specified in each member's `group_replication_local_address` system variable. This address is not the same as the MySQL server SQL protocol host and port, and is not specified in the `bind_address` system variable for the server instance. If a host name used as the Group Replication local address for a server instance resolves to both an IPv4 and an IPv6 address, the IPv4 address is preferred for Group Replication connections.

IP addresses specified as distributed recovery endpoints, and the IP address for the member's standard SQL client connection if that is used for distributed recovery (which is the default), do not need to be added to the allowlist. The allowlist is only for the address specified by `group_replication_local_address` for each member. A joining member must have its initial connection to the group permitted by the allowlist in order to retrieve the address or addresses for distributed recovery.

In the allowlist, you can specify any combination of the following:

- IPv4 addresses (for example, `198.51.100.44`)

- IPv4 addresses with CIDR notation (for example, `192.0.2.21/24`)
- IPv6 addresses, from MySQL 8.0.14 (for example, `2001:db8:85a3:8d3:1319:8a2e:370:7348`)
- IPv6 addresses with CIDR notation, from MySQL 8.0.14 (for example, `2001:db8:85a3:8d3::/64`)
- Host names (for example, `example.org`)
- Host names with CIDR notation (for example, `www.example.com/24`)

Before MySQL 8.0.14, host names could only resolve to IPv4 addresses. From MySQL 8.0.14, host names can resolve to IPv4 addresses, IPv6 addresses, or both. If a host name resolves to both an IPv4 and an IPv6 address, the IPv4 address is always used for Group Replication connections. You can use CIDR notation in combination with host names or IP addresses to permit a block of IP addresses with a particular network prefix, but do ensure that all the IP addresses in the specified subnet are under your control.

Note

When a connection attempt from an IP address is refused because the address is not in the allowlist, the refusal message always prints the IP address in IPv6 format. IPv4 addresses are preceded by `::ffff:` in this format (an IPV4-mapped IPv6 address). You do not need to use this format to specify IPv4 addresses in the allowlist; use the standard IPv4 format for them.

A comma must separate each entry in the allowlist. For example:

```
mysql> SET GLOBAL group_replication_ip_allowlist="192.0.2.21/24,198.51.100.44,203.0.113.0/24,2001:db8:8
```

To join a replication group, a server needs to be permitted on the seed member to which it makes the request to join the group. Typically, this would be the bootstrap member for the replication group, but it can be any of the servers listed by the `group_replication_group_seeds` option in the configuration for the server joining the group. If any of the seed members for the group are listed in the `group_replication_group_seeds` option with an IPv6 address when a joining member has an IPv4 `group_replication_local_address`, or the reverse, you must also set up and permit an alternative address for the joining member for the protocol offered by the seed member (or a host name that resolves to an address for that protocol). This is because when a server joins a replication group, it must make the initial contact with the seed member using the protocol that the seed member advertises in the `group_replication_group_seeds` option, whether that is IPv4 or IPv6. If a joining member does not have a permitted address for the appropriate protocol, its connection attempt is refused. For more information on managing mixed IPv4 and IPv6 replication groups, see Section 18.5.5, “Support For IPv6 And For Mixed IPv6 And IPv4 Groups”.

When a replication group is reconfigured (for example, when a new primary is elected or a member joins or leaves), the group members re-establish connections between themselves. If a group member is only permitted by servers that are no longer part of the replication group after the reconfiguration, it is unable to reconnect to the remaining servers in the replication group that do not permit it. To avoid this scenario entirely, specify the same allowlist for all servers that are members of the replication group.

Note

It is possible to configure different allowlists on different group members according to your security requirements, for example, in order to keep different subnets separate. If you need to configure different allowlists to meet your security requirements, ensure that there is sufficient overlap between the allowlists in the replication group to maximize the possibility of servers being able to reconnect in the absence of their original seed member.

For host names, name resolution takes place only when a connection request is made by another server. A host name that cannot be resolved is not considered for allowlist validation, and a warning

message is written to the error log. Forward-confirmed reverse DNS (FCrDNS) verification is carried out for resolved host names.

Warning

Host names are inherently less secure than IP addresses in an allowlist. FCrDNS verification provides a good level of protection, but can be compromised by certain types of attack. Specify host names in your allowlist only when strictly necessary, and ensure that all components used for name resolution, such as DNS servers, are maintained under your control. You can also implement name resolution locally using the hosts file, to avoid the use of external components.

18.7 Group Replication Performance and Troubleshooting

Group Replication is designed to create fault-tolerant systems with built-in failure detection and automated recovery. If a member server instance leaves voluntarily or stops communicating with the group, the remaining members agree a reconfiguration of the group between themselves, and choose a new primary if needed. Expelled members automatically attempt to rejoin the group, and are brought up to date by distributed recovery. If a group experiences a level of difficulties such that it cannot contact a majority of its members in order to agree on a decision, it identifies itself as having lost quorum and stops processing transactions. Group Replication also has built-in mechanisms and settings to help the group adapt to and manage variations in workload and message size, and stay within the limitations of the underlying system and networking resources.

The default settings for Group Replication's system variables are designed to maximize a group's performance and autonomy. The information in this section is to help you configure a replication group to optimize the automatic handling of any recurring issues that you experience on your particular systems, such as transient network outages or workloads and transactions that exceed a server instance's resources.

If you find that group members are being expelled and rejoining the group more frequently than you would like, it is possible that Group Replication's default failure detection settings are too sensitive for your system. This might be the case on slower networks or machines, networks with a high rate of unexpected transient outages, or during planned network outages. For advice on dealing with that situation by adjusting the settings, see Section 18.7.7, "Responses to Failure Detection and Network Partitioning".

You should only need to intervene manually in a Group Replication setup if something happens that the group cannot deal with automatically. Some key issues that can require administrator intervention are when a member is in `ERROR` status and cannot rejoin the group, or when a network partition causes the group to lose quorum.

- If an otherwise correctly functioning and configured member is unable to join or rejoin the group using distributed recovery, and remains in `ERROR` status, Section 18.5.4.4, "Fault Tolerance for Distributed Recovery" explains the possible issues. One likely cause is that the joining member has extra transactions that are not present on the existing members of the group. For advice on dealing with that situation, see Section 18.4.1, "GTIDs and Group Replication".
- If a group has lost quorum, this may be due to a network partition that divides the group into two parts, or possibly due to the failure of the majority of the servers. For advice on dealing with that situation, see Section 18.7.8, "Handling a Network Partition and Loss of Quorum".

18.7.1 Fine Tuning the Group Communication Thread

The group communication thread (GCT) runs in a loop while the Group Replication plugin is loaded. The GCT receives messages from the group and from the plugin, handles quorum and failure detection related tasks, sends out some keep alive messages and also handles the incoming and outgoing

transactions from/to the server/group. The GCT waits for incoming messages in a queue. When there are no messages, the GCT waits. By configuring this wait to be a little longer (doing an active wait) before actually going to sleep can prove to be beneficial in some cases. This is because the alternative is for the operating system to switch out the GCT from the processor and do a context switch.

To force the GCT to do an active wait, use the `group_replication_poll_spin_loops` option, which makes the GCT loop, doing nothing relevant for the configured number of loops, before actually polling the queue for the next message.

For example:

```
mysql> SET GLOBAL group_replication_poll_spin_loops= 10000;
```

18.7.2 Flow Control

Group Replication ensures that a transaction only commits after a majority of the members in a group have received it and agreed on the relative order between all transactions that were sent concurrently. This approach works well if the total number of writes to the group does not exceed the write capacity of any member in the group. If it does and some of the members have less write throughput than others, particularly less than the writer members, those members can start lagging behind of the writers.

Having some members lagging behind the group brings some problematic consequences, particularly, the reads on such members may externalize very old data. Depending on why the member is lagging behind, other members in the group may have to save more or less replication context to be able to fulfil potential data transfer requests from the slow member.

There is however a mechanism in the replication protocol to avoid having too much distance, in terms of transactions applied, between fast and slow members. This is known as the flow control mechanism. It tries to address several goals:

1. to keep the members close enough to make buffering and de-synchronization between members a small problem;
2. to adapt quickly to changing conditions like different workloads or more writers in the group;
3. to give each member a fair share of the available write capacity;
4. to not reduce throughput more than strictly necessary to avoid wasting resources.

Given the design of Group Replication, the decision whether to throttle or not may be decided taking into account two work queues: *(i)* the *certification* queue; *(ii)* and on the binary log *applier* queue. Whenever the size of one of these queues exceeds the user-defined threshold, the throttling mechanism is triggered. Only configure: *(i)* whether to do flow control at the certifier or at the applier level, or both; and *(ii)* what is the threshold for each queue.

The flow control depends on two basic mechanisms:

1. the monitoring of members to collect some statistics on throughput and queue sizes of all group members to make educated guesses on what is the maximum write pressure each member should be subjected to;
2. the throttling of members that are trying to write beyond their fair-share of the available capacity at each moment in time.

18.7.2.1 Probes and Statistics

The monitoring mechanism works by having each member deploying a set of probes to collect information about its work queues and throughput. It then propagates that information to the group periodically to share that data with the other members.

Such probes are scattered throughout the plugin stack and allow one to establish metrics, such as:

- the certifier queue size;
- the replication applier queue size;
- the total number of transactions certified;
- the total number of remote transactions applied in the member;
- the total number of local transactions.

Once a member receives a message with statistics from another member, it calculates additional metrics regarding how many transactions were certified, applied and locally executed in the last monitoring period.

Monitoring data is shared with others in the group periodically. The monitoring period must be high enough to allow the other members to decide on the current write requests, but low enough that it has minimal impact on group bandwidth. The information is shared every second, and this period is sufficient to address both concerns.

18.7.2.2 Group Replication Throttling

Based on the metrics gathered across all servers in the group, a throttling mechanism kicks in and decides whether to limit the rate a member is able to execute/commit new transactions.

Therefore, metrics acquired from all members are the basis for calculating the capacity of each member: if a member has a large queue (for certification or the applier thread), then the capacity to execute new transactions should be close to ones certified or applied in the last period.

The lowest capacity of all the members in the group determines the real capacity of the group, while the number of local transactions determines how many members are writing to it, and, consequently, how many members should that available capacity be shared with.

This means that every member has an established write quota based on the available capacity, in other words a number of transactions it can safely issue for the next period. The writer-quota is enforced by the throttling mechanism if the queue size of the certifier or the binary log applier exceeds a user-defined threshold.

The quota is reduced by the number of transactions that were delayed in the last period, and then also further reduced by 10% to allow the queue that triggered the problem to reduce its size. In order to avoid large jumps in throughput once the queue size goes beyond the threshold, the throughput is only allowed to grow by the same 10% per period after that.

The current throttling mechanism does not penalize transactions below quota, but delays finishing those transactions that exceed it until the end of the monitoring period. As a consequence, if the quota is very small for the write requests issued some transactions may have latencies close to the monitoring period.

18.7.3 Single Consensus Leader

By default, the group communication engine for Group Replication (XCom, a Paxos variant) operates using every member of the replication group as a leader. From MySQL 8.0.27, the group communication engine can use a single leader to drive consensus when the group is in single-primary mode. Operating with a single consensus leader improves performance and resilience in single-primary mode, particularly when some of the group's secondary members are currently unreachable.

To use a single consensus leader, the group must be configured as follows:

- The group must be in single-primary mode.

- The `group_replication_paxos_single_leader` system variable must be set to `ON`. With the default setting `OFF`, the behavior is disabled. You must carry out a full reboot of the replication group (bootstrap) for Group Replication to pick up a change to this setting.
- The Group Replication communication protocol version must be set to 8.0.27 or above. Use the `group_replication_get_communication_protocol()` function to view the group's communication protocol version. If a lower version is in use, the group cannot use this behavior. You can use the `group_replication_set_communication_protocol()` function to set the group's communication protocol to a higher version if all group members support it. MySQL InnoDB Cluster manages the communication protocol version automatically. For more information, see Section 18.5.1.4, “Setting a Group's Communication Protocol Version”.

When this configuration is in place, Group Replication instructs the group communication engine to use the group’s primary as the single leader to drive consensus. When a new primary is elected, Group Replication tells the group communication engine to use it instead. If the primary is currently unhealthy, the group communication engine uses an alternative member as the consensus leader. The Performance Schema table `replication_group_communication_information` shows the current preferred and actual consensus leader, with the preferred leader being Group Replication’s choice, and the actual leader being the one selected by the group communication engine.

If the group is in multi-primary mode, has a lower communication protocol version, or the behavior is disabled by the `group_replication_paxos_single_leader` setting, all members are used as leaders to drive consensus. In this situation, the Performance Schema table `replication_group_communication_information` shows all of the members as both the preferred and actual leaders.

The field `WRITE_CONSENSUS_SINGLE_LEADER_CAPABLE` in the Performance Schema table `replication_group_communication_information` shows whether the group supports the use of a single leader, even if `group_replication_paxos_single_leader` is currently set to `OFF` on the queried member. The field is set to 1 if the group was started with `group_replication_paxos_single_leader` set to `ON`, and its communication protocol version is MySQL 8.0.27 or above. This information is only returned for group members in `ONLINE` or `RECOVERING` state.

18.7.4 Message Compression

For messages sent between online group members, Group Replication enables message compression by default. Whether a specific message is compressed depends on the threshold that you configure using the `group_replication_compression_threshold` system variable. Messages that have a payload larger than the specified number of bytes are compressed.

The default compression threshold is 1000000 bytes. You could use the following statements to increase the compression threshold to 2MB, for example:

```
STOP GROUP_REPLICATION;
SET GLOBAL group_replication_compression_threshold = 2097152;
START GROUP_REPLICATION;
```

If you set `group_replication_compression_threshold` to zero, message compression is disabled.

Group Replication uses the LZ4 compression algorithm to compress messages sent in the group. Note that the maximum supported input size for the LZ4 compression algorithm is 2113929216 bytes. This limit is lower than the maximum possible value for the `group_replication_compression_threshold` system variable, which is matched to the maximum message size accepted by XCom. The LZ4 maximum input size is therefore a practical limit for message compression, and transactions above this size cannot be committed when message compression is enabled. With the LZ4 compression algorithm, do not set a value greater than 2113929216 bytes for `group_replication_compression_threshold`.

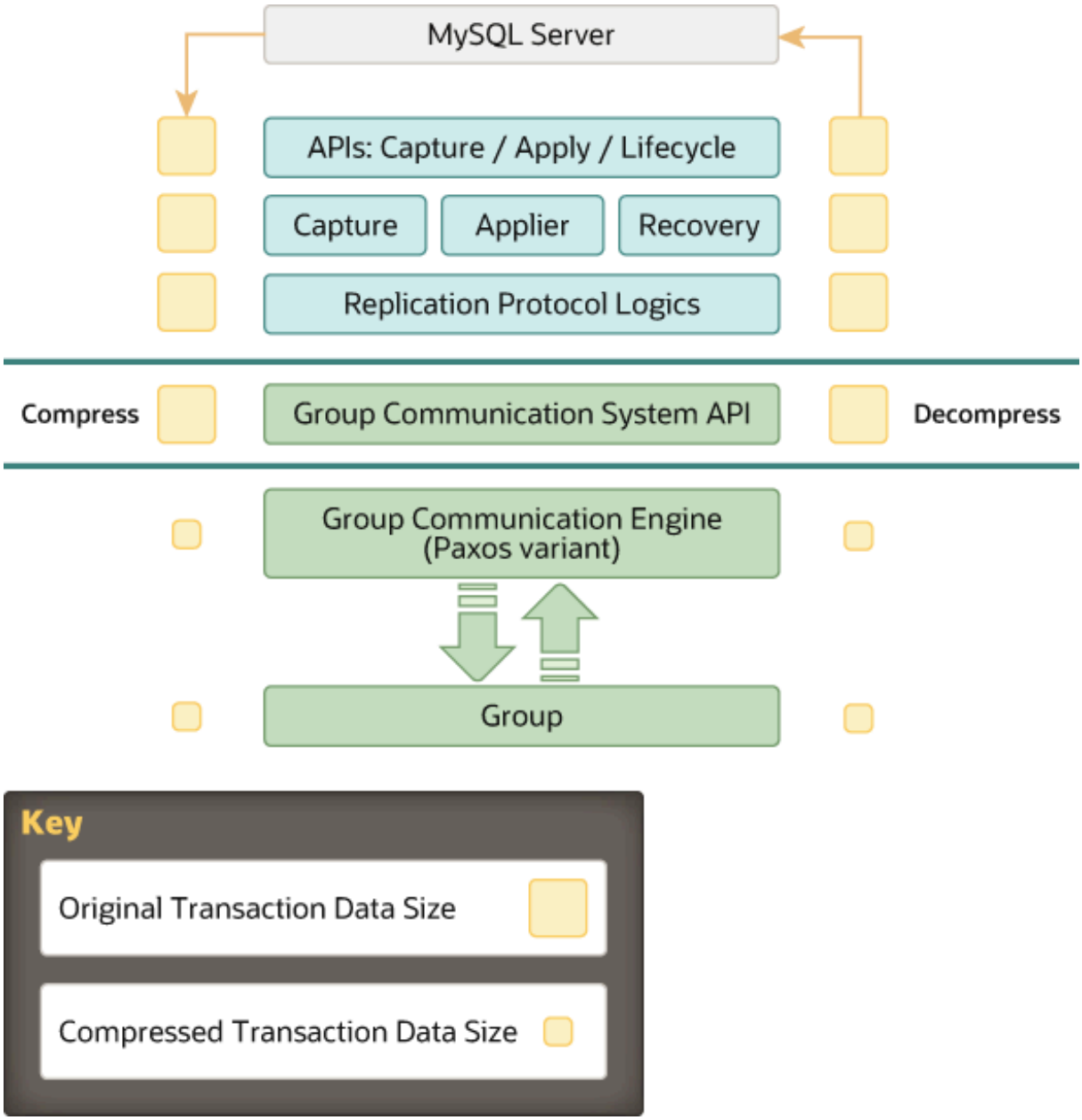
The value of `group_replication_compression_threshold` is not required by Group Replication to be the same on all group members. However, it is advisable to set the same value on all group members in order to avoid unnecessary rollback of transactions, failure of message delivery, or failure of message recovery.

From MySQL 8.0.18, you can also configure compression for messages sent for distributed recovery by the method of state transfer from a donor's binary log. Compression for these messages, which are sent from a donor already in the group to a joining member, is controlled separately using the `group_replication_recovery_compression_algorithms` and `group_replication_recovery_zstd_compression_level` system variables. For more information, see Section 4.2.8, “Connection Compression Control”.

Binary log transaction compression (available as of MySQL 8.0.20), which is activated by the `binlog_transaction_compression` system variable, can also be used to save bandwidth. The transaction payloads remain compressed when they are transferred between group members. If you use binary log transaction compression in combination with Group Replication's message compression, message compression has less opportunity to act on the data, but can still compress headers and those events and transaction payloads that are uncompressed. For more information on binary log transaction compression, see Section 5.4.4.5, “Binary Log Transaction Compression”.

Compression for messages sent in the group happens at the group communication engine level, before the data is handed over to the group communication thread, so it takes place within the context of the `mysql` user session thread. If the message payload size exceeds the threshold set by `group_replication_compression_threshold`, the transaction payload is compressed before being sent out to the group, and decompressed when it is received. Upon receiving a message, the member checks the message envelope to verify whether it is compressed or not. If needed, then the member decompresses the transaction, before delivering it to the upper layer. This process is shown in the following figure.

Figure 18.13 Compression Support



When network bandwidth is a bottleneck, message compression can provide up to 30-40% throughput improvement at the group communication level. This is especially important within the context of large groups of servers under load. The TCP peer-to-peer nature of the interconnections between *N* participants in the group makes the sender send the same amount of data *N* times. Furthermore, binary logs are likely to exhibit a high compression ratio. This makes compression a compelling feature for Group Replication workloads that contain large transactions.

18.7.5 Message Fragmentation

When an abnormally large message is sent between Group Replication group members, it can result in some group members being reported as failed and expelled from the group. This is because the single thread used by Group Replication's group communication engine (XCom, a Paxos variant) is occupied processing the message for too long, so some of the group members might report the receiver as failed. From MySQL 8.0.16, by default, large messages are automatically split into fragments that are sent separately and reassembled by the recipients.

The system variable `group_replication_communication_max_message_size` specifies a maximum message size for Group Replication communications, above which messages are fragmented. The default maximum message size is 10485760 bytes (10 MiB). The greatest permitted value is the same as the maximum value of the `replica_max_allowed_packet` and `slave_max_allowed_packet` system variables, which is 1073741824 bytes (1 GB). The setting for `group_replication_communication_max_message_size` must be less than the `replica_max_allowed_packet` or `slave_max_allowed_packet` setting, because the applier thread cannot handle message fragments larger than the maximum permitted packet size. To switch off fragmentation, specify a zero value for `group_replication_communication_max_message_size`.

As with most other Group Replication system variables, you must restart the Group Replication plugin for the change to take effect. For example:

```
STOP GROUP_REPLICATION;
SET GLOBAL group_replication_communication_max_message_size= 5242880;
START GROUP_REPLICATION;
```

Message delivery for a fragmented message is considered complete when all the fragments of the message have been received and reassembled by all the group members. Fragmented messages include information in their headers that enables a member joining during message transmission to recover the earlier fragments that were sent before it joined. If the joining member fails to recover the fragments, it expels itself from the group.

In order for a replication group to use fragmentation, all group members must be at MySQL 8.0.16 or above, and the Group Replication communication protocol version in use by the group must allow fragmentation. You can inspect the communication protocol in use by a group by using the `group_replication_get_communication_protocol()` function, which returns the oldest MySQL Server version that the group supports. Versions from MySQL 5.7.14 allow compression of messages, and versions from MySQL 8.0.16 also allow fragmentation of messages. If all group members are at MySQL 8.0.16 or above and there is no requirement to allow members at earlier releases to join, you can use the `group_replication_set_communication_protocol()` function to set the communication protocol version to MySQL 8.0.16 or above in order to allow fragmentation. For more information, see Section 18.5.1.4, “Setting a Group's Communication Protocol Version”.

If a replication group cannot use fragmentation because some members do not support it, the system variable `group_replication_transaction_size_limit` can be used to limit the maximum size of transactions the group accepts. In MySQL 8.0, the default setting is approximately 143 MB. Transactions above this size are rolled back. You can also use the system variable `group_replication_member_expel_timeout` to allow additional time (up to an hour) before a member under suspicion of having failed is expelled from the group.

18.7.6 XCom Cache Management

The group communication engine for Group Replication (XCom, a Paxos variant) includes a cache for messages (and their metadata) exchanged between the group members as a part of the consensus protocol. Among other functions, the message cache is used for recovery of missed messages by members that reconnect with the group after a period where they were unable to communicate with the other group members.

From MySQL 8.0.16, a cache size limit can be set for XCom's message cache using the `group_replication_message_cache_size` system variable. If the cache size limit is reached, XCom removes the oldest entries that have been decided and delivered. The same cache size limit should be set on all group members, because an unreachable member that is attempting to reconnect selects any other member at random for recovery of missed messages. The same messages should therefore be available in each member's cache.

Before MySQL 8.0.16, the cache size was 1 GB, and the default setting for the cache size from MySQL 8.0.16 is the same. Ensure that sufficient memory is available on your system for your chosen cache size limit, considering the size of MySQL Server's other caches and object pools. Note that the limit set

using `group_replication_message_cache_size` applies only to the data stored in the cache, and the cache structures require an additional 50 MB of memory.

When selecting a `group_replication_message_cache_size` setting, do so with reference to the expected volume of messages in the time period before a member is expelled. The length of this time period is controlled by the `group_replication_member_expel_timeout` system variable, which determines the waiting period (up to an hour) that is allowed in addition to the initial 5-second detection period for members to return to the group rather than being expelled. Note that before MySQL 8.0.21, this time period defaulted to 5 seconds from the member becoming unavailable, which is just the detection period before a suspicion is created, because the additional expel timeout set by the `group_replication_member_expel_timeout` system variable defaulted to zero. From 8.0.21 the expel timeout defaults to 5 seconds, so by default a member is not expelled until it has been absent for at least 10 seconds.

18.7.6.1 Increasing the cache size

If a member is absent for a period that is not long enough for it to be expelled from the group, it can reconnect and start participating in the group again by retrieving missed transactions from another member's XCom message cache. However, if the transactions that happened during the member's absence have been deleted from the other members' XCom message caches because their maximum size limit was reached, the member cannot reconnect in this way.

Group Replication's Group Communication System (GCS) alerts you, by a warning message, when a message that is likely to be needed for recovery by a member that is currently unreachable is removed from the message cache. This warning message is logged on all the active group members (only once for each unreachable member). Although the group members cannot know for sure what message was the last message seen by the unreachable member, the warning message indicates that the cache size might not be sufficient to support your chosen waiting period before a member is expelled.

In this situation, consider increasing the `group_replication_message_cache_size` limit with reference to the expected volume of messages in the time period specified by the `group_replication_member_expel_timeout` system variable plus the 5-second detection period, so that the cache contains all the missed messages required for members to return successfully. You can also consider increasing the cache size limit temporarily if you expect a member to become unreachable for an unusual period of time.

18.7.6.2 Reducing the cache size

The minimum setting for the XCom message cache size is 1 GB up to MySQL 8.0.20. From MySQL 8.0.21, the minimum setting is 134217728 bytes (128 MB), which enables deployment on a host that has a restricted amount of available memory. Having a very low `group_replication_message_cache_size` setting is not recommended if the host is on an unstable network, because a smaller message cache makes it harder for group members to reconnect after a transient loss of connectivity.

If a reconnecting member cannot retrieve all the messages it needs from the XCom message cache, the member must leave the group and rejoin it, in order to retrieve the missing transactions from another member's binary log using distributed recovery. From MySQL 8.0.21, a member that has left a group makes three auto-rejoin attempts by default, so the process of rejoining the group can still take place without operator intervention. However, rejoining using distributed recovery is a significantly longer and more complex process than retrieving messages from an XCom message cache, so the member takes longer to become available and the performance of the group can be impacted. On a stable network, which minimizes the frequency and duration of transient losses of connectivity for members, the frequency of this occurrence should also be minimized, so the group might be able to tolerate a smaller XCom message cache size without a significant impact on its performance.

If you are considering reducing the cache size limit, you can query the Performance Schema table `memory_summary_global_by_event_name` using the following statement:

```
SELECT * FROM performance_schema.memory_summary_global_by_event_name
```

```
  WHERE EVENT_NAME LIKE 'memory/group_rpl/GCS_XCom::xcom_cache';
```

This returns memory usage statistics for the message cache, including the current number of cached entries and current size of the cache. If you reduce the cache size limit, XCom removes the oldest entries that have been decided and delivered until the current size is below the limit. XCom might temporarily exceed the cache size limit while this removal process is ongoing.

18.7.7 Responses to Failure Detection and Network Partitioning

Group Replication's failure detection mechanism is designed to identify group members that are no longer communicating with the group, and expel them as and when it seems likely that they have failed. Having a failure detection mechanism increases the chance that the group contains a majority of correctly working members, and that requests from clients are therefore processed correctly.

Normally, all group members regularly exchange messages with all other group members. If a group member does not receive any messages from a particular fellow member for 5 seconds, when this detection period ends, it creates a suspicion of the fellow member. When a suspicion times out, the suspected member is assumed to have failed, and is expelled from the group. An expelled member is removed from the membership list seen by the other members, but it does not know that it has been expelled from the group, so it sees itself as online and the other members as unreachable. If the member has not in fact failed (for example, because it was just disconnected due to a temporary network issue) and it is able to resume communication with the other members, it receives a view containing the information that it has been expelled from the group.

The responses of group members, including the failed member itself, to these situations can be configured at a number of points in the process. By default, the following behaviors happen if a member is suspected of having failed:

1. Up to MySQL 8.0.20, when a suspicion is created, it times out immediately. The suspected member is liable for expulsion as soon as the expired suspicion is identified by the group. The member could potentially survive for a further few seconds after the timeout because the check for expired suspicions is carried out periodically. From MySQL 8.0.21, a waiting period of 5 seconds is added before the suspicion times out and the suspected member is liable for expulsion.

2. If an expelled member resumes communication and realises that it was expelled, up to MySQL 8.0.20, it does not try to rejoin the group. From MySQL 8.0.21, it makes three automatic attempts to rejoin the group (with 5 minutes between each attempt), and if this auto-rejoin procedure does not work, it then stops trying to rejoin the group.

3. When an expelled member is not trying to rejoin the group, it switches to super read only mode and awaits operator attention. (The exception is in releases from MySQL 8.0.12 to 8.0.15, where the default was for the member to shut itself down. From MySQL 8.0.16, the behavior was changed to match the behavior in MySQL 5.7.)

You can use the Group Replication configuration options described in this section to change these behaviors either permanently or temporarily, to suit your system's requirements and your priorities. If you are experiencing unnecessary expulsions caused by slower networks or machines, networks with a high rate of unexpected transient outages, or planned network outages, consider increasing the expel timeout and auto-rejoin attempts. From MySQL 8.0.21, the default settings have been changed in this direction to reduce the frequency of the need for operator intervention to reinstate expelled members in these situations. Note that while a member is undergoing any of the default behaviors described above, although it does not accept writes, reads can still be made if the member is still communicating with clients, with an increasing likelihood of stale reads over time. If avoiding stale reads is a higher priority for you than avoiding operator intervention, consider reducing the expel timeout and auto-rejoin attempts or setting them to zero.

Members that have not failed might lose contact with part, but not all, of the replication group due to a network partition. For example, in a group of 5 servers (S1,S2,S3,S4,S5), if there is a disconnection between (S1,S2) and (S3,S4,S5) there is a network partition. The first group (S1,S2) is now in a minority because it cannot contact more than half of the group. Any transactions that are

processed by the members in the minority group are blocked, because the majority of the group is unreachable, therefore the group cannot achieve quorum. For a detailed description of this scenario, see Section 18.7.8, “Handling a Network Partition and Loss of Quorum”. In this situation, the default behavior is for the members in both the minority and the majority to remain in the group, continue to accept transactions (although they are blocked on the members in the minority), and wait for operator intervention. This behavior is also configurable.

Note that where group members are at an older MySQL Server release that does not support a relevant setting, or at a release with a different default, they act towards themselves and other group members according to the default behaviors stated above. For example, a member that does not support the `group_replication_member_expel_timeout` system variable expels other members as soon as an expired suspicion is detected, and this expulsion is accepted by other members even if they support the system variable and have a longer timeout set.

18.7.7.1 Expel Timeout

You can use the `group_replication_member_expel_timeout` system variable, which is available from MySQL 8.0.13, to allow additional time between the creation of a suspicion and the expulsion of the suspect member. A suspicion is created when one server does not receive messages from another server, as explained in Section 18.1.4.2, “Failure Detection”.

There is an initial 5-second detection period before a Group Replication group member creates a suspicion of another member (or of itself). A group member is then expelled when another member's suspicion of it (or its own suspicion of itself) times out. A further short period of time might elapse after that before the expelling mechanism detects and implements the expulsion. `group_replication_member_expel_timeout` specifies the period of time in seconds, called the expel timeout, that a group member waits between creating a suspicion, and expelling the suspected member. Suspect members are listed as `UNREACHABLE` during this waiting period, but are not removed from the group's membership list.

- If a suspect member becomes active again before the suspicion times out at the end of the waiting period, the member applies all the messages that were buffered by the remaining group members in XCom's message cache and enters `ONLINE` state, without operator intervention. In this situation, the member is considered by the group as the same incarnation.
- If a suspect member becomes active only after the suspicion times out and is able to resume communications, it receives a view where it is expelled and at that point realises it was expelled. You can use the `group_replication_autorejoin_tries` system variable, which is available from MySQL 8.0.16, to make the member automatically try to rejoin the group at this point. From MySQL 8.0.21, this feature is activated by default and the member makes three auto-rejoin attempts. If the auto-rejoin procedure does not succeed or is not attempted, the expelled member then follows the exit action specified by `group_replication_exit_state_action`.

The waiting period before expelling a member only applies to members that have previously been active in the group. Non-members that were never active in the group do not get this waiting period and are removed after the initial detection period because they took too long to join.

If `group_replication_member_expel_timeout` is set to 0, there is no waiting period, and a suspected member is liable for expulsion immediately after the 5-second detection period ends. This setting is the default up to and including MySQL 8.0.20. This is also the behavior of a group member which is at a MySQL Server version that does not support the `group_replication_member_expel_timeout` system variable. From MySQL 8.0.21, the value defaults to 5, meaning that a suspected member is liable for expulsion 5 seconds after the 5-second detection period. It is not mandatory for all members of a group to have the same setting for `group_replication_member_expel_timeout`, but it is recommended in order to avoid unexpected expulsions. Any member can create a suspicion of any other member, including itself, so the effective expel timeout is that of the member with the lowest setting.

Consider increasing the value of `group_replication_member_expel_timeout` from the default in the following scenarios:

- The network is slow and the default 5 or 10 seconds before expulsion is not long enough for group members to always exchange at least one message.
- The network sometimes has transient outages and you want to avoid unnecessary expulsions and primary member changes at these times.
- The network is not under your direct control and you want to minimize the need for operator intervention.
- A temporary network outage is expected and you do not want some or all of the members to be expelled due to this.
- An individual machine is experiencing a slowdown and you do not want it to be expelled from the group.

You can specify an expel timeout up to a maximum of 3600 seconds (1 hour). It is important to ensure that XCom's message cache is sufficiently large to contain the expected volume of messages in your specified time period, plus the initial 5-second detection period, otherwise members cannot reconnect. You can adjust the cache size limit using the `group_replication_message_cache_size` system variable. For more information, see Section 18.7.6, “XCom Cache Management”.

If any members in a group are currently under suspicion, the group membership cannot be reconfigured (by adding or removing members or electing a new leader). If group membership changes need to be implemented while one or more members are under suspicion, and you want the suspect members to remain in the group, take any actions required to make the members active again, if that is possible. If you cannot make the members active again and you want them to be expelled from the group, you can force the suspicions to time out immediately. Do this by changing the value of `group_replication_member_expel_timeout` on any active members to a value lower than the time that has already elapsed since the suspicions were created. The suspect members then become liable for expulsion immediately.

If a replication group member stops unexpectedly and is immediately restarted (for example, because it was started with `mysqld_safe`), it automatically attempts to rejoin the group if `group_replication_start_on_boot=on` is set. In this situation, it is possible for the restart and rejoin attempt to take place before the member's previous incarnation has been expelled from the group, in which case the member cannot rejoin. From MySQL 8.0.19, Group Replication automatically uses a Group Communication System (GCS) feature to retry the rejoin attempt for the member 10 times, with a 5-second interval between each retry. This should cover most cases and allow enough time for the previous incarnation to be expelled from the group, letting the member rejoin. Note that if the `group_replication_member_expel_timeout` system variable is set to specify a longer waiting period before the member is expelled, the automatic rejoin attempts might still not succeed.

For alternative mitigation strategies to avoid unnecessary expulsions where the `group_replication_member_expel_timeout` system variable is not available, see Section 18.3.2, “Group Replication Limitations”.

18.7.7.2 Unreachable Majority Timeout

By default, members that find themselves in a minority due to a network partition do not automatically leave the group. You can use the system variable `group_replication_unreachable_majority_timeout` to set a number of seconds for a member to wait after losing contact with the majority of group members, and then exit the group. Setting a timeout means you do not need to pro-actively monitor for servers that are in a minority group after a network partition, and you can avoid the possibility of creating a split-brain situation (with two versions of the group membership) due to inappropriate intervention.

When the timeout specified by `group_replication_unreachable_majority_timeout` elapses, all pending transactions that have been processed by the member and the others in the minority group are rolled back, and the servers in that group move to the `ERROR` state. You can use the `group_replication_autorejoin_tries` system variable, which is available from MySQL 8.0.16,

to make the member automatically try to rejoin the group at this point. From MySQL 8.0.21, this feature is activated by default and the member makes three auto-rejoin attempts. If the auto-rejoin procedure does not succeed or is not attempted, the minority member then follows the exit action specified by `group_replication_exit_state_action`.

Consider the following points when deciding whether or not to set an unreachable majority timeout:

- In a symmetric group, for example a group with two or four servers, if both partitions contain an equal number of servers, both groups consider themselves to be in a minority and enter the `ERROR` state. In this situation, the group has no functional partition.
- While a minority group exists, any transactions processed by the minority group are accepted, but blocked because the minority servers cannot reach quorum, until either `STOP GROUP_REPLICATION` is issued on those servers or the unreachable majority timeout is reached.
- If you do not set an unreachable majority timeout, the servers in the minority group never enter the `ERROR` state automatically, and you must stop them manually.
- Setting an unreachable majority timeout has no effect if it is set on the servers in the minority group after the loss of majority has been detected.

If you do not use the `group_replication_unreachable_majority_timeout`system variable, the process for operator invention in the event of a network partition is described in Section 18.7.8, “Handling a Network Partition and Loss of Quorum”. The process involves checking which servers are functioning and forcing a new group membership if necessary.

18.7.7.3 Auto-Rejoin

The `group_replication_autorejoin_tries` system variable, which is available from MySQL 8.0.16, makes a member that has been expelled or reached its unreachable majority timeout try to rejoin the group automatically. Up to MySQL 8.0.20, the value of the system variable defaults to 0, so auto-rejoin is not activated by default. From MySQL 8.0.21, the value of the system variable defaults to 3, meaning that the member automatically makes 3 attempts to rejoin the group, with 5 minutes between each.

When auto-rejoin is not activated, a member accepts its expulsion as soon as it resumes communication, and proceeds to the action specified by the `group_replication_exit_state_action` system variable. After this, manual intervention is needed to bring the member back into the group. Using the auto-rejoin feature is appropriate if you can tolerate the possibility of stale reads and want to minimize the need for manual intervention, especially where transient network issues fairly often result in the expulsion of members.

With auto-rejoin, when the member's expulsion or unreachable majority timeout is reached, it makes an attempt to rejoin (using the current plugin option values), then continues to make further auto-rejoin attempts up to the specified number of tries. After an unsuccessful auto-rejoin attempt, the member waits 5 minutes before the next try. The auto-rejoin attempts and the time between them are called the auto-rejoin procedure. If the specified number of tries is exhausted without the member rejoining or being stopped, the member proceeds to the action specified by the `group_replication_exit_state_action` system variable.

During and between auto-rejoin attempts, a member remains in super read only mode and displays an `ERROR` state on its view of the replication group. During this time, the member does not accept writes. However, reads can still be made on the member, with an increasing likelihood of stale reads over time. If you do want to intervene to take the member offline during the auto-rejoin procedure, the member can be stopped manually at any time by using a `STOP GROUP_REPLICATION` statement or shutting down the server. If you cannot tolerate the possibility of stale reads for any period of time, set the `group_replication_autorejoin_tries` system variable to 0.

You can monitor the auto-rejoin procedure using the Performance Schema. While an auto-rejoin procedure is taking place, the Performance Schema table `events_stages_current`

shows the event “Undergoing auto-rejoin procedure”, with the number of retries that have been attempted so far during this instance of the procedure (in the `WORK_COMPLETED` field). The `events_stages_summary_global_by_event_name` table shows the number of times the server instance has initiated the auto-rejoin procedure (in the `COUNT_STAR` field). The `events_stages_history_long` table shows the time each of these auto-rejoin procedures was completed (in the `TIMER_END` field). While a member is rejoining a replication group, its status can be displayed as `OFFLINE` or `ERROR` before the group completes the compatibility checks and accepts it as a member. When the member is catching up with the group's transactions, its status is `RECOVERING`.

18.7.7.4 Exit Action

The `group_replication_exit_state_action` system variable, which is available from MySQL 8.0.12 and MySQL 5.7.24, specifies what Group Replication does when the member leaves the group unintentionally due to an error or problem, and either fails to auto-rejoin or does not try. Note that in the case of an expelled member, the member does not know that it was expelled until it reconnects to the group, so the specified action is only taken if the member manages to reconnect, or if the member raises a suspicion on itself and expels itself.

In order of impact, the exit actions are as follows:

1. If `READ_ONLY` is the exit action, the instance switches MySQL to super read only mode by setting the system variable `super_read_only` to `ON`. When the member is in super read only mode, clients cannot make any updates, even if they have the `CONNECTION_ADMIN` privilege (or the deprecated `SUPER` privilege). However, clients can still read data, and because updates are no longer being made, there is a probability of stale reads which increases over time. With this setting, you therefore need to pro-actively monitor the servers for failures. This exit action is the default from MySQL 8.0.15. After this exit action is taken, the member's status is displayed as `ERROR` in the view of the group.

2. If `OFFLINE_MODE` is the exit action, the instance switches MySQL to offline mode by setting the system variable `offline_mode` to `ON`. When the member is in offline mode, connected client users are disconnected on their next request and connections are no longer accepted, with the exception of client users that have the `CONNECTION_ADMIN` privilege (or the deprecated `SUPER` privilege). Group Replication also sets the system variable `super_read_only` to `ON`, so clients cannot make any updates, even if they have connected with the `CONNECTION_ADMIN` or `SUPER` privilege. This exit action prevents both updates and stale reads (with the exception of reads by client users with the stated privileges), and enables proxy tools such as MySQL Router to recognize that the server is unavailable and redirect client connections. It also leaves the instance running so that an administrator can attempt to resolve the issue without shutting down MySQL. This exit action is available from MySQL 8.0.18. After this exit action is taken, the member's status is displayed as `ERROR` in the view of the group (not `OFFLINE`, which means a member has Group Replication functionality available but does not currently belong to a group).

3. If `ABORT_SERVER` is the exit action, the instance shuts down MySQL. Instructing the member to shut itself down prevents all stale reads and client updates, but it means that the MySQL Server instance is unavailable and must be restarted, even if the issue could have been resolved without that step. This exit action was the default from MySQL 8.0.12, when the system variable was added, to MySQL 8.0.15 inclusive. After this exit action is taken, the member is removed from the listing of servers in the view of the group.

Bear in mind that operator intervention is required whatever exit action is set, as an ex-member that has exhausted its auto-rejoin attempts (or never had any) and has been expelled from the group is not allowed to rejoin without a restart of Group Replication. The exit action only influences whether or not clients can still read data on the server that was unable to rejoin the group, and whether or not the server stays running.

Important

If a failure occurs before the member has successfully joined the group, the exit action specified by `group_replication_exit_state_action` *is*

> *not taken*. This is the case if there is a failure during the local configuration check, or a mismatch between the configuration of the joining member and the configuration of the group. In these situations, the `super_read_only` system variable is left with its original value, and the server does not shut down MySQL. To ensure that the server cannot accept updates when Group Replication did not start, we therefore recommend that `super_read_only=ON` is set in the server's configuration file at startup, which Group Replication changes to `OFF` on primary members after it has been started successfully. This safeguard is particularly important when the server is configured to start Group Replication on server boot (`group_replication_start_on_boot=ON`), but it is also useful when Group Replication is started manually using a `START GROUP_REPLICATION` command.

If a failure occurs after the member has successfully joined the group, the specified exit action is taken. This is the case in the following situations:

1. *Applier error* - There is an error in the replication applier. This issue is not recoverable.
2. *Distributed recovery not possible* - There is an issue that means Group Replication's distributed recovery process (which uses remote cloning operations and state transfer from the binary log) cannot be completed. Group Replication retries distributed recovery automatically where this makes sense, but stops if there are no more options to complete the process. For details, see Section 18.5.4.4, “Fault Tolerance for Distributed Recovery”.
3. *Group configuration change error* - An error occurred during a group-wide configuration change carried out using a function, as described in Section 18.5.1, “Configuring an Online Group”.
4. *Primary election error* - An error occurred during election of a new primary member for a group in single-primary mode, as described in Section 18.1.3.1, “Single-Primary Mode”.
5. *Unreachable majority timeout* - The member has lost contact with a majority of the group members so is in a minority, and a timeout that was set by the `group_replication_unreachable_majority_timeout` system variable has expired.
6. *Member expelled from group* - A suspicion has been raised on the member, and any timeout set by the `group_replication_member_expel_timeout` system variable has expired, and the member has resumed communication with the group and found that it has been expelled.
7. *Out of auto-rejoin attempts* - The `group_replication_autorejoin_tries` system variable was set to specify a number of auto-rejoin attempts after a loss of majority or expulsion, and the member completed this number of attempts without success.

The following table summarizes the failure scenarios and actions in each case:

Table 18.3 Exit actions in Group Replication failure situations

Failure situation	Group Replication started with `START GROUP_REPLICATION`	Group Replication started with `group_replication_start_on_boot=ON`
Member fails local configuration check Mismatch between joining member and group configuration	`super_read_only` and `offline_mode` unchanged MySQL continues running Set `super_read_only=ON` at startup to prevent updates	`super_read_only` and `offline_mode` unchanged MySQL continues running Set `super_read_only=ON` at startup to prevent updates (Important)
Applier error on member Distributed recovery not possible	`super_read_only` set to `ON` OR	`super_read_only` set to `ON` OR

Failure situation	Group Replication started with `START GROUP_REPLICATION`	Group Replication started with `group_replication_start_on_boot=ON`
Group configuration change error Primary election error Unreachable majority timeout Member expelled from group Out of auto-rejoin attempts	`offline_mode` and `super_read_only` set to `ON` OR MySQL shuts down	`offline_mode` and `super_read_only` set to `ON` OR MySQL shuts down

18.7.8 Handling a Network Partition and Loss of Quorum

The group needs to achieve consensus whenever a change that needs to be replicated happens. This is the case for regular transactions but is also required for group membership changes and some internal messaging that keeps the group consistent. Consensus requires a majority of group members to agree on a given decision. When a majority of group members is lost, the group is unable to progress and blocks because it cannot secure majority or quorum.

Quorum may be lost when there are multiple involuntary failures, causing a majority of servers to be removed abruptly from the group. For example, in a group of 5 servers, if 3 of them become silent at once, the majority is compromised and thus no quorum can be achieved. In fact, the remaining two are not able to tell if the other 3 servers have crashed or whether a network partition has isolated these 2 alone and therefore the group cannot be reconfigured automatically.

On the other hand, if servers exit the group voluntarily, they instruct the group that it should reconfigure itself. In practice, this means that a server that is leaving tells others that it is going away. This means that other members can reconfigure the group properly, the consistency of the membership is maintained and the majority is recalculated. For example, in the above scenario of 5 servers where 3 leave at once, if the 3 leaving servers warn the group that they are leaving, one by one, then the membership is able to adjust itself from 5 to 2, and at the same time, securing quorum while that happens.

Note

Loss of quorum is by itself a side-effect of bad planning. Plan the group size for the number of expected failures (regardless whether they are consecutive, happen all at once or are sporadic).

For a group in single-primary mode, the primary might have transactions that are not yet present on other members at the time of the network partition. If you are considering excluding the primary from the new group, be aware that such transactions might be lost. A member with extra transactions cannot rejoin the group, and the attempt results in an error with the message `This member has more executed transactions than those present in the group`. Set the `group_replication_unreachable_majority_timeout` system variable for the group members to avoid this situation.

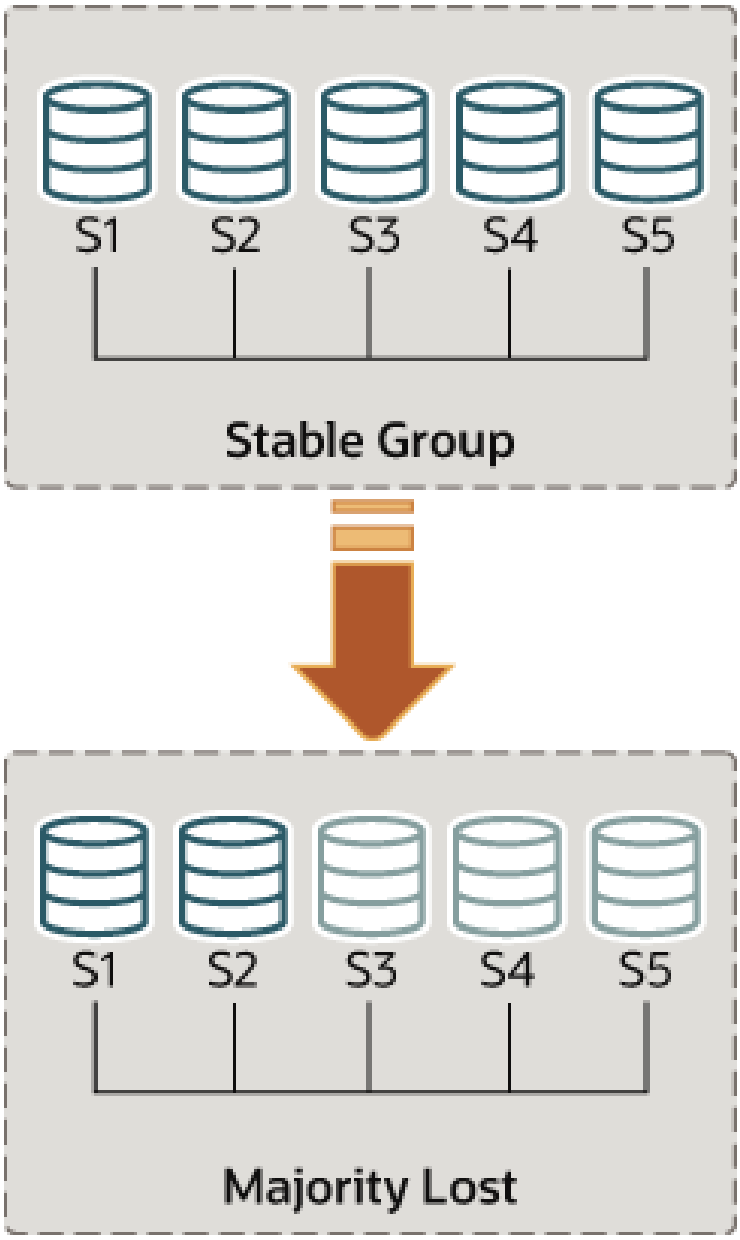
The following sections explain what to do if the system partitions in such a way that no quorum is automatically achieved by the servers in the group.

Detecting Partitions

The `replication_group_members` performance schema table presents the status of each server in the current view from the perspective of this server. The majority of the time the system does not run into partitioning, and therefore the table shows information that is consistent across all servers in the group. In other words, the status of each server on this table is agreed by all in the current

view. However, if there is network partitioning, and quorum is lost, then the table shows the status `UNREACHABLE` for those servers that it cannot contact. This information is exported by the local failure detector built into Group Replication.

Figure 18.14 Losing Quorum



To understand this type of network partition the following section describes a scenario where there are initially 5 servers working together correctly, and the changes that then happen to the group once only 2 servers are online. The scenario is depicted in the figure.

As such, lets assume that there is a group with these 5 servers in it:

- Server s1 with member identifier `199b2df7-4aaf-11e6-bb16-28b2bd168d07`
- Server s2 with member identifier `199bb88e-4aaf-11e6-babe-28b2bd168d07`
- Server s3 with member identifier `1999b9fb-4aaf-11e6-bb54-28b2bd168d07`
- Server s4 with member identifier `19ab72fc-4aaf-11e6-bb51-28b2bd168d07`

- Server s5 with member identifier `19b33846-4aaf-11e6-ba81-28b2bd168d07`

Initially the group is running fine and the servers are happily communicating with each other. You can verify this by logging into s1 and looking at its `replication_group_members` performance schema table. For example:

```
mysql> SELECT MEMBER_ID,MEMBER_STATE, MEMBER_ROLE FROM performance_schema.replication_group_members;
+--------------------------------------+--------------+-------------+
| MEMBER_ID                            | MEMBER_STATE | MEMBER_ROLE |
+--------------------------------------+--------------+-------------+
| 1999b9fb-4aaf-11e6-bb54-28b2bd168d07 | ONLINE       | SECONDARY   |
| 199b2df7-4aaf-11e6-bb16-28b2bd168d07 | ONLINE       | PRIMARY     |
| 199bb88e-4aaf-11e6-babe-28b2bd168d07 | ONLINE       | SECONDARY   |
| 19ab72fc-4aaf-11e6-bb51-28b2bd168d07 | ONLINE       | SECONDARY   |
| 19b33846-4aaf-11e6-ba81-28b2bd168d07 | ONLINE       | SECONDARY   |
+--------------------------------------+--------------+-------------+
```

However, moments later there is a catastrophic failure and servers s3, s4 and s5 stop unexpectedly. A few seconds after this, looking again at the `replication_group_members` table on s1 shows that it is still online, but several others members are not. In fact, as seen below they are marked as `UNREACHABLE`. Moreover, the system could not reconfigure itself to change the membership, because the majority has been lost.

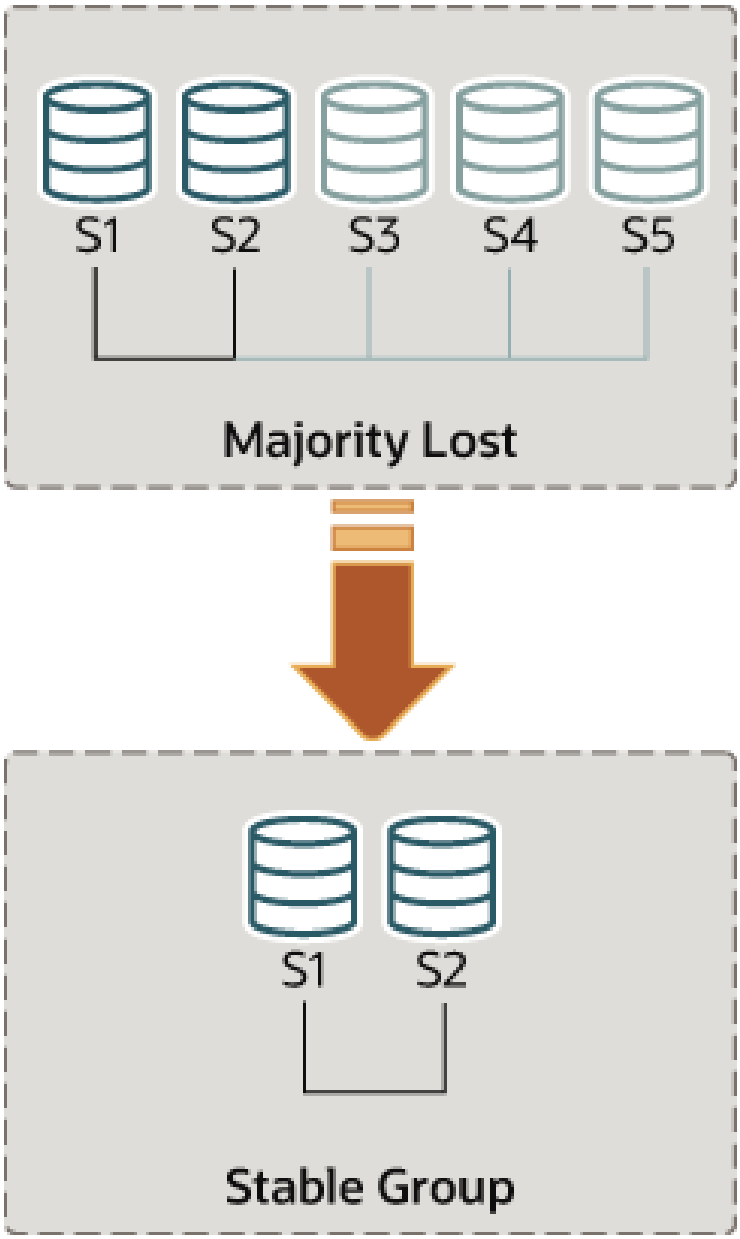
```
mysql> SELECT MEMBER_ID,MEMBER_STATE FROM performance_schema.replication_group_members;
+--------------------------------------+--------------+
| MEMBER_ID                            | MEMBER_STATE |
+--------------------------------------+--------------+
| 1999b9fb-4aaf-11e6-bb54-28b2bd168d07 | UNREACHABLE  |
| 199b2df7-4aaf-11e6-bb16-28b2bd168d07 | ONLINE       |
| 199bb88e-4aaf-11e6-babe-28b2bd168d07 | ONLINE       |
| 19ab72fc-4aaf-11e6-bb51-28b2bd168d07 | UNREACHABLE  |
| 19b33846-4aaf-11e6-ba81-28b2bd168d07 | UNREACHABLE  |
+--------------------------------------+--------------+
```

The table shows that s1 is now in a group that has no means of progressing without external intervention, because a majority of the servers are unreachable. In this particular case, the group membership list needs to be reset to allow the system to proceed, which is explained in this section. Alternatively, you could also choose to stop Group Replication on s1 and s2 (or stop completely s1 and s2), figure out what happened with s3, s4 and s5 and then restart Group Replication (or the servers).

Unblocking a Partition

Group replication enables you to reset the group membership list by forcing a specific configuration. For instance in the case above, where s1 and s2 are the only servers online, you could choose to force a membership configuration consisting of only s1 and s2. This requires checking some information about s1 and s2 and then using the `group_replication_force_members` variable.

Figure 18.15 Forcing a New Membership



Suppose that you are back in the situation where s1 and s2 are the only servers left in the group. Servers s3, s4 and s5 have left the group unexpectedly. To make servers s1 and s2 continue, you want to force a membership configuration that contains only s1 and s2.

Warning

This procedure uses `group_replication_force_members` and should be considered a last resort remedy. It *must* be used with extreme care and only for overriding loss of quorum. If misused, it could create an artificial split-brain scenario or block the entire system altogether.

When forcing a new membership configuration, make sure that any servers are going to be forced out of the group are indeed stopped. In the scenario depicted above, if s3, s4 and s5 are not really unreachable but instead are online, they may have formed their own functional partition (they are 3 out of 5, hence they have the majority). In that case, forcing a group membership list with s1 and s2 could create an artificial split-brain situation. Therefore it is important before forcing a new membership configuration to ensure that the servers to be excluded are indeed shut down and if they are not, shut them down before proceeding.

Warning

For a group in single-primary mode, the primary might have transactions that are not yet present on other members at the time of the network partition. If you are considering excluding the primary from the new group, be aware that such transactions might be lost. A member with extra transactions cannot rejoin the group, and the attempt results in an error with the message `This member has more executed transactions than those present in the group`. Set the `group_replication_unreachable_majority_timeout` system variable for the group members to avoid this situation.

Recall that the system is blocked and the current configuration is the following (as perceived by the local failure detector on s1):

```
mysql> SELECT MEMBER_ID,MEMBER_STATE FROM performance_schema.replication_group_members;
+--------------------------------------+--------------+
| MEMBER_ID                            | MEMBER_STATE |
+--------------------------------------+--------------+
| 1999b9fb-4aaf-11e6-bb54-28b2bd168d07 | UNREACHABLE  |
| 199b2df7-4aaf-11e6-bb16-28b2bd168d07 | ONLINE       |
| 199bb88e-4aaf-11e6-babe-28b2bd168d07 | ONLINE       |
| 19ab72fc-4aaf-11e6-bb51-28b2bd168d07 | UNREACHABLE  |
| 19b33846-4aaf-11e6-ba81-28b2bd168d07 | UNREACHABLE  |
+--------------------------------------+--------------+
```

The first thing to do is to check what is the local address (group communication identifier) for s1 and s2. Log in to s1 and s2 and get that information as follows.

```
mysql> SELECT @@group_replication_local_address;
```

Once you know the group communication addresses of s1 (`127.0.0.1:10000`) and s2 (`127.0.0.1:10001`), you can use that on one of the two servers to inject a new membership configuration, thus overriding the existing one that has lost quorum. To do that on s1:

```
mysql> SET GLOBAL group_replication_force_members="127.0.0.1:10000,127.0.0.1:10001";
```

This unblocks the group by forcing a different configuration. Check `replication_group_members` on both s1 and s2 to verify the group membership after this change. First on s1.

```
mysql> SELECT MEMBER_ID,MEMBER_STATE FROM performance_schema.replication_group_members;
+--------------------------------------+--------------+
| MEMBER_ID                            | MEMBER_STATE |
+--------------------------------------+--------------+
| b5ffe505-4ab6-11e6-b04b-28b2bd168d07 | ONLINE       |
| b60907e7-4ab6-11e6-afb7-28b2bd168d07 | ONLINE       |
+--------------------------------------+--------------+
```

And then on s2.

```
mysql> SELECT * FROM performance_schema.replication_group_members;
+--------------------------------------+--------------+
| MEMBER_ID                            | MEMBER_STATE |
+--------------------------------------+--------------+
| b5ffe505-4ab6-11e6-b04b-28b2bd168d07 | ONLINE       |
| b60907e7-4ab6-11e6-afb7-28b2bd168d07 | ONLINE       |
+--------------------------------------+--------------+
```

After you have used the `group_replication_force_members` system variable to successfully force a new group membership and unblock the group, ensure that you clear the system variable. `group_replication_force_members` must be empty in order to issue a `START GROUP_REPLICATION` statement.

18.7.9 Monitoring Group Replication Memory Usage with Performance Schema Memory Instrumentation

From MySQL 8.0.30, Performance Schema provides instrumentation for performance monitoring of Group Replication memory usage. To view the available Group Replication instrumentation, issue the following query:

```
mysql> SELECT NAME,ENABLED FROM performance_schema.setup_instruments
       WHERE NAME LIKE 'memory/group_rpl/%';
+-----------------------------------------------------------------------+---------+
| NAME                                                                  | ENABLED |
+-----------------------------------------------------------------------+---------+
| memory/group_rpl/write_set_encoded                                    | YES     |
| memory/group_rpl/certification_data                                   | YES     |
| memory/group_rpl/certification_data_gc                                | YES     |
| memory/group_rpl/certification_info                                   | YES     |
| memory/group_rpl/transaction_data                                     | YES     |
| memory/group_rpl/sql_service_command_data                             | YES     |
| memory/group_rpl/mysql_thread_queued_task                             | YES     |
| memory/group_rpl/message_service_queue                                | YES     |
| memory/group_rpl/message_service_received_message                     | YES     |
| memory/group_rpl/group_member_info                                    | YES     |
| memory/group_rpl/consistent_members_that_must_prepare_transaction     | YES     |
| memory/group_rpl/consistent_transactions                              | YES     |
| memory/group_rpl/consistent_transactions_prepared                     | YES     |
| memory/group_rpl/consistent_transactions_waiting                      | YES     |
| memory/group_rpl/consistent_transactions_delayed_view_change          | YES     |
| memory/group_rpl/GCS_XCom::xcom_cache                                 | YES     |
| memory/group_rpl/Gcs_message_data::m_buffer                           | YES     |
+-----------------------------------------------------------------------+---------+
```

For more information on Performance Schema's memory instrumentation and events, see Section 27.12.20.10, “Memory Summary Tables”.

Performance Schema Group Replication instruments memory allocation for Group Replication.

The `memory/group_rpl/` Performance Schema instrumentation was updated in 8.0.30 to extend monitoring of Group Replication memory usage. `memory/group_rpl/` contains the following instruments:

- `write_set_encoded`: Memory allocated to encode the write set before it is broadcast to the group members.
- `Gcs_message_data::m_buffer`: Memory allocated for the transaction data payload sent to the network.
- `certification_data`: Memory allocated for certification of incoming transactions.
- `certification_data_gc`: Memory allocated for the GTID_EXECUTED sent by each member for garbage collection.
- `certification_info`: Memory allocated for storage of certification information allocated to resolve conflicts between concurrent transactions.
- `transaction_data`: Memory allocated for incoming transactions queued for the plugin pipeline.
- `message_service_received_message`: Memory allocated to receiving messages from Group Replication delivery message service.
- `sql_service_command_data`: Memory allocated for processing the queue of internal SQL service commands.
- `mysql_thread_queued_task`: Memory allocated when a MySQL-thread dependent task is added to the processing queue.
- `message_service_queue`: Memory allocated for queued messages of the Group Replication delivery message service.
- `GCS_XCom::xcom_cache`: Memory allocated to XCOM ache for messaging and metadata exchanged between group members as part of the consensus protocol.

- `consistent_members_that_must_prepare_transaction`: Memory allocated to hold list of members preparing transaction for Group Replication transaction consistency guarantees.
- `consistent_transactions`: Memory allocated to hold transaction and list of members that must prepare that transaction for Group Replication transaction consistency guarantees.
- `consistent_transactions_prepared`: Memory allocated to hold list of transaction's info prepared for the Group Replication Transaction Consistency Guarantees.
- `consistent_transactions_waiting`: Memory allocated to hold information on a list of transactions while preceding prepared transactions with consistency of `AFTER` and `BEFORE_AND_AFTER` are processed.
- `consistent_transactions_delayed_view_change`: Memory allocated to hold list of view change events (`view_change_log_event`) delayed by prepared consistent transactions waiting for prepare acknowledgement.
- `group_member_info`: Memory allocated to hold the group member properties. Properties such as hostname, port, member weight and role, and so on.

The following instruments in the `memory/sql/` grouping are also used to monitor Group Replication memory:

- `Log_event`: Memory allocated to the write set generation process.
- `write_set_extraction`: Memory allocated to the transaction's generated write set before it is committed.
- `Gtid_set::to_string`: Memory allocated to stored the string representation of a GTID set.
- `Gtid_set::Interval_chunk`: Memory allocated to store the GTID object.

18.7.9.1 Enabling or Disabling Group Replication Instrumentation

To enable all the Group Replication instrumentation from the command line, run the following in the SQL client of your choice:

```
UPDATE performance_schema.setup_instruments SET ENABLED = 'YES'
WHERE NAME LIKE 'memory/group_rpl/%';
```

To disable all the Group Replication instrumentation from the command line, run the following in the SQL client of your choice:

```
UPDATE performance_schema.setup_instruments SET ENABLED = 'NO'
WHERE NAME LIKE 'memory/group_rpl/%';
```

To enable all the Group Replication instrumentation at server startup, add the following to your option file:

```
[mysqld]
performance-schema-instrument='memory/group_rpl/%=ON'
```

To disable all the Group Replication instrumentation at server startup, add the following to your option file:

```
[mysqld]
performance-schema-instrument='memory/group_rpl/%=OFF'
```

To enable or disable individual instruments in that group, replace the wildcard (%) with the full name of the instrument.

For more information, see Section 27.3, “Performance Schema Startup Configuration” and Section 27.4, “Performance Schema Runtime Configuration”.

18.7.9.2 Example Queries

This section describes sample queries using the instruments and events for monitoring Group Replication memory usage. The queries retrieve data from the `memory_summary_global_by_event_name` table.

The memory data can be queried for individual events, for example:

```
SELECT * FROM performance_schema.memory_summary_global_by_event_name
WHERE EVENT_NAME = 'memory/group_rpl/write_set_encoded'\G

*************************** 1. row ***************************
                 EVENT_NAME: memory/group_rpl/write_set_encoded
                COUNT_ALLOC: 1
                 COUNT_FREE: 0
   SUM_NUMBER_OF_BYTES_ALLOC: 45
    SUM_NUMBER_OF_BYTES_FREE: 0
              LOW_COUNT_USED: 0
          CURRENT_COUNT_USED: 1
             HIGH_COUNT_USED: 1
    LOW_NUMBER_OF_BYTES_USED: 0
CURRENT_NUMBER_OF_BYTES_USED: 45
   HIGH_NUMBER_OF_BYTES_USED: 45
```

See Section 27.12.20.10, “Memory Summary Tables” for more information on the columns.

You can also define queries which sum various events to provide overviews of specific areas of memory usage.

The following examples are described:

- Memory Used to Capture Transactions
- Memory Used to Broadcast Transactions
- Total Memory Used in Group Replication
- Memory Used in Certification
- Memory Used in Certification
- Memory Used in Replication Pipeline
- Memory Used in Consistency
- Memory Used in Delivery Message Service
- Memory Used to Broadcast and Receive Transactions

Memory Used to Capture Transactions

The memory allocated to capture user transactions is a sum of the `write_set_encoded`, `write_set_extraction`, and `Log_event` event's values. For example:

```
          mysql> SELECT * FROM (
                   SELECT
                     (CASE
                        WHEN EVENT_NAME LIKE 'memory/group_rpl/write_set_encoded'
                        THEN 'memory/group_rpl/memory_gr'
                        WHEN EVENT_NAME = 'memory/sql/write_set_extraction'
                        THEN 'memory/group_rpl/memory_gr'
                        WHEN EVENT_NAME = 'memory/sql/Log_event'
```

```
                    THEN 'memory/group_rpl/memory_gr'
                    ELSE 'memory_gr_rest'
                  END) AS EVENT_NAME, SUM(COUNT_ALLOC), SUM(COUNT_FREE),
                SUM(SUM_NUMBER_OF_BYTES_ALLOC),
                SUM(SUM_NUMBER_OF_BYTES_FREE), SUM(LOW_COUNT_USED),
                SUM(CURRENT_COUNT_USED), SUM(HIGH_COUNT_USED),
                SUM(LOW_NUMBER_OF_BYTES_USED), SUM(CURRENT_NUMBER_OF_BYTES_USED),
                SUM(HIGH_NUMBER_OF_BYTES_USED)
              FROM performance_schema.memory_summary_global_by_event_name
              GROUP BY (CASE
                          WHEN EVENT_NAME LIKE 'memory/group_rpl/write_set_encoded'
                          THEN 'memory/group_rpl/memory_gr'
                          WHEN EVENT_NAME = 'memory/sql/write_set_extraction'
                          THEN 'memory/group_rpl/memory_gr'
                          WHEN EVENT_NAME = 'memory/sql/Log_event'
                          THEN 'memory/group_rpl/memory_gr'
                          ELSE 'memory_gr_rest'
                        END)
      ) f
      WHERE f.EVENT_NAME != 'memory_gr_rest'
      *************************** 1. row ***************************
                           EVENT_NAME: memory/group_rpl/memory_gr
                    SUM(COUNT_ALLOC): 127
                     SUM(COUNT_FREE): 117
        SUM(SUM_NUMBER_OF_BYTES_ALLOC): 54808
         SUM(SUM_NUMBER_OF_BYTES_FREE): 52051
                   SUM(LOW_COUNT_USED): 0
                SUM(CURRENT_COUNT_USED): 10
                   SUM(HIGH_COUNT_USED): 35
         SUM(LOW_NUMBER_OF_BYTES_USED): 0
      SUM(CURRENT_NUMBER_OF_BYTES_USED): 2757
        SUM(HIGH_NUMBER_OF_BYTES_USED): 15630
```

Memory Used to Broadcast Transactions

The memory allocated to broadcast transactions is a sum of the `Gcs_message_data::m_buffer`, `transaction_data`, and `GCS_XCom::xcom_cache` event values. For example:

```
mysql> SELECT * FROM (
         SELECT
           (CASE
              WHEN EVENT_NAME =  'memory/group_rpl/Gcs_message_data::m_buffer'
              THEN 'memory/group_rpl/memory_gr'
              WHEN EVENT_NAME = 'memory/group_rpl/GCS_XCom::xcom_cache'
              THEN 'memory/group_rpl/memory_gr'
              WHEN EVENT_NAME = 'memory/group_rpl/transaction_data'
              THEN 'memory/group_rpl/memory_gr'
              ELSE 'memory_gr_rest'
            END) AS EVENT_NAME, SUM(COUNT_ALLOC), SUM(COUNT_FREE),
            SUM(SUM_NUMBER_OF_BYTES_ALLOC),
            SUM(SUM_NUMBER_OF_BYTES_FREE), SUM(LOW_COUNT_USED),
            SUM(CURRENT_COUNT_USED), SUM(HIGH_COUNT_USED),
            SUM(LOW_NUMBER_OF_BYTES_USED), SUM(CURRENT_NUMBER_OF_BYTES_USED),
            SUM(HIGH_NUMBER_OF_BYTES_USED)
          FROM performance_schema.memory_summary_global_by_event_name
          GROUP BY (CASE
                      WHEN EVENT_NAME =  'memory/group_rpl/Gcs_message_data::m_buffer'
                      THEN 'memory/group_rpl/memory_gr'
                      WHEN EVENT_NAME = 'memory/group_rpl/GCS_XCom::xcom_cache'
                      THEN 'memory/group_rpl/memory_gr'
                      WHEN EVENT_NAME = 'memory/group_rpl/transaction_data'
                      THEN 'memory/group_rpl/memory_gr'
                      ELSE 'memory_gr_rest'
                    END)
   ) f
   WHERE f.EVENT_NAME != 'memory_gr_rest'\G
   *************************** 1. row ***************************
                        EVENT_NAME: memory/group_rpl/memory_gr
                 SUM(COUNT_ALLOC): 84
                  SUM(COUNT_FREE): 31
     SUM(SUM_NUMBER_OF_BYTES_ALLOC): 1072324
```

```
          SUM(SUM_NUMBER_OF_BYTES_FREE): 7149
                    SUM(LOW_COUNT_USED): 0
                 SUM(CURRENT_COUNT_USED): 53
                    SUM(HIGH_COUNT_USED): 59
          SUM(LOW_NUMBER_OF_BYTES_USED): 0
      SUM(CURRENT_NUMBER_OF_BYTES_USED): 1065175
         SUM(HIGH_NUMBER_OF_BYTES_USED): 1065809
```

Total Memory Used in Group Replication

The memory allocation to sending and receiving transactions, certification, and all other major processes. It is calculated by querying all the events of the `memory/group_rpl/` group. For example:

```
         mysql> SELECT * FROM (
                 SELECT
                   (CASE
                      WHEN EVENT_NAME LIKE 'memory/group_rpl/%'
                      THEN 'memory/group_rpl/memory_gr'
                      ELSE 'memory_gr_rest'
                    END) AS EVENT_NAME, SUM(COUNT_ALLOC), SUM(COUNT_FREE),
                    SUM(SUM_NUMBER_OF_BYTES_ALLOC),
                    SUM(SUM_NUMBER_OF_BYTES_FREE), SUM(LOW_COUNT_USED),
                    SUM(CURRENT_COUNT_USED), SUM(HIGH_COUNT_USED),
                    SUM(LOW_NUMBER_OF_BYTES_USED), SUM(CURRENT_NUMBER_OF_BYTES_USED),
                    SUM(HIGH_NUMBER_OF_BYTES_USED)
                 FROM performance_schema.memory_summary_global_by_event_name
                 GROUP BY (CASE
                             WHEN EVENT_NAME LIKE 'memory/group_rpl/%'
                             THEN 'memory/group_rpl/memory_gr'
                             ELSE 'memory_gr_rest'
                           END)
       ) f
       WHERE f.EVENT_NAME != 'memory_gr_rest'\G
       *************************** 1. row ***************************
                           EVENT_NAME: memory/group_rpl/memory_gr
                         SUM(COUNT_ALLOC): 190
                          SUM(COUNT_FREE): 127
          SUM(SUM_NUMBER_OF_BYTES_ALLOC): 1096370
           SUM(SUM_NUMBER_OF_BYTES_FREE): 28675
                     SUM(LOW_COUNT_USED): 0
                  SUM(CURRENT_COUNT_USED): 63
                     SUM(HIGH_COUNT_USED): 77
           SUM(LOW_NUMBER_OF_BYTES_USED): 0
       SUM(CURRENT_NUMBER_OF_BYTES_USED): 1067695
          SUM(HIGH_NUMBER_OF_BYTES_USED): 1069255
```

Memory Used in Certification

The memory allocation in the certification process is a sum of the `certification_data`, `certification_data_gc`, and `certification_info` event values. For example:

```
        mysql> SELECT * FROM (
                 SELECT
                   (CASE
                      WHEN EVENT_NAME = 'memory/group_rpl/certification_data'
                      THEN 'memory/group_rpl/certification'
                      WHEN EVENT_NAME = 'memory/group_rpl/certification_data_gc'
                      THEN 'memory/group_rpl/certification'
                      WHEN EVENT_NAME = 'memory/group_rpl/certification_info'
                      THEN 'memory/group_rpl/certification'
                      ELSE 'memory_gr_rest'
                    END) AS EVENT_NAME, SUM(COUNT_ALLOC), SUM(COUNT_FREE),
                    SUM(SUM_NUMBER_OF_BYTES_ALLOC),
                    SUM(SUM_NUMBER_OF_BYTES_FREE), SUM(LOW_COUNT_USED),
                    SUM(CURRENT_COUNT_USED), SUM(HIGH_COUNT_USED),
                    SUM(LOW_NUMBER_OF_BYTES_USED), SUM(CURRENT_NUMBER_OF_BYTES_USED),
                    SUM(HIGH_NUMBER_OF_BYTES_USED)
                 FROM performance_schema.memory_summary_global_by_event_name
                 GROUP BY (CASE
```

```
                          WHEN EVENT_NAME = 'memory/group_rpl/certification_data'
                          THEN 'memory/group_rpl/certification'
                          WHEN EVENT_NAME = 'memory/group_rpl/certification_data_gc'
                          THEN 'memory/group_rpl/certification'
                          WHEN EVENT_NAME = 'memory/group_rpl/certification_info'
                          THEN 'memory/group_rpl/certification'
                          ELSE 'memory_gr_rest'
                        END)
      ) f
      WHERE f.EVENT_NAME != 'memory_gr_rest'\G
      *************************** 1. row ***************************
                          EVENT_NAME: memory/group_rpl/certification
                      SUM(COUNT_ALLOC): 80
                       SUM(COUNT_FREE): 80
         SUM(SUM_NUMBER_OF_BYTES_ALLOC): 9442
          SUM(SUM_NUMBER_OF_BYTES_FREE): 9442
                    SUM(LOW_COUNT_USED): 0
                 SUM(CURRENT_COUNT_USED): 0
                    SUM(HIGH_COUNT_USED): 66
          SUM(LOW_NUMBER_OF_BYTES_USED): 0
      SUM(CURRENT_NUMBER_OF_BYTES_USED): 0
         SUM(HIGH_NUMBER_OF_BYTES_USED): 6561
```

Memory Used in Replication Pipeline

The memory allocation of the replication pipeline is the sum of the `certification_data` and `transaction_data` event values. For example:

```
  mysql> SELECT * FROM (
           SELECT
             (CASE
                WHEN EVENT_NAME LIKE 'memory/group_rpl/certification_data'
                THEN 'memory/group_rpl/pipeline'
                WHEN EVENT_NAME LIKE 'memory/group_rpl/transaction_data'
                THEN 'memory/group_rpl/pipeline'
                ELSE 'memory_gr_rest'
              END) AS EVENT_NAME, SUM(COUNT_ALLOC), SUM(COUNT_FREE),
              SUM(SUM_NUMBER_OF_BYTES_ALLOC),
              SUM(SUM_NUMBER_OF_BYTES_FREE), SUM(LOW_COUNT_USED),
              SUM(CURRENT_COUNT_USED), SUM(HIGH_COUNT_USED),
              SUM(LOW_NUMBER_OF_BYTES_USED), SUM(CURRENT_NUMBER_OF_BYTES_USED),
              SUM(HIGH_NUMBER_OF_BYTES_USED)
            FROM performance_schema.memory_summary_global_by_event_name
            GROUP BY (CASE
                        WHEN EVENT_NAME LIKE 'memory/group_rpl/certification_data'
                        THEN 'memory/group_rpl/pipeline'
                        WHEN EVENT_NAME LIKE 'memory/group_rpl/transaction_data'
                        THEN 'memory/group_rpl/pipeline'
                        ELSE 'memory_gr_rest'
                      END)
      ) f
      WHERE f.EVENT_NAME != 'memory_gr_rest'\G
      *************************** 1. row ***************************
                        EVENT_NAME: memory/group_rpl/pipeline
                       COUNT_ALLOC: 17
                        COUNT_FREE: 13
         SUM_NUMBER_OF_BYTES_ALLOC: 2483
          SUM_NUMBER_OF_BYTES_FREE: 1668
                    LOW_COUNT_USED: 0
                 CURRENT_COUNT_USED: 4
                    HIGH_COUNT_USED: 4
          LOW_NUMBER_OF_BYTES_USED: 0
      CURRENT_NUMBER_OF_BYTES_USED: 815
         HIGH_NUMBER_OF_BYTES_USED: 815
```

Memory Used in Consistency

The memory allocation for transaction consistency guarantees is the sum of the `consistent_members_that_must_prepare_transaction`, `consistent_transactions`,

`consistent_transactions_prepared`, `consistent_transactions_waiting`, and `consistent_transactions_delayed_view_change` event values. For example:

```
mysql> SELECT * FROM (
         SELECT
           (CASE
              WHEN EVENT_NAME = 'memory/group_rpl/consistent_members_that_must_prepare_transac
              THEN 'memory/group_rpl/consistency'
              WHEN EVENT_NAME = 'memory/group_rpl/consistent_transactions'
              THEN 'memory/group_rpl/consistency'
              WHEN EVENT_NAME = 'memory/group_rpl/consistent_transactions_prepared'
              THEN 'memory/group_rpl/consistency'
              WHEN EVENT_NAME = 'memory/group_rpl/consistent_transactions_waiting'
              THEN 'memory/group_rpl/consistency'
              WHEN EVENT_NAME = 'memory/group_rpl/consistent_transactions_delayed_view_change'
              THEN 'memory/group_rpl/consistency'
              ELSE 'memory_gr_rest'
            END) AS EVENT_NAME, SUM(COUNT_ALLOC), SUM(COUNT_FREE),
           SUM(SUM_NUMBER_OF_BYTES_ALLOC),
           SUM(SUM_NUMBER_OF_BYTES_FREE), SUM(LOW_COUNT_USED),
           SUM(CURRENT_COUNT_USED), SUM(HIGH_COUNT_USED),
           SUM(LOW_NUMBER_OF_BYTES_USED), SUM(CURRENT_NUMBER_OF_BYTES_USED),
           SUM(HIGH_NUMBER_OF_BYTES_USED)
         FROM performance_schema.memory_summary_global_by_event_name
         GROUP BY (CASE
                     WHEN EVENT_NAME = 'memory/group_rpl/consistent_members_that_must_prepare_
                     THEN 'memory/group_rpl/consistency'
                     WHEN EVENT_NAME = 'memory/group_rpl/consistent_transactions'
                     THEN 'memory/group_rpl/consistency'
                     WHEN EVENT_NAME = 'memory/group_rpl/consistent_transactions_prepared'
                     THEN 'memory/group_rpl/consistency'
                     WHEN EVENT_NAME = 'memory/group_rpl/consistent_transactions_waiting'
                     THEN 'memory/group_rpl/consistency'
                     WHEN EVENT_NAME = 'memory/group_rpl/consistent_transactions_delayed_view_
                     THEN 'memory/group_rpl/consistency'
                     ELSE 'memory_gr_rest'
                   END)
) f
WHERE f.EVENT_NAME != 'memory_gr_rest'\G
*************************** 1. row ***************************
                 EVENT_NAME: memory/group_rpl/consistency
                COUNT_ALLOC: 16
                 COUNT_FREE: 6
   SUM_NUMBER_OF_BYTES_ALLOC: 1464
    SUM_NUMBER_OF_BYTES_FREE: 528
              LOW_COUNT_USED: 0
          CURRENT_COUNT_USED: 10
             HIGH_COUNT_USED: 11
    LOW_NUMBER_OF_BYTES_USED: 0
CURRENT_NUMBER_OF_BYTES_USED: 936
   HIGH_NUMBER_OF_BYTES_USED: 1024
```

Memory Used in Delivery Message Service

Note

This instrumentation applies only to data received, not data sent.

The memory allocation for the Group Replication delivery message service is the sum of the `message_service_received_message` and `message_service_queue` event values. For example:

```
mysql> SELECT * FROM (
         SELECT
           (CASE
              WHEN EVENT_NAME = 'memory/group_rpl/message_service_received_message'
              THEN 'memory/group_rpl/message_service'
              WHEN EVENT_NAME = 'memory/group_rpl/message_service_queue'
              THEN 'memory/group_rpl/message_service'
```

```
                      ELSE 'memory_gr_rest'
                    END) AS EVENT_NAME, SUM(COUNT_ALLOC), SUM(COUNT_FREE),
                   SUM(SUM_NUMBER_OF_BYTES_ALLOC),
                   SUM(SUM_NUMBER_OF_BYTES_FREE), SUM(LOW_COUNT_USED),
                   SUM(CURRENT_COUNT_USED), SUM(HIGH_COUNT_USED),
                   SUM(LOW_NUMBER_OF_BYTES_USED), SUM(CURRENT_NUMBER_OF_BYTES_USED),
                   SUM(HIGH_NUMBER_OF_BYTES_USED)
                 FROM performance_schema.memory_summary_global_by_event_name
                 GROUP BY (CASE
                             WHEN EVENT_NAME = 'memory/group_rpl/message_service_received_message'
                             THEN 'memory/group_rpl/message_service'
                             WHEN EVENT_NAME = 'memory/group_rpl/message_service_queue'
                             THEN 'memory/group_rpl/message_service'
                             ELSE 'memory_gr_rest'
                           END)
       ) f
       WHERE f.EVENT_NAME != 'memory_gr_rest'\G
       *************************** 1. row ***************************
                       EVENT_NAME: memory/group_rpl/message_service
                      COUNT_ALLOC: 2
                       COUNT_FREE: 0
          SUM_NUMBER_OF_BYTES_ALLOC: 1048664
           SUM_NUMBER_OF_BYTES_FREE: 0
                     LOW_COUNT_USED: 0
                  CURRENT_COUNT_USED: 2
                     HIGH_COUNT_USED: 2
           LOW_NUMBER_OF_BYTES_USED: 0
       CURRENT_NUMBER_OF_BYTES_USED: 1048664
          HIGH_NUMBER_OF_BYTES_USED: 1048664
```

Memory Used to Broadcast and Receive Transactions

The memory allocation for the broadcasting and receiving transactions to and from the network is the sum of the `wGcs_message_data::m_buffer` and `GCS_XCom::xcom_cache` event values. For example:

```
mysql> SELECT * FROM (
         SELECT
           (CASE
              WHEN EVENT_NAME = 'memory/group_rpl/Gcs_message_data::m_buffer'
              THEN 'memory/group_rpl/memory_gr'
              WHEN EVENT_NAME = 'memory/group_rpl/GCS_XCom::xcom_cache'
              THEN 'memory/group_rpl/memory_gr'
              ELSE 'memory_gr_rest'
            END) AS EVENT_NAME, SUM(COUNT_ALLOC), SUM(COUNT_FREE),
           SUM(SUM_NUMBER_OF_BYTES_ALLOC),
           SUM(SUM_NUMBER_OF_BYTES_FREE), SUM(LOW_COUNT_USED),
           SUM(CURRENT_COUNT_USED), SUM(HIGH_COUNT_USED),
           SUM(LOW_NUMBER_OF_BYTES_USED), SUM(CURRENT_NUMBER_OF_BYTES_USED),
           SUM(HIGH_NUMBER_OF_BYTES_USED)
         FROM performance_schema.memory_summary_global_by_event_name
         GROUP BY (CASE
                     WHEN EVENT_NAME = 'memory/group_rpl/Gcs_message_data::m_buffer'
                     THEN 'memory/group_rpl/memory_gr'
                     WHEN EVENT_NAME = 'memory/group_rpl/GCS_XCom::xcom_cache'
                     THEN 'memory/group_rpl/memory_gr'
                     ELSE 'memory_gr_rest'
                   END)
       ) f
       WHERE f.EVENT_NAME != 'memory_gr_rest'\G
       *************************** 1. row ***************************
                           EVENT_NAME: memory/group_rpl/memory_gr
                      SUM(COUNT_ALLOC): 73
                       SUM(COUNT_FREE): 20
          SUM(SUM_NUMBER_OF_BYTES_ALLOC): 1070845
           SUM(SUM_NUMBER_OF_BYTES_FREE): 5670
                     SUM(LOW_COUNT_USED): 0
                  SUM(CURRENT_COUNT_USED): 53
                     SUM(HIGH_COUNT_USED): 56
           SUM(LOW_NUMBER_OF_BYTES_USED): 0
```

```
       SUM(CURRENT_NUMBER_OF_BYTES_USED): 1065175
          SUM(HIGH_NUMBER_OF_BYTES_USED): 1065175
```

18.8 Upgrading Group Replication

This section explains how to upgrade a Group Replication setup. The basic process of upgrading members of a group is the same as upgrading stand-alone instances, see Section 2.10, “Upgrading MySQL” for the actual process of doing upgrade and types available. Choosing between an in-place or logical upgrade depends on the amount of data stored in the group. Usually an in-place upgrade is faster, and therefore is recommended. You should also consult Section 17.5.3, “Upgrading a Replication Topology”.

While you are in the process of upgrading an online group, in order to maximize availability, you might need to have members with different MySQL Server versions running at the same time. Group Replication includes compatibility policies that enable you to safely combine members running different versions of MySQL in the same group during the upgrade procedure. Depending on your group, the effects of these policies might affect the order in which you should upgrade group members. For details, see Section 18.8.1, “Combining Different Member Versions in a Group”.

If your group can be taken fully offline see Section 18.8.2, “Group Replication Offline Upgrade”. If your group needs to remain online, as is common with production deployments, see Section 18.8.3, “Group Replication Online Upgrade” for the different approaches available for upgrading a group with minimal downtime.

18.8.1 Combining Different Member Versions in a Group

Group Replication is versioned according to the MySQL Server version that the Group Replication plugin was bundled with. For example, if a member is running MySQL 5.7.26 then that is the version of the Group Replication plugin. To check the version of MySQL Server on a group member issue:

```
SELECT MEMBER_HOST,MEMBER_PORT,MEMBER_VERSION FROM performance_schema.replication_group_members;
+-------------+-------------+----------------+
| member_host | member_port | member_version |
+-------------+-------------+----------------+
| example.com |    3306     |   8.0.13       |
+-------------+-------------+----------------+
```

For guidance on understanding the MySQL Server version and selecting a version, see Section 2.1.2, “Which MySQL Version and Distribution to Install”.

For optimal compatibility and performance, all members of a group should run the same version of MySQL Server and therefore of Group Replication. However, while you are in the process of upgrading an online group, in order to maximize availability, you might need to have members with different MySQL Server versions running at the same time. Depending on the changes made between the versions of MySQL, you could encounter incompatibilities in this situation. For example, if a feature has been deprecated between major versions, then combining the versions in a group might cause members that rely on the deprecated feature to fail. Conversely, writing to a member running a newer MySQL version while there are read-write members in the group running an older MySQL version might cause issues on members that lack functions introduced in the newer release.

To prevent these issues, Group Replication includes compatibility policies that enable you to safely combine members running different versions of MySQL in the same group. A member applies these policies to decide whether to join the group normally, or join in read-only mode, or not join the group, depending on which choice results in the safe operation of the joining member and of the existing members of the group. In an upgrade scenario, each server must leave the group, be upgraded, and rejoin the group with its new server version. At this point the member applies the policies for its new server version, which might have changed from the policies it applied when it originally joined the group.

As the administrator, you can instruct any server to attempt to join any group by configuring the server appropriately and issuing a `START GROUP_REPLICATION` statement. A decision to join or not join the group, or to join the group in read-only mode, is made and implemented by the joining member itself after you attempt to add it to the group. The joining member receives information on the MySQL Server versions of the current group members, assesses its own compatibility with those members, and applies the policies used in its own MySQL Server version (*not* the policies used by the existing members) to decide whether it is compatible.

The compatibility policies that a joining member applies when attempting to join a group are as follows:

- A member does not join a group if it is running a lower MySQL Server version than the lowest version that the existing group members are running.
- A member joins a group normally if it is running the same MySQL Server version as the lowest version that the existing group members are running.
- A member joins a group but remains in read-only mode if it is running a higher MySQL Server version than the lowest version that the existing group members are running. This behavior only makes a difference when the group is running in multi-primary mode, because in a group that is running in single-primary mode, newly added members default to being read-only in any case.

Members running MySQL 8.0.17 or higher take into account the patch version of the release when checking their compatibility. Members running MySQL 8.0.16 or lower, or MySQL 5.7, only take into account the major version. For example, if you have a group with members all running MySQL version 8.0.13:

- A member that is running MySQL version 5.7 does not join.
- A member running MySQL 8.0.16 joins normally (because it considers the major version).
- A member running MySQL 8.0.17 joins but remains in read-only mode (because it considers the patch version).

Note that joining members running releases before MySQL 5.7.27 check against all group members to find whether their own MySQL Server major version is lower. They therefore fail this check for a group where any members are running MySQL 8.0 releases, and cannot join the group even if it already has other members running MySQL 5.7. From MySQL 5.7.27, joining members only check against the group members that are running the lowest major version, so they can join a mixed version group where other MySQL 5.7 servers are present.

In a multi-primary mode group with members that use different MySQL Server versions, Group Replication automatically manages the read-write and read-only status of members running MySQL 8.0.17 or higher. If a member leaves the group, the members running the version that is now the lowest are automatically set to read-write mode. When you change a group that was running in single-primary mode to run in multi-primary mode, using the `group_replication_switch_to_multi_primary_mode()` function, Group Replication automatically sets members to the correct mode. Members are automatically placed in read-only mode if they are running a higher MySQL server version than the lowest version present in the group, and members running the lowest version are placed in read-write mode.

18.8.1.1 Member Versions During Upgrades

During an online upgrade procedure, if the group is in single-primary mode, all the servers that are not currently offline for upgrading function as they did before. The group elects a new primary whenever necessary, following the election policies described in Section 18.1.3.1, “Single-Primary Mode”. Note that if you require the primary to remain the same throughout (except when it is being upgraded itself), you must first upgrade all of the secondaries to a version higher than or equal to the target primary member version, then upgrade the primary last. The primary cannot remain as the primary unless it is running the lowest MySQL Server version in the group. After the primary has been upgraded, you can use the `group_replication_set_as_primary()` function to reappoint it as the primary.

If the group is in multi-primary mode, fewer online members are available to perform writes during the upgrade procedure, because upgraded members join in read-only mode after their upgrade. From MySQL 8.0.17, this applies to upgrades between patch versions, and for lower releases, this only applies to upgrades between major versions. When all members have been upgraded to the same release, from MySQL 8.0.17, they all change back to read-write mode automatically. For earlier releases, you must set `super_read_only` to `OFF` manually on each member that should function as a primary following the upgrade.

To deal with a problem situation, for example if you have to roll back an upgrade or add extra capacity to a group in an emergency, it is possible to allow a member to join an online group although it is running a lower MySQL Server version than the lowest version in use by other group members. The Group Replication system variable `group_replication_allow_local_lower_version_join` can be used in such situations to override the normal compatibility policies. It is important to note that setting the option to `ON` does not make the new member compatible with the group, and allows it to join the group without any safeguards against incompatible behaviors by the existing members. The option must therefore only be used carefully in specific situations, and you must take additional precautions to avoid the new member failing due to normal group activity. For details of these precautions, see the description for `group_replication_allow_local_lower_version_join`.

18.8.1.2 Group Replication Communication Protocol Version

A replication group uses a Group Replication communication protocol version that can differ from the MySQL Server version of the members. To check the group's communication protocol version, issue the following statement on any member:

```
SELECT group_replication_get_communication_protocol();
```

The return value shows the oldest MySQL Server version that can join this group and use the group's communication protocol. Versions from MySQL 5.7.14 allow compression of messages, and versions from MySQL 8.0.16 also allow fragmentation of messages. Note that the `group_replication_get_communication_protocol()` function returns the minimum MySQL version that the group supports, which might differ from the version number that was passed to the `group_replication_set_communication_protocol()` function, and from the MySQL Server version that is installed on the member where you use the function.

When you upgrade all the members of a replication group to a new MySQL Server release, the Group Replication communication protocol version is not automatically upgraded, in case there is still a requirement to allow members at earlier releases to join. If you do not need to support older members and want to allow the upgraded members to use any added communication capabilities, after the upgrade use the `group_replication_set_communication_protocol()` function to upgrade the communication protocol, specifying the new MySQL Server version to which you have upgraded the members. For more information, see Section 18.5.1.4, “Setting a Group's Communication Protocol Version”.

18.8.2 Group Replication Offline Upgrade

To perform an offline upgrade of a Group Replication group, you remove each member from the group, perform an upgrade of the member and then restart the group as usual. In a multi-primary group you can shutdown the members in any order. In a single-primary group, shutdown each secondary first and then finally the primary. See Section 18.8.3.2, “Upgrading a Group Replication Member” for how to remove members from a group and shutdown MySQL.

Once the group is offline, upgrade all of the members. See Section 2.10, “Upgrading MySQL” for how to perform an upgrade. When all members have been upgraded, restart the members.

If you upgrade all the members of a replication group when they are offline and then restart the group, the members join using the new release's Group Replication communication protocol version, so that becomes the group's communication protocol version. If you have a requirement to allow members at earlier releases to join, you can use the `group_replication_set_communication_protocol()`

function to downgrade the communication protocol version, specifying the MySQL Server version of the prospective group member that has the oldest installed server version.

18.8.3 Group Replication Online Upgrade

When you have a group running which you want to upgrade but you need to keep the group online to serve your application, you need to consider your approach to the upgrade. This section describes the different elements involved in an online upgrade, and various methods of how to upgrade your group.

18.8.3.1 Online Upgrade Considerations

When upgrading an online group you should consider the following points:

- Regardless of the way which you upgrade your group, it is important to disable any writes to group members until they are ready to rejoin the group.
- When a member is stopped, the `super_read_only` variable is set to on automatically, but this change is not persisted.
- When MySQL 5.7.22 or MySQL 8.0.11 tries to join a group running MySQL 5.7.21 or lower it fails to join the group because MySQL 5.7.21 does not send its value of `lower_case_table_names`.

18.8.3.2 Upgrading a Group Replication Member

This section explains the steps required for upgrading a member of a group. This procedure is part of the methods described at Section 18.8.3.3, “Group Replication Online Upgrade Methods”. The process of upgrading a member of a group is common to all methods and is explained first. The way which you join upgraded members can depend on which method you are following, and other factors such as whether the group is operating in single-primary or multi-primary mode. How you upgrade the server instance, using either the in-place or provision approach, does not impact on the methods described here.

The process of upgrading a member consists of removing it from the group, following your chosen method of upgrading the member, and then rejoining the upgraded member to a group. The recommended order of upgrading members in a single-primary group is to upgrade all secondaries, and then upgrade the primary last. If the primary is upgraded before a secondary, a new primary using the older MySQL version is chosen, but there is no need for this step.

To upgrade a member of a group:

- Connect a client to the group member and issue `STOP GROUP_REPLICATION`. Before proceeding, ensure that the member's status is `OFFLINE` by monitoring the `replication_group_members` table.
- Disable Group Replication from starting up automatically so that you can safely connect to the member after upgrading and configure it without it rejoining the group by setting `group_replication_start_on_boot=0`.

Important

If an upgraded member has `group_replication_start_on_boot=1` then it could rejoin the group before you can perform the MySQL upgrade procedure and could result in issues. For example, if the upgrade fails and the server restarts again, then a possibly broken server could try to join the group.

- Stop the member, for example using `mysqladmin shutdown` or the `SHUTDOWN` statement. Any other members in the group continue running.
- Upgrade the member, using the in-place or provisioning approach. See Section 2.10, “Upgrading MySQL” for details. When restarting the upgraded member, because

`group_replication_start_on_boot` is set to 0, Group Replication does not start on the instance, and therefore it does not rejoin the group.

- Once the MySQL upgrade procedure has been performed on the member, `group_replication_start_on_boot` must be set to 1 to ensure Group Replication starts correctly after restart. Restart the member.
- Connect to the upgraded member and issue `START GROUP_REPLICATION`. This rejoins the member to the group. The Group Replication metadata is in place on the upgraded server, therefore there is usually no need to reconfigure Group Replication. The server has to catch up with any transactions processed by the group while the server was offline. Once it has caught up with the group, it becomes an online member of the group.

Note

The longer it takes to upgrade a server, the more time that member is offline and therefore the more time it takes for the server to catch up when added back to the group.

When an upgraded member joins a group which has any member running an earlier MySQL Server version, the upgraded member joins with `super_read_only=on`. This ensures that no writes are made to upgraded members until all members are running the newer version. In a multi-primary mode group, when the upgrade has been completed successfully and the group is ready to process transactions, members that are intended as writeable primaries must be set to read-write mode. From MySQL 8.0.17, when all members of a group have been upgraded to the same release, they all change back to read-write mode automatically. For earlier releases you must set each member manually to read-write mode. Connect to each member and issue:

```
SET GLOBAL super_read_only=OFF;
```

18.8.3.3 Group Replication Online Upgrade Methods

Choose one of the following methods of upgrading a Group Replication group:

Rolling In-Group Upgrade

This method is supported provided that servers running a newer version are not generating workload to the group while there are still servers with an older version in it. In other words servers with a newer version can join the group only as secondaries. In this method there is only ever one group, and each server instance is removed from the group, upgraded and then rejoined to the group.

This method is well suited to single-primary groups. When the group is operating in single-primary mode, if you require the primary to remain the same throughout (except when it is being upgraded itself), it should be the last member to be upgraded. The primary cannot remain as the primary unless it is running the lowest MySQL Server version in the group. After the primary has been upgraded, you can use the `group_replication_set_as_primary()` function to reappoint it as the primary. If you do not mind which member is the primary, the members can be upgraded in any order. The group elects a new primary whenever necessary from among the members running the lowest MySQL Server version, following the election policies described in Section 18.1.3.1, “Single-Primary Mode”.

For groups operating in multi-primary mode, during a rolling in-group upgrade the number of primaries is decreased, causing a reduction in write availability. This is because if a member joins a group when it is running a higher MySQL Server version than the lowest version that the existing group members are running, it automatically remains in read-only mode (`super_read_only=ON`). Note that members running MySQL 8.0.17 or higher take into account the patch version of the release when checking this, but members running MySQL 8.0.16 or lower, or MySQL 5.7, only take into account the major version. When all members have been upgraded to the same release, from MySQL 8.0.17, they all change back to read-write mode automatically. For earlier releases, you must set `super_read_only=OFF` manually on each member that should function as a primary following the upgrade.

For full information on version compatibility in a group and how this influences the behavior of a group during an upgrade process, see Section 18.8.1, “Combining Different Member Versions in a Group” .

Rolling Migration Upgrade

In this method you remove members from the group, upgrade them and then create a second group using the upgraded members. For groups operating in multi-primary mode, during this process the number of primaries is decreased, causing a reduction in write availability. This does not impact groups operating in single-primary mode.

Because the group running the older version is online while you are upgrading the members, you need the group running the newer version to catch up with any transactions executed while the members were being upgraded. Therefore one of the servers in the new group is configured as a replica of a primary from the older group. This ensures that the new group catches up with the older group. Because this method relies on an asynchronous replication channel which is used to replicate data from one group to another, it is supported under the same assumptions and requirements of asynchronous source-replica replication, see Chapter 17, *Replication*. For groups operating in single-primary mode, the asynchronous replication connection to the old group must send data to the primary in the new group, for a multi-primary group the asynchronous replication channel can connect to any primary.

The process is to:

- remove members from the original group running the older server version one by one, see Section 18.8.3.2, “Upgrading a Group Replication Member”
- upgrade the server version running on the member, see Section 2.10, “Upgrading MySQL”. You can either follow an in-place or provision approach to upgrading.
- create a new group with the upgraded members, see Chapter 18, *Group Replication*. In this case you need to configure a new group name on each member (because the old group is still running and using the old name), bootstrap an initial upgraded member, and then add the remaining upgraded members.
- set up an asynchronous replication channel between the old group and the new group, see Section 17.1.3.4, “Setting Up Replication Using GTIDs”. Configure the older primary to function as the asynchronous replication source server and the new group member as a GTID-based replica.

Before you can redirect your application to the new group, you must ensure that the new group has a suitable number of members, for example so that the group can handle the failure of a member. Issue `SELECT * FROM performance_schema.replication_group_members` and compare the initial group size and the new group size. Wait until all data from the old group is propagated to the new group and then drop the asynchronous replication connection and upgrade any missing members.

Rolling Duplication Upgrade

In this method you create a second group consisting of members which are running the newer version, and the data missing from the older group is replicated to the newer group. This assumes that you have enough servers to run both groups simultaneously. Due to the fact that during this process the number of primaries is *not* decreased, for groups operating in multi-primary mode there is no reduction in write availability. This makes rolling duplication upgrade well suited to groups operating in multi-primary mode. This does not impact groups operating in single-primary mode.

Because the group running the older version is online while you are provisioning the members in the new group, you need the group running the newer version to catch up with any transactions executed while the members were being provisioned. Therefore one of the servers in the new group is configured as a replica of a primary from the older group. This ensures that the new group catches up with the older group. Because this method relies on an asynchronous replication channel which is used to replicate data from one group to another, it is supported under the same assumptions and requirements of asynchronous source-replica replication, see Chapter 17, *Replication*. For groups

operating in single-primary mode, the asynchronous replication connection to the old group must send data to the primary in the new group, for a multi-primary group the asynchronous replication channel can connect to any primary.

The process is to:

- deploy a suitable number of members so that the group running the newer version can handle failure of a member
- take a backup of the existing data from a member of the group
- use the backup from the older member to provision the members of the new group, see Section 18.8.3.4, “Group Replication Upgrade with `mysqlbackup`” for one method.

Note

You must restore the backup to the same version of MySQL which the backup was taken from, and then perform an in-place upgrade. For instructions, see Section 2.10, “Upgrading MySQL”.

- create a new group with the upgraded members, see Chapter 18, *Group Replication*. In this case you need to configure a new group name on each member (because the old group is still running and using the old name), bootstrap an initial upgraded member, and then add the remaining upgraded members.
- set up an asynchronous replication channel between the old group and the new group, see Section 17.1.3.4, “Setting Up Replication Using GTIDs”. Configure the older primary to function as the asynchronous replication source server and the new group member as a GTID-based replica.

Once the ongoing data missing from the newer group is small enough to be quickly transferred, you must redirect write operations to the new group. Wait until all data from the old group is propagated to the new group and then drop the asynchronous replication connection.

18.8.3.4 Group Replication Upgrade with `mysqlbackup`

As part of a provisioning approach you can use MySQL Enterprise Backup to copy and restore the data from a group member to new members. However you cannot use this technique to directly restore a backup taken from a member running an older version of MySQL to a member running a newer version of MySQL. The solution is to restore the backup to a new server instance which is running the same version of MySQL as the member which the backup was taken from, and then upgrade the instance. This process consists of:

- Take a backup from a member of the older group using `mysqlbackup`. See Section 18.5.6, “Using MySQL Enterprise Backup with Group Replication”.
- Deploy a new server instance, which must be running the same version of MySQL as the older member where the backup was taken.
- Restore the backup from the older member to the new instance using `mysqlbackup`.
- Upgrade MySQL on the new instance, see Section 2.10, “Upgrading MySQL”.

Repeat this process to create a suitable number of new instances, for example to be able to handle a failover. Then join the instances to a group based on the Section 18.8.3.3, “Group Replication Online Upgrade Methods”.`

18.9 Group Replication System Variables

This section lists the system variables that are specific to the Group Replication plugin. Every configuration option is prefixed with "`group_replication`".

Note

InnoDB Cluster uses Group Replication, but the default values of the Group Replication system variables may differ from the defaults documented in this section. For example, in InnoDB Cluster, the default value of `group_replication_communication_stack` is `MYSQL`, not `XCOM` as it is for a default Group Replication implementation.

For more information, see MySQL InnoDB Cluster.

Some system variables on a Group Replication group member, including some Group Replication-specific system variables and some general system variables, are group-wide configuration settings. These system variables must have the same value on all group members, and require a full reboot of the group (a bootstrap by a server with `group_replication_bootstrap_group=ON`) in order for the value change to take effect. For instructions to reboot a group where every member has been stopped, see Section 18.5.2, “Restarting a Group”.

If a running group has a value set for a group-wide configuration setting, and a joining member has a different value set for that system variable, the joining member cannot join the group until the value is changed to match. If the group has a value set for one of these system variables, and the joining member does not support the system variable, it cannot join the group.

The following system variables are group-wide configuration settings:

- `group_replication_single_primary_mode`
- `group_replication_enforce_update_everywhere_checks`
- `group_replication_gtid_assignment_block_size`
- `group_replication_view_change_uuid`
- `group_replication_paxos_single_leader`
- `group_replication_communication_stack` (a special case not policed by Group Replication's own checks; see the system variable description for details)
- `default_table_encryption`
- `lower_case_table_names`
- `transaction_write_set_extraction` (deprecated from MySQL 8.0.26)

Group-wide configuration settings cannot be changed by the usual methods while Group Replication is running. However, from MySQL 8.0.16, you can use the `group_replication_switch_to_single_primary_mode()` and `group_replication_switch_to_multi_primary_mode()` functions to change the values of `group_replication_single_primary_mode` and `group_replication_enforce_update_everywhere_checks` while the group is still running. For more information, see Section 18.5.1.2, “Changing a Group's Mode”.

Most system variables for Group Replication can have different values on different group members. For the following system variables, it is advisable to set the same value on all members of a group in order to avoid unnecessary rollback of transactions, failure of message delivery, or failure of message recovery:

- `group_replication_auto_increment_increment`
- `group_replication_communication_max_message_size`
- `group_replication_compression_threshold`

- group_replication_message_cache_size
- group_replication_transaction_size_limit

Most system variables for Group Replication are described as dynamic, and their values can be changed while the server is running. However, in most cases, the change only takes effect after you stop and restart Group Replication on the group member using a STOP GROUP_REPLICATION statement followed by a START GROUP_REPLICATION statement. Changes to the following system variables take effect without stopping and restarting Group Replication:

- group_replication_advertise_recovery_endpoints
- group_replication_autorejoin_tries
- group_replication_consistency
- group_replication_exit_state_action
- group_replication_flow_control_applier_threshold
- group_replication_flow_control_certifier_threshold
- group_replication_flow_control_hold_percent
- group_replication_flow_control_max_quota
- group_replication_flow_control_member_quota_percent
- group_replication_flow_control_min_quota
- group_replication_flow_control_min_recovery_quota
- group_replication_flow_control_mode
- group_replication_flow_control_period
- group_replication_flow_control_release_percent
- group_replication_force_members
- group_replication_ip_allowlist
- group_replication_ip_whitelist
- group_replication_member_expel_timeout
- group_replication_member_weight
- group_replication_transaction_size_limit
- group_replication_unreachable_majority_timeout

When you change the values of any Group Replication system variables, bear in mind that if there is a point where Group Replication is stopped on every member at once by a STOP GROUP_REPLICATION statement or system shutdown, the group must be restarted by bootstrapping as if it was being started for the first time. For instructions to do this safely, see Section 18.5.2, “Restarting a Group”. In the case of group-wide configuration settings, this is required, but if you are changing other settings, try to ensure at least one member is running at all times.

Important

- A number of system variables for Group Replication are not completely validated during server startup if they are passed

as command line arguments to the server. These system variables include `group_replication_group_name`, `group_replication_single_primary_mode`, `group_replication_force_members`, the SSL variables, and the flow control system variables. They are only fully validated after the server has started.

- System variables for Group Replication that specify IP addresses or host names for group members are not validated until a `START GROUP_REPLICATION` statement is issued. Group Replication's Group Communication System (GCS) is not available to validate the values until that point.

The system variables that are specific to the Group Replication plugin are as follows:

- `group_replication_advertise_recovery_endpoints`

Command-Line Format	`--group-replication-advertise-recovery-endpoints=value`
Introduced	8.0.21
System Variable	`group_replication_advertise_recovery_endpoints`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String
Default Value	`DEFAULT`

The value of this system variable can be changed while Group Replication is running. The change takes effect immediately on the member. However, a joining member that already received the previous value of the system variable continues to use that value. Only members that join after the value change receive the new value.

`group_replication_advertise_recovery_endpoints` specifies how a joining member can establish a connection to an existing member for state transfer for distributed recovery. The connection is used for both remote cloning operations and state transfer from the donor's binary log.

A value of `DEFAULT`, which is the default setting, means joining members use the existing member's standard SQL client connection, as specified by MySQL Server's `hostname` and `port` system variables. If an alternative port number is specified by the `report_port` system variable, that one is used instead. The Performance Schema table `replication_group_members` shows this connection's address and port number in the `MEMBER_HOST` and `MEMBER_PORT` fields. This is the behavior of group members at releases up to and including MySQL 8.0.20.

Instead of `DEFAULT`, you can specify one or more distributed recovery endpoints, which the existing member advertises to joining members for them to use. Offering distributed recovery endpoints lets administrators control distributed recovery traffic separately from regular MySQL client connections to the group members. Joining members try each of the endpoints in turn in the order they are specified on the list.

Specify the distributed recovery endpoints as a comma-separated list of IP addresses and port numbers, for example:

```
group_replication_advertise_recovery_endpoints= "127.0.0.1:3306,127.0.0.1:4567,[::1]:3306,localhost:3306"
```

IPv4 and IPv6 addresses and host names can be used in any combination. IPv6 addresses must be specified in square brackets. Host names must resolve to a local IP address. Wildcard address formats cannot be used, and you cannot specify an empty list. Note that the standard SQL client

connection is not automatically included on a list of distributed recovery endpoints. If you want to use it as an endpoint, you must include it explicitly on the list.

For details of how to select IP addresses and ports as distributed recovery endpoints, and how joining members use them, see Selecting addresses for distributed recovery endpoints. A summary of the requirements is as follows:

- The IP addresses do not have to be configured for MySQL Server, but they do have to be assigned to the server.
- The ports do have to be configured for MySQL Server using the `port`, `report_port`, or `admin_port` system variable.
- Appropriate permissions are required for the replication user for distributed recovery if the `admin_port` is used.
- The IP addresses do not need to be added to the Group Replication allowlist specified by the `group_replication_ip_allowlist` or `group_replication_ip_whitelist` system variable.
- The SSL requirements for the connection are as specified by the `group_replication_recovery_ssl_*` options.

- `group_replication_allow_local_lower_version_join`

Command-Line Format	`--group-replication-allow-local-lower-version-join[={OFF|ON}]`
System Variable	`group_replication_allow_local_lower_version`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Boolean
Default Value	`OFF`

The value of this system variable can be changed while Group Replication is running, but the change only takes effect after you stop and restart Group Replication on the group member.

`group_replication_allow_local_lower_version_join` allows the current server to join the group even if it is running a lower MySQL Server version than the group. With the default setting `OFF`, servers are not permitted to join a replication group if they are running a lower version than the existing group members. This standard policy ensures that all members of a group are able to exchange messages and apply transactions. Note that members running MySQL 8.0.17 or higher take into account the patch version of the release when checking their compatibility. Members running MySQL 8.0.16 or lower, or MySQL 5.7, only take into account the major version.

Set `group_replication_allow_local_lower_version_join` to `ON` only in the following scenarios:

- A server must be added to the group in an emergency in order to improve the group's fault tolerance, and only older versions are available.
- You want to roll back an upgrade for one or more replication group members without shutting down the whole group and bootstrapping it again.

Warning

Setting this option to `ON` does not make the new member compatible with the group, and allows it to join the group without any safeguards against

incompatible behaviors by the existing members. To ensure the new member's correct operation, take *both* of the following precautions:

1. Before the server running the lower version joins the group, stop all writes on that server.
2. From the point where the server running the lower version joins the group, stop all writes on the other servers in the group.

Without these precautions, the server running the lower version is likely to experience difficulties and terminate with an error.

- `group_replication_auto_increment_increment`

Command-Line Format	`--group-replication-auto-increment-increment=#`
System Variable	`group_replication_auto_increment_increment`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`7`
Minimum Value	`1`
Maximum Value	`65535`

This system variable should have the same value on all group members. You cannot change the value of this system variable while Group Replication is running. You must stop Group Replication, change the value of the system variable, then restart Group Replication, on each of the group members. During this process, the value of the system variable is permitted to differ between group members, but some transactions on group members might be rolled back.

`group_replication_auto_increment_increment` determines the interval between successive values for auto-incremented columns for transactions that execute on this server instance. Adding an interval avoids the selection of duplicate auto-increment values for writes on group members, which causes rollback of transactions. The default value of 7 represents a balance between the number of usable values and the permitted maximum size of a replication group (9 members). If your group has more or fewer members, you can set this system variable to match the expected number of group members before Group Replication is started.

Important

Setting `group_replication_auto_increment_increment` has no effect when `group_replication_single_primary_mode` is `ON`.

When Group Replication is started on a server instance, the value of the server system variable `auto_increment_increment` is changed to this value, and the value of the server system variable `auto_increment_offset` is changed to the server ID. The changes are reverted when Group Replication is stopped. These changes are only made and reverted if `auto_increment_increment` and `auto_increment_offset` each have their default value of 1. If their values have already been modified from the default, Group Replication does not alter them. In MySQL 8.0, the system variables are also not modified when Group Replication is in single-primary mode, where only one server writes.

- `group_replication_autorejoin_tries`

Command-Line Format	`--group-replication-autorejoin-tries=#`
Introduced	8.0.16
System Variable	`group_replication_autorejoin_tries`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value (≥ 8.0.21)	`3`
Default Value (≤ 8.0.20)	`0`
Minimum Value	`0`
Maximum Value	`2016`

The value of this system variable can be changed while Group Replication is running, and the change takes effect immediately. The system variable's current value is read when an issue occurs that means the behavior is needed.

`group_replication_autorejoin_tries` specifies the number of tries that a member makes to automatically rejoin the group if it is expelled, or if it is unable to contact a majority of the group before the `group_replication_unreachable_majority_timeout` setting is reached. When the member's expulsion or unreachable majority timeout is reached, it makes an attempt to rejoin (using the current plugin option values), then continues to make further auto-rejoin attempts up to the specified number of tries. After an unsuccessful auto-rejoin attempt, the member waits 5 minutes before the next try. If the specified number of tries is exhausted without the member rejoining or being stopped, the member proceeds to the action specified by the `group_replication_exit_state_action` system variable.

Up to MySQL 8.0.20, the default setting is 0, meaning that the member does not try to rejoin automatically. From MySQL 8.0.21, the default setting is 3, meaning that the member automatically makes 3 attempts to rejoin the group, with 5 minutes between each. You can specify a maximum of 2016 tries.

During and between auto-rejoin attempts, a member remains in super read only mode and does not accept writes, but reads can still be made on the member, with an increasing likelihood of stale reads over time. If you cannot tolerate the possibility of stale reads for any period of time, set `group_replication_autorejoin_tries` to 0. For more information on the auto-rejoin feature, and considerations when choosing a value for this option, see Section 18.7.7.3, “Auto-Rejoin”.

- `group_replication_bootstrap_group`

Command-Line Format	`--group-replication-bootstrap-group[={OFF|ON}]`
System Variable	`group_replication_bootstrap_group`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Boolean

Default Value	OFF

group_replication_bootstrap_group configures this server to bootstrap the group. This system variable must *only* be set on one server, and *only* when starting the group for the first time or restarting the entire group. After the group has been bootstrapped, set this option to OFF. It should be set to OFF both dynamically and in the configuration files. Starting two servers or restarting one server with this option set while the group is running may lead to an artificial split brain situation, where two independent groups with the same name are bootstrapped.

For instructions to bootstrap a group for the first time, see Section 18.2.1.5, “Bootstrapping the Group”. For instructions to safely bootstrap a group where transactions have been executed and certified, see Section 18.5.2, “Restarting a Group”.

- group_replication_clone_threshold

Command-Line Format	--group-replication-clone-threshold=#
Introduced	8.0.17
System Variable	group_replication_clone_threshold
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	9223372036854775807
Minimum Value	1
Maximum Value	9223372036854775807
Unit	transactions

The value of this system variable can be changed while Group Replication is running, but the change only takes effect after you stop and restart Group Replication on the group member.

group_replication_clone_threshold specifies the transaction gap, as a number of transactions, between the existing member (donor) and the joining member (recipient) that triggers the use of a remote cloning operation for state transfer to the joining member during the distributed recovery process. If the transaction gap between the joining member and a suitable donor exceeds the threshold, Group Replication begins distributed recovery with a remote cloning operation. If the transaction gap is below the threshold, or if the remote cloning operation is not technically possible, Group Replication proceeds directly to state transfer from a donor's binary log.

Warning

Do not use a low setting for group_replication_clone_threshold in an active group. If a number of transactions above the threshold takes place in the group while the remote cloning operation is in progress, the joining member triggers a remote cloning operation again after restarting, and could continue this indefinitely. To avoid this situation, ensure that you set the threshold to a number higher than the number of transactions that you would expect to occur in the group during the time taken for the remote cloning operation.

To use this function, both the donor and the joining member must be set up beforehand to support cloning. For instructions, see Section 18.5.4.2, “Cloning for Distributed Recovery”. When a remote cloning operation is carried out, Group Replication manages it for you, including the required server restart, provided that group_replication_start_on_boot=ON is set. If not, you must restart the server manually. The remote cloning operation replaces the existing data dictionary on the joining

member, but Group Replication checks and does not proceed if the joining member has additional transactions that are not present on the other group members, because these transactions would be erased by the cloning operation.

The default setting (which is the maximum permitted sequence number for a transaction in a GTID) means that state transfer from a donor's binary log is virtually always attempted rather than cloning. However, note that Group Replication always attempts to execute a cloning operation, regardless of your threshold, if state transfer from a donor's binary log is impossible, for example because the transactions needed by the joining member are not available in the binary logs on any existing group member. If you do not want to use cloning at all in your replication group, do not install the clone plugin on the members.

- `group_replication_communication_debug_options`

Command-Line Format	`--group-replication-communication-debug-options=value`
System Variable	`group_replication_communication_debug_optio`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String
Default Value	`GCS_DEBUG_NONE`

Valid Values	`GCS_DEBUG_NONE`
	`GCS_DEBUG_BASIC`
	`GCS_DEBUG_TRACE`
	`XCOM_DEBUG_BASIC`
	`XCOM_DEBUG_TRACE`
	`GCS_DEBUG_ALL`

The value of this system variable can be changed while Group Replication is running, and the change takes effect immediately.

`group_replication_communication_debug_options` configures the level of debugging messages to provide for the different Group Replication components, such as the Group Communication System (GCS) and the group communication engine (XCom, a Paxos variant). The debug information is stored in the `GCS_DEBUG_TRACE` file in the data directory.

The set of available options, specified as strings, can be combined. The following options are available:

- `GCS_DEBUG_NONE` disables all debugging levels for both GCS and XCom.
- `GCS_DEBUG_BASIC` enables basic debugging information in GCS.
- `GCS_DEBUG_TRACE` enables trace information in GCS.
- `XCOM_DEBUG_BASIC` enables basic debugging information in XCom.
- `XCOM_DEBUG_TRACE` enables trace information in XCom.

- `GCS_DEBUG_ALL` enables all debugging levels for both GCS and XCom.

Setting the debug level to `GCS_DEBUG_NONE` only has an effect when provided without any other option. Setting the debug level to `GCS_DEBUG_ALL` overrides all other options.

- `group_replication_communication_max_message_size`

Command-Line Format	`--group-replication-communication-max-message-size=#`
Introduced	8.0.16
System Variable	`group_replication_communication_max_message_si`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`10485760`
Minimum Value	`0`
Maximum Value	`1073741824`
Unit	bytes

This system variable should have the same value on all group members. You cannot change the value of this system variable while Group Replication is running. You must stop Group Replication, change the value of the system variable, then restart Group Replication, on each of the group members. During this process, the value of the system variable is permitted to differ between group members, but some transactions on group members might be rolled back.

`group_replication_communication_max_message_size` specifies a maximum message size for Group Replication communications. Messages greater than this size are automatically split into fragments that are sent separately and reassembled by the recipients. For more information, see Section 18.7.5, “Message Fragmentation”.

A maximum message size of 10485760 bytes (10 MiB) is set by default, which means that fragmentation is used by default in releases from MySQL 8.0.16. The greatest permitted value is the same as the maximum value of the `replica_max_allowed_packet` and `slave_max_allowed_packet` system variables, which is 1073741824 bytes (1 GB). The setting for `group_replication_communication_max_message_size` must be less than the `replica_max_allowed_packet` or `slave_max_allowed_packet` setting, because the applier thread cannot handle message fragments larger than the maximum permitted packet size. To switch off fragmentation, specify a zero value for `group_replication_communication_max_message_size`.

In order for members of a replication group to use fragmentation, the group's communication protocol version must be MySQL 8.0.16 or above. Use the `group_replication_get_communication_protocol()` function to view the group's communication protocol version. If a lower version is in use, group members do not fragment messages. You can use the `group_replication_set_communication_protocol()` function to set the group's communication protocol to a higher version if all group members support it. For more information, see Section 18.5.1.4, “Setting a Group's Communication Protocol Version”.

- `group_replication_communication_stack`

Introduced	8.0.27
System Variable	`group_replication_communication_stack`

Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String
Default Value	XCOM
Valid Values	XCOM MYSQL

Note

This system variable is effectively a group-wide configuration setting, and a full reboot of the replication group is required for a change to take effect.

group_replication_communication_stack specifies whether the XCom communication stack or the MySQL communication stack is to be used to establish group communication connections between members. The XCom communication stack is Group Replication's own implementation, as used always in releases before MySQL 8.0.27, and does not support authentication or network namespaces. The MySQL communication stack is MySQL Server's native implementation, with support for authentication and network namespaces, and access to new security functions immediately on release. All members of a group must use the same communication stack.

When you use MySQL's communication stack in place of XCom's, MySQL Server establishes each connection between group members using its own authentication and encryption protocols.

Note

If you are using InnoDB Cluster, the default value of group_replication_communication_stack is MYSQL.

For more information, see MySQL InnoDB Cluster.

Additional configuration is required when you set up a group to use MySQL's communication stack; see Section 18.6.1, "Communication Stack for Connection Security Management".

group_replication_communication_stack is effectively a group-wide configuration setting, and the setting must be the same on all group members. However, this is not policed by Group Replication's own checks for group-wide configuration settings. A member with a different value from the rest of the group cannot communicate with the other members at all, because the communication protocols are incompatible, so it cannot exchange information about its configuration settings.

This means that although the value of the system variable can be changed while Group Replication is running, and takes effect after you restart Group Replication on the group member, the member still cannot rejoin the group until the setting has been changed on all the members. You must therefore stop Group Replication on all of the members and change the value of the system variable on them all before you can restart the group. Because all of the members are stopped, a full reboot of the group (a bootstrap by a server with group_replication_bootstrap_group=ON) is required in order for the value change to take effect. For instructions to migrate from one communication stack to another, see Section 18.6.1, "Communication Stack for Connection Security Management".

- group_replication_components_stop_timeout

Command-Line Format	--group-replication-components-stop-timeout=#
System Variable	group_replication_components_stop_timeout
Scope	Global

Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value (≥ 8.0.27)	300
Default Value (≤ 8.0.26)	31536000
Minimum Value	2
Maximum Value	31536000
Unit	seconds

The value of this system variable can be changed while Group Replication is running, but the change takes effect only after you stop and restart Group Replication on the group member.

group_replication_components_stop_timeout specifies the time, in seconds, for which Group Replication waits for each of its modules to complete ongoing processes while shutting down. The component timeout applies after a STOP GROUP_REPLICATION statement is issued, which happens automatically during server restart or auto-rejoin.

The timeout is used to resolve situations in which Group Replication components cannot be stopped normally, which might happen if a member is expelled from the group while it is in an error state, or while a process such as MySQL Enterprise Backup is holding a global lock on tables on the member. In such situations, the member cannot stop the applier thread or complete the distributed recovery process to rejoin. STOP GROUP_REPLICATION does not complete until either the situation is resolved (for example, by the lock being released), or the component timeout expires and the modules are shut down regardless of their status.

Before MySQL 8.0.27, the default component timeout is 31536000 seconds, or 365 days. With this setting, the component timeout does not help in situations such as those described, so a lower setting is recommended. Beginning with MySQL 8.0.27, the default value is 300 seconds, so that Group Replication components are stopped after 5 minutes if the situation is not resolved before that time, allowing the member to be restarted and to rejoin.

- group_replication_compression_threshold

Command-Line Format	--group-replication-compression-threshold=#
System Variable	group_replication_compression_threshold
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	1000000
Minimum Value	0
Maximum Value	4294967295
Unit	bytes

The threshold value in bytes above which compression is applied to messages sent between group members. If this system variable is set to zero, compression is disabled. The value of group_replication_compression_threshold should be the same on all group members.

Group Replication uses the LZ4 compression algorithm to compress messages sent in the group. Note that the maximum supported input size for the LZ4 compression algorithm is 2113929216 bytes. This limit is lower than the maximum possible value for the

`group_replication_compression_threshold` system variable, which is matched to the maximum message size accepted by XCom. With the LZ4 compression algorithm, do not set a value greater than 2113929216 bytes for `group_replication_compression_threshold`, because transactions above this size cannot be committed when message compression is enabled.

For more information, see Section 18.7.4, “Message Compression”.

- `group_replication_consistency`

Command-Line Format	`--group-replication-consistency=value`
Introduced	8.0.14
System Variable	`group_replication_consistency`
Scope	Global, Session
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Enumeration
Default Value	`EVENTUAL`

Valid Values	`EVENTUAL`
	`BEFORE_ON_PRIMARY_FAILOVER`
	`BEFORE`
	`AFTER`
	`BEFORE_AND_AFTER`

The value of this system variable can be changed while Group Replication is running. `group_replication_consistency` is a server system variable rather than a Group Replication plugin-specific variable, so a restart of Group Replication is not required for the change to take effect. Changing the session value of the system variable takes effect immediately, and changing the global value takes effect for new sessions that start after the change. The `GROUP_REPLICATION_ADMIN` privilege is required to change the global setting for this system variable.

`group_replication_consistency` controls the transaction consistency guarantee which a group provides. You can configure the consistency globally or per transaction. `group_replication_consistency` also configures the fencing mechanism used by newly elected primaries in single primary groups. The effect of the variable must be considered for both read only (RO) and read write (RW) transactions. The following list shows the possible values of this variable, in order of increasing transaction consistency guarantee:

- `EVENTUAL`

  Both RO and RW transactions do not wait for preceding transactions to be applied before executing. This was the behavior of Group Replication before this variable was added. A RW transaction does not wait for other members to apply a transaction. This means that a transaction could be externalized on one member before the others. This also means that in the event of a primary failover, the new primary can accept new RO and RW transactions before the previous primary transactions are all applied. RO transactions could result in outdated values, RW transactions could result in a rollback due to conflicts.

- `BEFORE_ON_PRIMARY_FAILOVER`

  New RO or RW transactions with a newly elected primary that is applying backlog from the old primary are held (not applied) until any backlog has been applied. This ensures that when a primary failover happens, intentionally or not, clients always see the latest value on the primary.

This guarantees consistency, but means that clients must be able to handle the delay in the event that a backlog is being applied. Usually this delay should be minimal, but does depend on the size of the backlog.

- `BEFORE`

  A RW transaction waits for all preceding transactions to complete before being applied. A RO transaction waits for all preceding transactions to complete before being executed. This ensures that this transaction reads the latest value by only affecting the latency of the transaction. This reduces the overhead of synchronization on every RW transaction, by ensuring synchronization is used only on RO transactions. This consistency level also includes the consistency guarantees provided by `BEFORE_ON_PRIMARY_FAILOVER`.

- `AFTER`

  A RW transaction waits until its changes have been applied to all of the other members. This value has no effect on RO transactions. This mode ensures that when a transaction is committed on the local member, any subsequent transaction reads the written value or a more recent value on any group member. Use this mode with a group that is used for predominantly RO operations to ensure that applied RW transactions are applied everywhere once they commit. This could be used by your application to ensure that subsequent reads fetch the latest data which includes the latest writes. This reduces the overhead of synchronization on every RO transaction, by ensuring synchronization is used only on RW transactions. This consistency level also includes the consistency guarantees provided by `BEFORE_ON_PRIMARY_FAILOVER`.

- `BEFORE_AND_AFTER`

  A RW transaction waits for 1) all preceding transactions to complete before being applied and 2) until its changes have been applied on other members. A RO transaction waits for all preceding transactions to complete before execution takes place. This consistency level also includes the consistency guarantees provided by `BEFORE_ON_PRIMARY_FAILOVER`.

For more information, see Section 18.5.3, “Transaction Consistency Guarantees”.

- `group_replication_enforce_update_everywhere_checks`

Command-Line Format	`--group-replication-enforce-update-everywhere-checks[={OFF|ON}]`
System Variable	`group_replication_enforce_update_everywhere_ch`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Boolean
Default Value	`OFF`

Note

This system variable is a group-wide configuration setting, and a full reboot of the replication group is required for a change to take effect.

`group_replication_enforce_update_everywhere_checks` enables or disables strict consistency checks for multi-primary update everywhere. The default is that checks are disabled. In single-primary mode, this option must be disabled on all group members. In multi-primary mode,

when this option is enabled, statements are checked as follows to ensure they are compatible with multi-primary mode:

- If a transaction is executed under the `SERIALIZABLE` isolation level, then its commit fails when synchronizing itself with the group.
- If a transaction executes against a table that has foreign keys with cascading constraints, then the transaction fails to commit when synchronizing itself with the group.

This system variable is a group-wide configuration setting. It must have the same value on all group members, cannot be changed while Group Replication is running, and requires a full reboot of the group (a bootstrap by a server with `group_replication_bootstrap_group=ON`) in order for the value change to take effect. For instructions to safely bootstrap a group where transactions have been executed and certified, see Section 18.5.2, "Restarting a Group".

If the group has a value set for this system variable, and a joining member has a different value set for the system variable, the joining member cannot join the group until the value is changed to match. If the group members have a value set for this system variable, and the joining member does not support the system variable, it cannot join the group.

From MySQL 8.0.16, you can use the `group_replication_switch_to_single_primary_mode()` and `group_replication_switch_to_multi_primary_mode()` functions to change the value of this system variable while the group is still running. For more information, see Section 18.5.1.2, "Changing a Group's Mode".

- `group_replication_exit_state_action`

Command-Line Format	`--group-replication-exit-state-action=value`
Introduced	8.0.12
System Variable	`group_replication_exit_state_action`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Enumeration
Default Value (≥ 8.0.16)	`READ_ONLY`
Default Value (≥ 8.0.12, ≤ 8.0.15)	`ABORT_SERVER`

Valid Values (≥ 8.0.18)	`ABORT_SERVER`
	`OFFLINE_MODE`
	`READ_ONLY`

Valid Values (≥ 8.0.12, ≤ 8.0.17)	`ABORT_SERVER` `READ_ONLY`

The value of this system variable can be changed while Group Replication is running, and the change takes effect immediately. The system variable's current value is read when an issue occurs that means the behavior is needed.

`group_replication_exit_state_action` configures how Group Replication behaves when this server instance leaves the group unintentionally, for example after encountering an applier error, or in the case of a loss of majority, or when another member of the group expels it due to a suspicion timing out. The timeout period for a member to leave the group in the case of a loss of majority is set by the `group_replication_unreachable_majority_timeout` system variable, and

the timeout period for suspicions is set by the `group_replication_member_expel_timeout` system variable. Note that an expelled group member does not know that it was expelled until it reconnects to the group, so the specified action is only taken if the member manages to reconnect, or if the member raises a suspicion on itself and expels itself.

When a group member is expelled due to a suspicion timing out or a loss of majority, if the member has the `group_replication_autorejoin_tries` system variable set to specify a number of auto-rejoin attempts, it first makes the specified number of attempts while in super read only mode, and then follows the action specified by `group_replication_exit_state_action`. Auto-rejoin attempts are not made in case of an applier error, because these are not recoverable.

When `group_replication_exit_state_action` is set to `READ_ONLY`, if the member exits the group unintentionally or exhausts its auto-rejoin attempts, the instance switches MySQL to super read only mode (by setting the system variable `super_read_only` to `ON`). The `READ_ONLY` exit action was the behavior for MySQL 8.0 releases before the system variable was introduced, and became the default again from MySQL 8.0.16.

When `group_replication_exit_state_action` is set to `OFFLINE_MODE`, if the member exits the group unintentionally or exhausts its auto-rejoin attempts, the instance switches MySQL to offline mode (by setting the system variable `offline_mode` to `ON`). In this mode, connected client users are disconnected on their next request and connections are no longer accepted, with the exception of client users that have the `CONNECTION_ADMIN` privilege (or the deprecated `SUPER` privilege). Group Replication also sets the system variable `super_read_only` to `ON`, so clients cannot make any updates, even if they have connected with the `CONNECTION_ADMIN` or `SUPER` privilege. The `OFFLINE_MODE` exit action is available from MySQL 8.0.18.

When `group_replication_exit_state_action` is set to `ABORT_SERVER`, if the member exits the group unintentionally or exhausts its auto-rejoin attempts, the instance shuts down MySQL. This setting was the default from MySQL 8.0.12, when the system variable was added, to MySQL 8.0.15 inclusive.

Important

If a failure occurs before the member has successfully joined the group, the specified exit action *is not taken*. This is the case if there is a failure during the local configuration check, or a mismatch between the configuration of the joining member and the configuration of the group. In these situations, the `super_read_only` system variable is left with its original value, connections continue to be accepted, and the server does not shut down MySQL. To ensure that the server cannot accept updates when Group Replication did not start, we therefore recommend that `super_read_only=ON` is set in the server's configuration file at startup, which Group Replication changes to `OFF` on primary members after it has been started successfully. This safeguard is particularly important when the server is configured to start Group Replication on server boot (`group_replication_start_on_boot=ON`), but it is also useful when Group Replication is started manually using a `START GROUP_REPLICATION` command.

For more information on using this option, and the full list of situations in which the exit action is taken, see Section 18.7.7.4, “Exit Action”.

- `group_replication_flow_control_applier_threshold`

Command-Line Format	`--group-replication-flow-control-applier-threshold=#`
System Variable	`group_replication_flow_control_applier_thresho`
Scope	Global
Dynamic	Yes

SET_VAR Hint Applies	No
Type	Integer
Default Value	25000
Minimum Value	0
Maximum Value	2147483647
Unit	transactions

The value of this system variable can be changed while Group Replication is running, and the change takes effect immediately.

group_replication_flow_control_applier_threshold specifies the number of waiting transactions in the applier queue that trigger flow control.

- group_replication_flow_control_certifier_threshold

Command-Line Format	--group-replication-flow-control-certifier-threshold=#
System Variable	group_replication_flow_control_certifier_th
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	25000
Minimum Value	0
Maximum Value	2147483647
Unit	transactions

The value of this system variable can be changed while Group Replication is running, and the change takes effect immediately.

group_replication_flow_control_certifier_threshold specifies the number of waiting transactions in the certifier queue that trigger flow control.

- group_replication_flow_control_hold_percent

Command-Line Format	--group-replication-flow-control-hold-percent=#
System Variable	group_replication_flow_control_hold_percent
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	10
Minimum Value	0
Maximum Value	100

Unit	percentage

The value of this system variable can be changed while Group Replication is running, and the change takes effect immediately.

`group_replication_flow_control_hold_percent` defines what percentage of the group quota remains unused to allow a cluster under flow control to catch up on backlog. A value of 0 implies that no part of the quota is reserved for catching up on the work backlog.

- `group_replication_flow_control_max_quota`

Command-Line Format	`--group-replication-flow-control-max-quota=#`
System Variable	`group_replication_flow_control_max_quota`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`2147483647`

The value of this system variable can be changed while Group Replication is running, and the change takes effect immediately.

`group_replication_flow_control_max_quota` defines the maximum flow control quota of the group, or the maximum available quota for any period while flow control is enabled. A value of 0 implies that there is no maximum quota set. The value of this system variable cannot be smaller than `group_replication_flow_control_min_quota` and `group_replication_flow_control_min_recovery_quota`.

- `group_replication_flow_control_member_quota_percent`

Command-Line Format	`--group-replication-flow-control-member-quota-percent=#`
System Variable	`group_replication_flow_control_member_quota_pe`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`100`
Unit	percentage

The value of this system variable can be changed while Group Replication is running, and the change takes effect immediately.

`group_replication_flow_control_member_quota_percent` defines the percentage of the quota that a member should assume is available for itself when calculating the quotas. A value of 0 implies that the quota should be split equally between members that were writers in the last period.

- `group_replication_flow_control_min_quota`

Command-Line Format	`--group-replication-flow-control-min-quota=#`
System Variable	`group_replication_flow_control_min_quota`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`2147483647`

The value of this system variable can be changed while Group Replication is running, and the change takes effect immediately.

`group_replication_flow_control_min_quota` controls the lowest flow control quota that can be assigned to a member, independently of the calculated minimum quota executed in the last period. A value of 0 implies that there is no minimum quota. The value of this system variable cannot be larger than `group_replication_flow_control_max_quota`.

- `group_replication_flow_control_min_recovery_quota`

Command-Line Format	`--group-replication-flow-control-min-recovery-quota=#`
System Variable	`group_replication_flow_control_min_recovery`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`2147483647`

The value of this system variable can be changed while Group Replication is running, and the change takes effect immediately.

`group_replication_flow_control_min_recovery_quota` controls the lowest quota that can be assigned to a member because of another recovering member in the group, independently of the calculated minimum quota executed in the last period. A value of 0 implies that there is no minimum quota. The value of this system variable cannot be larger than `group_replication_flow_control_max_quota`.

- `group_replication_flow_control_mode`

Command-Line Format	`--group-replication-flow-control-mode=value`
System Variable	`group_replication_flow_control_mode`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No

Type	Enumeration
Default Value	`QUOTA`

Valid Values	`DISABLED`
	
	`QUOTA`

The value of this system variable can be changed while Group Replication is running, and the change takes effect immediately.

`group_replication_flow_control_mode` specifies the mode used for flow control.

- `group_replication_flow_control_period`

Command-Line Format	`--group-replication-flow-control-period=#`
System Variable	`group_replication_flow_control_period`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`1`
Minimum Value	`1`
Maximum Value	`60`
Unit	seconds

The value of this system variable can be changed while Group Replication is running, and the change takes effect immediately.

`group_replication_flow_control_period` defines how many seconds to wait between flow control iterations, in which flow control messages are sent and flow control management tasks are run.

- `group_replication_flow_control_release_percent`

Command-Line Format	`--group-replication-flow-control-release-percent=#`
System Variable	`group_replication_flow_control_release_percent`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`50`
Minimum Value	`0`
Maximum Value	`1000`
Unit	percentage

The value of this system variable can be changed while Group Replication is running, and the change takes effect immediately.

`group_replication_flow_control_release_percent` defines how the group quota should be released when flow control no longer needs to throttle the writer members, with this

percentage being the quota increase per flow control period. A value of 0 implies that once the flow control thresholds are within limits the quota is released in a single flow control iteration. The range allows the quota to be released at up to 10 times current quota, as that allows a greater degree of adaptation, mainly when the flow control period is large and the quotas are very small.

- `group_replication_force_members`

Command-Line Format	`--group-replication-force-members=value`
System Variable	`group_replication_force_members`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String

This system variable is used to force a new group membership. The value of this system variable can be changed while Group Replication is running, and the change takes effect immediately. You only need to set the value of the system variable on one of the group members that is to remain in the group. For details of the situation in which you might need to force a new group membership, and a procedure to follow when using this system variable, see Section 18.7.8, “Handling a Network Partition and Loss of Quorum”.

`group_replication_force_members` specifies a list of peer addresses as a comma separated list, such as `host1:port1,host2:port2`. Any existing members that are not included in the list do not receive a new view of the group and are blocked. For each existing member that is to continue as a member, you must include the IP address or host name and the port, as they are given in the `group_replication_local_address` system variable for each member. An IPv6 address must be specified in square brackets. For example:

```
"198.51.100.44:33061,[2001:db8:85a3:8d3:1319:8a2e:370:7348]:33061,example.org:33061"
```

The group communication engine for Group Replication (XCom) checks that the supplied IP addresses are in a valid format, and checks that you have not included any group members that are currently unreachable. Otherwise, the new configuration is not validated, so you must be careful to include only online servers that are reachable members of the group. Any incorrect values or invalid host names in the list could cause the group to be blocked with an invalid configuration.

It is important before forcing a new membership configuration to ensure that the servers to be excluded have been shut down. If they are not, shut them down before proceeding. Group members that are still online can automatically form new configurations, and if this has already taken place, forcing a further new configuration could create an artificial split-brain situation for the group.

After you have used the `group_replication_force_members` system variable to successfully force a new group membership and unblock the group, ensure that you clear the system variable. `group_replication_force_members` must be empty in order to issue a `START GROUP_REPLICATION` statement.

- `group_replication_group_name`

Command-Line Format	`--group-replication-group-name=value`
System Variable	`group_replication_group_name`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No

Type	String

The value of this system variable cannot be changed while Group Replication is running.

`group_replication_group_name` specifies the name of the group which this server instance belongs to, which must be a valid UUID. This UUID forms part of the GTIDs that are used when transactions received by group members from clients, and view change events that are generated internally by the group members, are written to the binary log.

Important

A unique UUID must be used.

- `group_replication_group_seeds`

Command-Line Format	`--group-replication-group-seeds=value`
System Variable	`group_replication_group_seeds`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String

The value of this system variable can be changed while Group Replication is running, but the change only takes effect after you stop and restart Group Replication on the group member.

`group_replication_group_seeds` is a list of group members to which a joining member can connect to obtain details of all the current group members. The joining member uses these details to select and connect to a group member to obtain the data needed for synchrony with the group. The list consists of a single internal network address or host name for each included seed member, as configured in the seed member's `group_replication_local_address` system variable (not the seed member's SQL client connection, as specified by MySQL Server's `hostname` and `port` system variables). The addresses of the seed members are specified as a comma separated list, such as `host1:port1`,`host2:port2`. An IPv6 address must be specified in square brackets. For example:

```
group_replication_group_seeds= "198.51.100.44:33061,[2001:db8:85a3:8d3:1319:8a2e:370:7348]:33061, example
```

Note that the value you specify for this variable is not validated until a `START GROUP_REPLICATION` statement is issued and the Group Communication System (GCS) is available.

Usually this list consists of all members of the group, but you can choose a subset of the group members to be seeds. The list must contain at least one valid member address. Each address is validated when starting Group Replication. If the list does not contain any valid member addresses, issuing `START GROUP_REPLICATION` fails.

When a server is joining a replication group, it attempts to connect to the first seed member listed in its `group_replication_group_seeds` system variable. If the connection is refused, the joining member tries to connect to each of the other seed members in the list in order. If the joining member connects to a seed member but does not get added to the replication group as a result (for example, because the seed member does not have the joining member's address in its allowlist and closes the connection), the joining member continues to try the remaining seed members in the list in order.

A joining member must communicate with the seed member using the same protocol (IPv4 or IPv6) that the seed member advertises in the `group_replication_group_seeds` option. For the purpose of IP address permissions for Group Replication, the allowlist on the seed member must include an IP address for the joining member for the protocol offered by the seed member, or a host name that resolves to an address for that protocol. This address or host name must be set up

and permitted in addition to the joining member's `group_replication_local_address` if the protocol for that address does not match the seed member's advertised protocol. If a joining member does not have a permitted address for the appropriate protocol, its connection attempt is refused. For more information, see Section 18.6.4, “Group Replication IP Address Permissions”.

- `group_replication_gtid_assignment_block_size`

Command-Line Format	`--group-replication-gtid-assignment-block-size=#`
System Variable	`group_replication_gtid_assignment_block_siz`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`1000000`
Minimum Value	`1`
Maximum Value (64-bit platforms)	`9223372036854775807`
Maximum Value (32-bit platforms)	`4294967295`

Note

This system variable is a group-wide configuration setting, and a full reboot of the replication group is required for a change to take effect.

`group_replication_gtid_assignment_block_size` specifies the number of consecutive GTIDs that are reserved for each group member. Each member consumes its own blocks and reserves more when needed.

This system variable is a group-wide configuration setting. It must have the same value on all group members, cannot be changed while Group Replication is running, and requires a full reboot of the group (a bootstrap by a server with `group_replication_bootstrap_group=ON`) in order for the value change to take effect. For instructions to safely bootstrap a group where transactions have been executed and certified, see Section 18.5.2, “Restarting a Group”.

If the group has a value set for this system variable, and a joining member has a different value set for the system variable, the joining member cannot join the group until the value is changed to match. If the group members have a value set for this system variable, and the joining member does not support the system variable, it cannot join the group.

- `group_replication_ip_allowlist`

Command-Line Format	`--group-replication-ip-allowlist=value`
Introduced	8.0.22
System Variable	`group_replication_ip_allowlist`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String
Default Value	`AUTOMATIC`

`group_replication_ip_allowlist` is available from MySQL 8.0.22 to replace `group_replication_ip_whitelist`. From MySQL 8.0.24, the value of this system variable

can be changed while Group Replication is running, and the change takes effect immediately on the member.

`group_replication_ip_allowlist` specifies which hosts are permitted to connect to the group. When the XCom communication stack is in use for the group (`group_replication_communication_stack=XCOM`), the allowlist is used to control access to the group. When the MySQL communication stack is in use for the group (`group_replication_communication_stack=MYSQL`), user authentication is used to control access to the group, and the allowlist is not used and is ignored if set.

The address that you specify for each group member in `group_replication_local_address` must be permitted on the other servers in the replication group. Note that the value you specify for this variable is not validated until a `START GROUP_REPLICATION` statement is issued and the Group Communication System (GCS) is available.

By default, this system variable is set to `AUTOMATIC`, which permits connections from private subnetworks active on the host. The group communication engine for Group Replication (XCom) automatically scans active interfaces on the host, and identifies those with addresses on private subnetworks. These addresses and the `localhost` IP address for IPv4 and (from MySQL 8.0.14) IPv6 are used to create the Group Replication allowlist. For a list of the ranges from which addresses are automatically permitted, see Section 18.6.4, “Group Replication IP Address Permissions”.

The automatic allowlist of private addresses cannot be used for connections from servers outside the private network. For Group Replication connections between server instances that are on different machines, you must provide public IP addresses and specify these as an explicit allowlist. If you specify any entries for the allowlist, the private addresses are not added automatically, so if you use any of these, you must specify them explicitly. The `localhost` IP addresses are added automatically.

As the value of the `group_replication_ip_allowlist` option, you can specify any combination of the following:

- IPv4 addresses (for example, `198.51.100.44`)
- IPv4 addresses with CIDR notation (for example, `192.0.2.21/24`)
- IPv6 addresses, from MySQL 8.0.14 (for example, `2001:db8:85a3:8d3:1319:8a2e:370:7348`)
- IPv6 addresses with CIDR notation, from MySQL 8.0.14 (for example, `2001:db8:85a3:8d3::/64`)
- Host names (for example, `example.org`)
- Host names with CIDR notation (for example, `www.example.com/24`)

Before MySQL 8.0.14, host names could only resolve to IPv4 addresses. From MySQL 8.0.14, host names can resolve to IPv4 addresses, IPv6 addresses, or both. If a host name resolves to both an IPv4 and an IPv6 address, the IPv4 address is always used for Group Replication connections. You can use CIDR notation in combination with host names or IP addresses to permit a block of IP addresses with a particular network prefix, but do ensure that all the IP addresses in the specified subnet are under your control.

A comma must separate each entry in the allowlist. For example:

```
"192.0.2.21/24,198.51.100.44,203.0.113.0/24,2001:db8:85a3:8d3:1319:8a2e:370:7348,example.org,www.example
```

If any of the seed members for the group are listed in the `group_replication_group_seeds` option with an IPv6 address when a joining member has an IPv4 `group_replication_local_address`, or the reverse, you must also set up and permit an

alternative address for the joining member for the protocol offered by the seed member (or a host name that resolves to an address for that protocol). For more information, see Section 18.6.4, “Group Replication IP Address Permissions”.

It is possible to configure different allowlists on different group members according to your security requirements, for example, in order to keep different subnets separate. However, this can cause issues when a group is reconfigured. If you do not have a specific security requirement to do otherwise, use the same allowlist on all members of a group. For more details, see Section 18.6.4, “Group Replication IP Address Permissions”.

For host names, name resolution takes place only when a connection request is made by another server. A host name that cannot be resolved is not considered for allowlist validation, and a warning message is written to the error log. Forward-confirmed reverse DNS (FCrDNS) verification is carried out for resolved host names.

Warning

Host names are inherently less secure than IP addresses in an allowlist. FCrDNS verification provides a good level of protection, but can be compromised by certain types of attack. Specify host names in your allowlist only when strictly necessary, and ensure that all components used for name resolution, such as DNS servers, are maintained under your control. You can also implement name resolution locally using the hosts file, to avoid the use of external components.

- `group_replication_ip_whitelist`

Command-Line Format	`--group-replication-ip-whitelist=value`
Deprecated	8.0.22
System Variable	`group_replication_ip_whitelist`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String
Default Value	`AUTOMATIC`

From MySQL 8.0.22, `group_replication_ip_whitelist` is deprecated, and `group_replication_ip_allowlist` is available to replace it. For both system variables, the default value is `AUTOMATIC`.

At Group Replication startup, if either one of the system variables has been set to a user-defined value and the other has not, the changed value is used. If both of the system variables have been set to a user-defined value, the value of `group_replication_ip_allowlist` is used.

If you change the value of `group_replication_ip_whitelist` or `group_replication_ip_allowlist` while Group Replication is running, which is possible from MySQL 8.0.24, neither variable has precedence over the other.

The new system variable works in the same way as the old system variable, only the terminology has changed. The behavior description given for `group_replication_ip_allowlist` applies to both the old and new system variables.

- `group_replication_local_address`

Command-Line Format	`--group-replication-local-address=value`

System Variable	group_replication_local_address
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

The value of this system variable can be changed while Group Replication is running, but the change only takes effect after you stop and restart Group Replication on the group member.

group_replication_local_address sets the network address which the member provides for connections from other members, specified as a host:port formatted string. This address must be reachable by all members of the group because it is used by the group communication engine for Group Replication (XCom, a Paxos variant) for TCP communication between remote XCom instances. If you are using the MySQL communication stack to establish group communication connections between members (group_replication_communication_stack = MYSQL), the address must be one of the IP addresses and ports where MySQL Server is listening on, as specified by the bind_address system variable for the server.

Warning

Do not use this address to query or administer the databases on the member. This is not the SQL client connection host and port.

The address or host name that you specify in group_replication_local_address is used by Group Replication as the unique identifier for a group member within the replication group. You can use the same port for all members of a replication group as long as the host names or IP addresses are all different, and you can use the same host name or IP address for all members as long as the ports are all different. The recommended port for group_replication_local_address is 33061. Note that the value you specify for this variable is not validated until the START GROUP_REPLICATION statement is issued and the Group Communication System (GCS) is available.

The network address configured by group_replication_local_address must be resolvable by all group members. For example, if each server instance is on a different machine with a fixed network address, you could use the IP address of the machine, such as 10.0.0.1. If you use a host name, you must use a fully qualified name, and ensure it is resolvable through DNS, correctly configured /etc/hosts files, or other name resolution processes. From MySQL 8.0.14, IPv6 addresses (or host names that resolve to them) can be used as well as IPv4 addresses. An IPv6 address must be specified in square brackets in order to distinguish the port number, for example:

```
group_replication_local_address= "[2001:db8:85a3:8d3:1319:8a2e:370:7348]:33061"
```

If a host name specified as the Group Replication local address for a server instance resolves to both an IPv4 and an IPv6 address, the IPv4 address is always used for Group Replication connections. For more information on Group Replication support for IPv6 networks and on replication groups with a mix of members using IPv4 and members using IPv6, see Section 18.5.5, “Support For IPv6 And For Mixed IPv6 And IPv4 Groups”.

If you are using the XCom communication stack to establish group communication connections between members (group_replication_communication_stack = XCOM), the address that you specify for each group member in group_replication_local_address must be added to the list for the group_replication_ip_allowlist (from MySQL 8.0.22) or group_replication_ip_whitelist (for MySQL 8.0.21 and earlier) system variable on the other servers in the replication group. When the XCom communication stack is in use for the group, the allowlist is used to control access to the group. When the MySQL communication stack is in use for the group, user authentication is used to control access to the group, and the allowlist is not used and is ignored if set. Note that if any of the seed members for the group are listed in the

group_replication_group_seeds option with an IPv6 address when this member has an IPv4 group_replication_local_address, or the reverse, you must also set up and permit an alternative address for this member for the required protocol (or a host name that resolves to an address for that protocol). For more information, see Section 18.6.4, “Group Replication IP Address Permissions”.

- group_replication_member_expel_timeout

Command-Line Format	--group-replication-member-expel-timeout=#
Introduced	8.0.13
System Variable	group_replication_member_expel_timeout
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value (≥ 8.0.21)	5
Default Value (≤ 8.0.20)	0
Minimum Value	0
Maximum Value (≥ 8.0.14)	3600
Maximum Value (≤ 8.0.13)	31536000
Unit	seconds

The value of this system variable can be changed while Group Replication is running, and the change takes effect immediately. The current value of the system variable is read whenever Group Replication checks the timeout. It is not mandatory for all members of a group to have the same setting, but it is recommended in order to avoid unexpected expulsions.

group_replication_member_expel_timeout specifies the period of time in seconds that a Group Replication group member waits after creating a suspicion, before expelling from the group the member suspected of having failed. The initial 5-second detection period before a suspicion is created does not count as part of this time. Up to and including MySQL 8.0.20, the value of group_replication_member_expel_timeout defaults to 0, meaning that there is no waiting period and a suspected member is liable for expulsion immediately after the 5-second detection period ends. From MySQL 8.0.21, the value defaults to 5, meaning that a suspected member is liable for expulsion 5 seconds after the 5-second detection period.

Changing the value of group_replication_member_expel_timeout on a group member takes effect immediately for existing as well as future suspicions on that group member. You can therefore use this as a method to force a suspicion to time out and expel a suspected member, allowing changes to the group configuration. For more information, see Section 18.7.7.1, “Expel Timeout”.

Increasing the value of group_replication_member_expel_timeout can help to avoid unnecessary expulsions on slower or less stable networks, or in the case of expected transient network outages or machine slowdowns. If a suspect member becomes active again before the suspicion times out, it applies all the messages that were buffered by the remaining group members and enters ONLINE state, without operator intervention. You can specify a timeout value up to a maximum of 3600 seconds (1 hour). It is important to ensure that XCom's message cache is sufficiently large to contain the expected volume of messages in your specified time period, plus the initial 5-second detection period, otherwise members are unable to reconnect. You can adjust

the cache size limit using the `group_replication_message_cache_size` system variable. For more information, see Section 18.7.6, “XCom Cache Management”.

If the timeout is exceeded, the suspect member is liable for expulsion immediately after the suspicion times out. If the member is able to resume communications and receives a view where it is expelled, and the member has the `group_replication_autorejoin_tries` system variable set to specify a number of auto-rejoin attempts, it proceeds to make the specified number of attempts to rejoin the group while in super read only mode. If the member does not have any auto-rejoin attempts specified, or if it has exhausted the specified number of attempts, it follows the action specified by the system variable `group_replication_exit_state_action`.

For more information on using the `group_replication_member_expel_timeout` setting, see Section 18.7.7.1, “Expel Timeout”. For alternative mitigation strategies to avoid unnecessary expulsions where this system variable is not available, see Section 18.3.2, “Group Replication Limitations”.

- `group_replication_member_weight`

Command-Line Format	`--group-replication-member-weight=#`
System Variable	`group_replication_member_weight`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`50`
Minimum Value	`0`
Maximum Value	`100`
Unit	percentage

The value of this system variable can be changed while Group Replication is running, and the change takes effect immediately. The system variable's current value is read when a failover situation occurs.

`group_replication_member_weight` specifies a percentage weight that can be assigned to members to influence the chance of the member being elected as primary in the event of failover, for example when the existing primary leaves a single-primary group. Assign numeric weights to members to ensure that specific members are elected, for example during scheduled maintenance of the primary or to ensure certain hardware is prioritized in the event of failover.

For a group with members configured as follows:

- `member-1`: group_replication_member_weight=30, server_uuid=aaaa
- `member-2`: group_replication_member_weight=40, server_uuid=bbbb
- `member-3`: group_replication_member_weight=40, server_uuid=cccc
- `member-4`: group_replication_member_weight=40, server_uuid=dddd

during election of a new primary the members above would be sorted as `member-2`, `member-3`, `member-4`, and `member-1`. This results in `member`-2 being chosen as the new primary in the event of failover. For more information, see Section 18.1.3.1, “Single-Primary Mode”.

- `group_replication_message_cache_size`

Command-Line Format	`--group-replication-message-cache-size=#`
Introduced	8.0.16
System Variable	`group_replication_message_cache_size`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`1073741824 (1 GB)`
Minimum Value (64-bit platforms, ≥ 8.0.21)	`134217728 (128 MB)`
Minimum Value (64-bit platforms, ≤ 8.0.20)	`1073741824 (1 GB)`
Minimum Value (32-bit platforms, ≥ 8.0.21)	`134217728 (128 MB)`
Minimum Value (32-bit platforms, ≤ 8.0.20)	`1073741824 (1 GB)`
Maximum Value (64-bit platforms)	`18446744073709551615 (16 EiB)`
Maximum Value (32-bit platforms)	`315360004294967295 (4 GB)`
Unit	bytes

This system variable should have the same value on all group members. The value of this system variable can be changed while Group Replication is running. The change takes effect on each group member after you stop and restart Group Replication on the member. During this process, the value of the system variable is permitted to differ between group members, but members might be unable to reconnect in the event of a disconnection.

`group_replication_message_cache_size` sets the maximum amount of memory that is available for the message cache in the group communication engine for Group Replication (XCom). The XCom message cache holds messages (and their metadata) that are exchanged between the group members as a part of the consensus protocol. Among other functions, the message cache is used for recovery of missed messages by members that reconnect with the group after a period where they were unable to communicate with the other group members.

The `group_replication_member_expel_timeout` system variable determines the waiting period (up to an hour) that is allowed in addition to the initial 5-second detection period for members to return to the group rather than being expelled. The size of the XCom message cache should be set with reference to the expected volume of messages in this time period, so that it contains all the missed messages required for members to return successfully. Up to MySQL 8.0.20, the default is only the 5-second detection period, but from MySQL 8.0.21, the default is a 5-second waiting period after the 5-second detection period, for a total time period of 10 seconds.

Ensure that sufficient memory is available on your system for your chosen cache size limit, considering the size of MySQL Server's other caches and object pools. The default setting is 1073741824 bytes (1 GB). The minimum setting is also 1 GB up to MySQL 8.0.20. From MySQL 8.0.21, the minimum setting is 134217728 bytes (128 MB), which enables deployment on a host that has a restricted amount of available memory, and good network connectivity to minimize the frequency and duration of transient losses of connectivity for group members. Note that the limit set using `group_replication_message_cache_size` applies only to the data stored in the cache, and the cache structures require an additional 50 MB of memory.

The cache size limit can be increased or reduced dynamically at runtime. If you reduce the cache size limit, XCom removes the oldest entries that have been decided and delivered until the current size is below the limit. Group Replication's Group Communication System (GCS) alerts you, by a warning message, when a message that is likely to be needed for recovery by a member that is

currently unreachable is removed from the message cache. For more information on tuning the message cache size, see Section 18.7.6, “XCom Cache Management”.

- `group_replication_paxos_single_leader`

Command-Line Format	`--group-replication-paxos-single-leader[={OFF|ON}]`
Introduced	8.0.27
System Variable	`group_replication_paxos_single_leader`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Boolean
Default Value	`OFF`

Note

This system variable is a group-wide configuration setting, and a full reboot of the replication group is required for a change to take effect.

`group_replication_paxos_single_leader` is available from MySQL 8.0.27. It enables the group communication engine to operate with a single consensus leader when the group is in single-primary mode. With the default setting `OFF`, this behavior is disabled, and every member of the group is used as a leader, which is the behavior in releases before this system variable was available. When the system variable is set to `ON`, the group communication engine can use a single leader to drive consensus. Operating with a single consensus leader improves performance and resilience in single-primary mode, particularly when some of the group’s secondary members are currently unreachable. For more information, see Section 18.7.3, “Single Consensus Leader”.

In order for the group communication engine to use a single consensus leader, the group's communication protocol version must be MySQL 8.0.27 or above. Use the `group_replication_get_communication_protocol()` function to view the group's communication protocol version. If a lower version is in use, the group cannot use this behavior. You can use the `group_replication_set_communication_protocol()` function to set the group's communication protocol to a higher version if all group members support it. For more information, see Section 18.5.1.4, “Setting a Group's Communication Protocol Version”.

This system variable is a group-wide configuration setting. It must have the same value on all group members, cannot be changed while Group Replication is running, and requires a full reboot of the group (a bootstrap by a server with `group_replication_bootstrap_group=ON`) in order for the value change to take effect. For instructions to safely bootstrap a group where transactions have been executed and certified, see Section 18.5.2, “Restarting a Group”.

If the group has a value set for this system variable, and a joining member has a different value set for the system variable, the joining member cannot join the group until the value is changed to match. If the group members have a value set for this system variable, and the joining member does not support the system variable, it cannot join the group.

The field `WRITE_CONSENSUS_SINGLE_LEADER_CAPABLE` in the Performance Schema table `replication_group_communication_information` shows whether the group supports the use of a single leader, even if `group_replication_paxos_single_leader` is currently set to `OFF` on the queried member. The field is set to 1 if the group was started with `group_replication_paxos_single_leader` set to `ON`, and its communication protocol version is MySQL 8.0.27 or above.

- `group_replication_poll_spin_loops`

Command-Line Format	`--group-replication-poll-spin-loops=#`
System Variable	`group_replication_poll_spin_loops`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

The value of this system variable can be changed while Group Replication is running, but the change only takes effect after you stop and restart Group Replication on the group member.

`group_replication_poll_spin_loops` specifies the number of times the group communication thread waits for the communication engine mutex to be released before the thread waits for more incoming network messages.

- `group_replication_recovery_complete_at`

Command-Line Format	`--group-replication-recovery-complete-at=value`
Deprecated	8.0.34
System Variable	`group_replication_recovery_complete_at`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Enumeration
Default Value	`TRANSACTIONS_APPLIED`
Valid Values	`TRANSACTIONS_CERTIFIED` `TRANSACTIONS_APPLIED`

The value of this system variable can be changed while Group Replication is running, but the change only takes effect after you stop and restart Group Replication on the group member.

`group_replication_recovery_complete_at` specifies the policy applied during the distributed recovery process when handling cached transactions after state transfer from an existing member. You can choose whether a member is marked online after it has received and certified all transactions that it missed before it joined the group (`TRANSACTIONS_CERTIFIED`), or only after it has received, certified, and applied them (`TRANSACTIONS_APPLIED`).

This variable is deprecated as of MySQL 8.0.34 (as is `TRANSACTIONS_CERTIFIED`). Expect its removal in a future release of MySQL.

- `group_replication_recovery_compression_algorithms`

Command-Line Format	`--group-replication-recovery-compression-algorithms=value`

Introduced	8.0.18
System Variable	group_replication_recovery_compression_algorit
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Set
Default Value	uncompressed
Valid Values	zlib zstd uncompressed

The value of this system variable can be changed while Group Replication is running, but the change only takes effect after you stop and restart Group Replication on the group member.

group_replication_recovery_compression_algorithms specifies the compression algorithms permitted for Group Replication distributed recovery connections for state transfer from a donor's binary log. The available algorithms are the same as for the protocol_compression_algorithms system variable. For more information, see Section 4.2.8, "Connection Compression Control".

This setting does not apply if the server has been set up to support cloning (see Section 18.5.4.2, "Cloning for Distributed Recovery") and a remote cloning operation is used during distributed recovery. For this method of state transfer, the clone plugin's clone_enable_compression setting applies.

- group_replication_recovery_get_public_key

Command-Line Format	--group-replication-recovery-get-public-key[={OFF|ON}]
System Variable	group_replication_recovery_get_public_key
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

The value of this system variable can be changed while Group Replication is running, but the change only takes effect after you stop and restart Group Replication on the group member.

group_replication_recovery_get_public_key specifies whether to request from the source the public key required for RSA key pair-based password exchange. If group_replication_recovery_public_key_path is set to a valid public key file, it takes precedence over group_replication_recovery_get_public_key. This variable applies if you are not using SSL for distributed recovery over the group_replication_recovery channel (group_replication_recovery_use_ssl=ON), and the replication user account for Group Replication authenticates with the caching_sha2_password plugin (which is the default in MySQL 8.0). For more details, see Replication User With The Caching SHA-2 Authentication Plugin.

- group_replication_recovery_public_key_path

Command-Line Format	--group-replication-recovery-public-key-path=file_name

System Variable	group_replication_recovery_public_key_path
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	File name
Default Value	NULL

The value of this system variable can be changed while Group Replication is running, but the change only takes effect after you stop and restart Group Replication on the group member.

group_replication_recovery_public_key_path specifies the path name to a file containing a replica-side copy of the public key required by the source for RSA key pair-based password exchange. The file must be in PEM format. If group_replication_recovery_public_key_path is set to a valid public key file, it takes precedence over group_replication_recovery_get_public_key. This variable applies if you are not using SSL for distributed recovery over the group_replication_recovery channel (so group_replication_recovery_use_ssl is set to OFF), and the replication user account for Group Replication authenticates with the caching_sha2_password plugin (which is the default in MySQL 8.0) or the sha256_password plugin. (For sha256_password, setting group_replication_recovery_public_key_path applies only if MySQL was built using OpenSSL.) For more details, see Replication User With The Caching SHA-2 Authentication Plugin.

- group_replication_recovery_reconnect_interval

Command-Line Format	--group-replication-recovery-reconnect-interval=#
System Variable	group_replication_recovery_reconnect_interv
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	60
Minimum Value	0
Maximum Value	31536000
Unit	seconds

The value of this system variable can be changed while Group Replication is running, but the change only takes effect after you stop and restart Group Replication on the group member.

group_replication_recovery_reconnect_interval specifies the sleep time, in seconds, between reconnection attempts when no suitable donor was found in the group for distributed recovery.

- group_replication_recovery_retry_count

Command-Line Format	--group-replication-recovery-retry-count=#
System Variable	group_replication_recovery_retry_count
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No

Type	Integer
Default Value	`10`
Minimum Value	`0`
Maximum Value	`31536000`

The value of this system variable can be changed while Group Replication is running, but the change only takes effect after you stop and restart Group Replication on the group member.

`group_replication_recovery_retry_count` specifies the number of times that the member that is joining tries to connect to the available donors for distributed recovery before giving up.

- `group_replication_recovery_ssl_ca`

Command-Line Format	`--group-replication-recovery-ssl-ca=value`
System Variable	`group_replication_recovery_ssl_ca`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String

The value of this system variable can be changed while Group Replication is running, but the change only takes effect after you stop and restart Group Replication on the group member.

`group_replication_recovery_ssl_ca` specifies the path to a file that contains a list of trusted SSL certificate authorities for distributed recovery connections. See Section 18.6.2, “Securing Group Communication Connections with Secure Socket Layer (SSL)” for information on configuring SSL for distributed recovery.

If this server has been set up to support cloning (see Section 18.5.4.2, “Cloning for Distributed Recovery”), and you have set `group_replication_recovery_use_ssl` to `ON`, Group Replication automatically configures the setting for the clone SSL option `clone_ssl_ca` to match your setting for `group_replication_recovery_ssl_ca`.

When the MySQL communication stack is in use for the group (`group_replication_communication_stack = MYSQL`), this setting is used for the TLS/SSL configuration for group communication connections, as well as for distributed recovery connections.

- `group_replication_recovery_ssl_capath`

Command-Line Format	`--group-replication-recovery-ssl-capath=value`
System Variable	`group_replication_recovery_ssl_capath`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String

The value of this system variable can be changed while Group Replication is running, but the change only takes effect after you stop and restart Group Replication on the group member.

`group_replication_recovery_ssl_capath` specifies the path to a directory that contains trusted SSL certificate authority certificates for distributed recovery connections. See Section 18.6.2,

"Securing Group Communication Connections with Secure Socket Layer (SSL)" for information on configuring SSL for distributed recovery.

When the MySQL communication stack is in use for the group (`group_replication_communication_stack = MYSQL`), this setting is used for the TLS/SSL configuration for group communication connections, as well as for distributed recovery connections.

- `group_replication_recovery_ssl_cert`

Command-Line Format	`--group-replication-recovery-ssl-cert=value`
System Variable	`group_replication_recovery_ssl_cert`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String

The value of this system variable can be changed while Group Replication is running, but the change only takes effect after you stop and restart Group Replication on the group member.

`group_replication_recovery_ssl_cert` specifies the name of the SSL certificate file to use for establishing a secure connection for distributed recovery. See Section 18.6.2, "Securing Group Communication Connections with Secure Socket Layer (SSL)" for information on configuring SSL for distributed recovery.

If this server has been set up to support cloning (see Section 18.5.4.2, "Cloning for Distributed Recovery"), and you have set `group_replication_recovery_use_ssl` to `ON`, Group Replication automatically configures the setting for the clone SSL option `clone_ssl_cert` to match your setting for `group_replication_recovery_ssl_cert`.

When the MySQL communication stack is in use for the group (`group_replication_communication_stack = MYSQL`), this setting is used for the TLS/SSL configuration for group communication connections, as well as for distributed recovery connections.

- `group_replication_recovery_ssl_cipher`

Command-Line Format	`--group-replication-recovery-ssl-cipher=value`
System Variable	`group_replication_recovery_ssl_cipher`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String

The value of this system variable can be changed while Group Replication is running, but the change only takes effect after you stop and restart Group Replication on the group member.

`group_replication_recovery_ssl_cipher` specifies the list of permissible ciphers for SSL encryption. See Section 18.6.2, "Securing Group Communication Connections with Secure Socket Layer (SSL)" for information on configuring SSL for distributed recovery.

When the MySQL communication stack is in use for the group (`group_replication_communication_stack = MYSQL`), this setting is used for the TLS/SSL configuration for group communication connections, as well as for distributed recovery connections.

- `group_replication_recovery_ssl_crl`

Command-Line Format	`--group-replication-recovery-ssl-crl=value`
System Variable	`group_replication_recovery_ssl_crl`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	File name

The value of this system variable can be changed while Group Replication is running, but the change only takes effect after you stop and restart Group Replication on the group member.

`group_replication_recovery_ssl_crl` specifies the path to a directory that contains files containing certificate revocation lists. See Section 18.6.2, “Securing Group Communication Connections with Secure Socket Layer (SSL)” for information on configuring SSL for distributed recovery.

When the MySQL communication stack is in use for the group (`group_replication_communication_stack = MYSQL`), this setting is used for the TLS/SSL configuration for group communication connections, as well as for distributed recovery connections.

- `group_replication_recovery_ssl_crlpath`

Command-Line Format	`--group-replication-recovery-ssl-crlpath=value`
System Variable	`group_replication_recovery_ssl_crlpath`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Directory name

The value of this system variable can be changed while Group Replication is running, but the change only takes effect after you stop and restart Group Replication on the group member.

`group_replication_recovery_ssl_crlpath` specifies the path to a directory that contains files containing certificate revocation lists. See Section 18.6.2, “Securing Group Communication Connections with Secure Socket Layer (SSL)” for information on configuring SSL for distributed recovery.

When the MySQL communication stack is in use for the group (`group_replication_communication_stack = MYSQL`), this setting is used for the TLS/SSL configuration for group communication connections, as well as for distributed recovery connections.

- `group_replication_recovery_ssl_key`

Command-Line Format	`--group-replication-recovery-ssl-key=value`
System Variable	`group_replication_recovery_ssl_key`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No

Type	String

The value of this system variable can be changed while Group Replication is running, but the change only takes effect after you stop and restart Group Replication on the group member.

`group_replication_recovery_ssl_key` specifies the name of the SSL key file to use for establishing a secure connection. See Section 18.6.2, “Securing Group Communication Connections with Secure Socket Layer (SSL)” for information on configuring SSL for distributed recovery.

If this server has been set up to support cloning (see Section 18.5.4.2, “Cloning for Distributed Recovery”), and you have set `group_replication_recovery_use_ssl` to `ON`, Group Replication automatically configures the setting for the clone SSL option `clone_ssl_key` to match your setting for `group_replication_recovery_ssl_key`.

When the MySQL communication stack is in use for the group (`group_replication_communication_stack = MYSQL`), this setting is used for the TLS/SSL configuration for group communication connections, as well as for distributed recovery connections.

- `group_replication_recovery_ssl_verify_server_cert`

Command-Line Format	`--group-replication-recovery-ssl-verify-server-cert[={OFF|ON}]`
System Variable	`group_replication_recovery_ssl_verify_serve`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Boolean
Default Value	`OFF`

The value of this system variable can be changed while Group Replication is running, but the change only takes effect after you stop and restart Group Replication on the group member.

`group_replication_recovery_ssl_verify_server_cert` specifies whether the distributed recovery connection should check the server's Common Name value in the certificate sent by the donor. See Section 18.6.2, “Securing Group Communication Connections with Secure Socket Layer (SSL)” for information on configuring SSL for distributed recovery.

When the MySQL communication stack is in use for the group (`group_replication_communication_stack = MYSQL`), this setting is used for the TLS/SSL configuration for group communication connections, as well as for distributed recovery connections.

- `group_replication_recovery_tls_ciphersuites`

Command-Line Format	`--group-replication-recovery-tls-ciphersuites=value`
Introduced	8.0.19
System Variable	`group_replication_recovery_tls_ciphersuites`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String

Default Value	NULL

The value of this system variable can be changed while Group Replication is running, but the change only takes effect after you stop and restart Group Replication on the group member.

group_replication_recovery_tls_ciphersuites specifies a colon-separated list of one or more permitted ciphersuites when TLSv1.3 is used for connection encryption for the distributed recovery connection, and this server instance is the client in the distributed recovery connection, that is, the joining member. If this system variable is set to NULL when TLSv1.3 is used (which is the default if you do not set the system variable), the ciphersuites that are enabled by default are allowed, as listed in Section 6.3.2, "Encrypted Connection TLS Protocols and Ciphers". If this system variable is set to the empty string, no cipher suites are allowed, and TLSv1.3 is therefore not used. This system variable is available beginning with MySQL 8.0.19. See Section 18.6.2, "Securing Group Communication Connections with Secure Socket Layer (SSL)", for information on configuring SSL for distributed recovery.

When the MySQL communication stack is in use for the group (group_replication_communication_stack = MYSQL), this setting is used for the TLS/SSL configuration for group communication connections, as well as for distributed recovery connections.

- group_replication_recovery_tls_version

Command-Line Format	--group-replication-recovery-tls-version=value
Introduced	8.0.19
System Variable	group_replication_recovery_tls_version
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value (≥ 8.0.28)	TLSv1.2,TLSv1.3
Default Value (≥ 8.0.19, ≤ 8.0.27)	TLSv1,TLSv1.1,TLSv1.2,TLSv1.3

The value of this system variable can be changed while Group Replication is running, but the change only takes effect after you stop and restart Group Replication on the group member.

group_replication_recovery_tls_version specifies a comma-separated list of one or more permitted TLS protocols for connection encryption when this server instance is the client in the distributed recovery connection, that is, the joining member. The group members involved in each distributed recovery connection as the client (joining member) and server (donor) negotiate the highest protocol version that they are both set up to support. This system variable is available from MySQL 8.0.19.

When the MySQL communication stack is in use for the group (group_replication_communication_stack = MYSQL), this setting is used for the TLS/SSL configuration for group communication connections, as well as for distributed recovery connections.

If this system variable is not set, the default "TLSv1,TLSv1.1,TLSv1.2,TLSv1.3" is used up to and including MySQL 8.0.27, and from MySQL 8.0.28, the default "TLSv1.2,TLSv1.3" is used. Ensure the specified protocol versions are contiguous, with no versions numbers skipped from the middle of the sequence.

Important

- Support for the TLSv1 and TLSv1.1 connection protocols is removed from MySQL Server as of MySQL 8.0.28. The protocols were deprecated from

MySQL 8.0.26, though MySQL Server clients, including Group Replication server instances acting as a client, do not return warnings to the user if a deprecated TLS protocol version is used. See Removal of Support for the TLSv1 and TLSv1.1 Protocols for more information.

- Support for the TLSv1.3 protocol is available in MySQL Server as of MySQL 8.0.16, provided that MySQL Server was compiled using OpenSSL 1.1.1. The server checks the version of OpenSSL at startup, and if it is lower than 1.1.1, TLSv1.3 is removed from the default value for the system variable. In that case, the defaults are "`TLSv1,TLSv1.1,TLSv1.2`" up to and including MySQL 8.0.27, and "`TLSv1.2`" from MySQL 8.0.28.
- Group Replication supports TLSv1.3 from MySQL 8.0.18, with support for ciphersuite selection from MySQL 8.0.19. See Section 18.6.2, "Securing Group Communication Connections with Secure Socket Layer (SSL)" for more information.

See Section 18.6.2, "Securing Group Communication Connections with Secure Socket Layer (SSL)" for information on configuring SSL for distributed recovery.

- `group_replication_recovery_use_ssl`

Command-Line Format	`--group-replication-recovery-use-ssl[={OFF|ON}]`
System Variable	`group_replication_recovery_use_ssl`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Boolean
Default Value	`OFF`

The value of this system variable can be changed while Group Replication is running, but the change only takes effect after you stop and restart Group Replication on the group member.

`group_replication_recovery_use_ssl` specifies whether Group Replication distributed recovery connections between group members should use SSL or not. See Section 18.6.2, "Securing Group Communication Connections with Secure Socket Layer (SSL)" for information on configuring SSL for distributed recovery.

If this server has been set up to support cloning (see Section 18.5.4.2, "Cloning for Distributed Recovery"), and you set this option to `ON`, Group Replication uses SSL for remote cloning operations as well as for state transfer from a donor's binary log. If you set this option to `OFF`, Group Replication does not use SSL for remote cloning operations.

- `group_replication_recovery_zstd_compression_level`

Command-Line Format	`--group-replication-recovery-zstd-compression-level=#`
Introduced	8.0.18
System Variable	`group_replication_recovery_zstd_compression`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer

Default Value	3
Minimum Value	1
Maximum Value	22

The value of this system variable can be changed while Group Replication is running, but the change only takes effect after you stop and restart Group Replication on the group member.

`group_replication_recovery_zstd_compression_level` specifies the compression level to use for Group Replication distributed recovery connections that use the `zstd` compression algorithm. The permitted levels are from 1 to 22, with larger values indicating increasing levels of compression. The default `zstd` compression level is 3. For distributed recovery connections that do not use `zstd` compression, this variable has no effect.

For more information, see Section 4.2.8, “Connection Compression Control”.

- `group_replication_single_primary_mode`

Command-Line Format	`--group-replication-single-primary-mode[={OFF|ON}]`
System Variable	`group_replication_single_primary_mode`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Boolean
Default Value	`ON`

Note

This system variable is a group-wide configuration setting, and a full reboot of the replication group is required for a change to take effect.

`group_replication_single_primary_mode` instructs the group to pick a single server automatically to be the one that handles read/write workload. This server is the primary and all others are secondaries.

This system variable is a group-wide configuration setting. It must have the same value on all group members, cannot be changed while Group Replication is running, and requires a full reboot of the group (a bootstrap by a server with `group_replication_bootstrap_group=ON`) in order for the value change to take effect. For instructions to safely bootstrap a group where transactions have been executed and certified, see Section 18.5.2, “Restarting a Group”.

If the group has a value set for this system variable, and a joining member has a different value set for the system variable, the joining member cannot join the group until the value is changed to match. If the group members have a value set for this system variable, and the joining member does not support the system variable, it cannot join the group.

Setting this variable `ON` causes any setting for `group_replication_auto_increment_increment` to be ignored.

In MySQL 8.0.16 and later, you can use the `group_replication_switch_to_single_primary_mode()` and `group_replication_switch_to_multi_primary_mode()` functions to change the value of this system variable while the group is still running. For more information, see Section 18.5.1.2, “Changing a Group's Mode”.

- `group_replication_ssl_mode`

Command-Line Format	`--group-replication-ssl-mode=value`
System Variable	`group_replication_ssl_mode`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Enumeration
Default Value	`DISABLED`

Valid Values	`DISABLED`
	`REQUIRED`
	`VERIFY_CA`
	`VERIFY_IDENTITY`

The value of this system variable can be changed while Group Replication is running, but the change only takes effect after you stop and restart Group Replication on the group member.

`group_replication_ssl_mode` sets the security state of group communication connections between Group Replication members. The possible values are as follows:

DISABLED — Establish an unencrypted connection (the default).

REQUIRED — Establish a secure connection if the server supports secure connections.

VERIFY_CA — Like `REQUIRED`, but additionally verify the server TLS certificate against the configured Certificate Authority (CA) certificates.

VERIFY_IDENTITY — Like `VERIFY_CA`, but additionally verify that the server certificate matches the host to which the connection is attempted.

See Section 18.6.2, “Securing Group Communication Connections with Secure Socket Layer (SSL)” for information on configuring SSL for group communication.

- `group_replication_start_on_boot`

Command-Line Format	`--group-replication-start-on-boot[={OFF|ON}]`
System Variable	`group_replication_start_on_boot`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Boolean

Default Value	ON

The value of this system variable can be changed while Group Replication is running, but the change only takes effect after you stop and restart Group Replication on the group member.

group_replication_start_on_boot specifies whether the server should start Group Replication automatically (ON) or not (OFF) during server start. When you set this option to ON, Group Replication restarts automatically after a remote cloning operation is used for distributed recovery.

To start Group Replication automatically during server start, the user credentials for distributed recovery must be stored in the replication metadata repositories on the server using the CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO statement. If you prefer to specify the user credentials on the START GROUP_REPLICATION statement, which stores the user credentials only in memory, ensure that group_replication_start_on_boot is set to OFF.

- group_replication_tls_source

Command-Line Format	--group-replication-tls-source=value
Introduced	8.0.21
System Variable	group_replication_tls_source
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	mysql_main
Valid Values	mysql_main mysql_admin

The value of this system variable can be changed while Group Replication is running, but the change only takes effect after you stop and restart Group Replication on the group member.

group_replication_tls_source specifies the source of TLS material for Group Replication.

- group_replication_transaction_size_limit

Command-Line Format	--group-replication-transaction-size-limit=#
System Variable	group_replication_transaction_size_limit
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	150000000
Minimum Value	0
Maximum Value	2147483647
Unit	bytes

This system variable should have the same value on all group members. The value of this system variable can be changed while Group Replication is running. The change takes effect immediately on the group member, and applies from the next transaction started on that member. During this

process, the value of the system variable is permitted to differ between group members, but some transactions might be rejected.

`group_replication_transaction_size_limit` configures the maximum transaction size in bytes which the replication group accepts. Transactions larger than this size are rolled back by the receiving member and are not broadcast to the group. Large transactions can cause problems for a replication group in terms of memory allocation, which can cause the system to slow down, or in terms of network bandwidth consumption, which can cause a member to be suspected of having failed because it is busy processing the large transaction.

When this system variable is set to 0 there is no limit to the size of transactions the group accepts. From MySQL 8.0, the default setting for this system variable is 150000000 bytes (approximately 143 MB). Adjust the value of this system variable depending on the maximum message size that you need the group to tolerate, bearing in mind that the time taken to process a transaction is proportional to its size. The value of `group_replication_transaction_size_limit` should be the same on all group members. For further mitigation strategies for large transactions, see Section 18.3.2, “Group Replication Limitations”.

- `group_replication_unreachable_majority_timeout`

Command-Line Format	`--group-replication-unreachable-majority-timeout=#`
System Variable	`group_replication_unreachable_majority_time`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`31536000`
Unit	seconds

The value of this system variable can be changed while Group Replication is running, and the change takes effect immediately. The current value of the system variable is read when an issue occurs that means the behavior is needed.

`group_replication_unreachable_majority_timeout` specifies a number of seconds for which members that suffer a network partition and cannot connect to the majority wait before leaving the group. In a group of 5 servers (S1,S2,S3,S4,S5), if there is a disconnection between (S1,S2) and (S3,S4,S5) there is a network partition. The first group (S1,S2) is now in a minority because it cannot contact more than half of the group. While the majority group (S3,S4,S5) remains running, the minority group waits for the specified time for a network reconnection. For a detailed description of this scenario, see Section 18.7.8, “Handling a Network Partition and Loss of Quorum”.

By default, `group_replication_unreachable_majority_timeout` is set to 0, which means that members that find themselves in a minority due to a network partition wait forever to leave the group. If you set a timeout, when the specified time elapses, all pending transactions processed by the minority are rolled back, and the servers in the minority partition move to the `ERROR` state. If a member has the `group_replication_autorejoin_tries` system variable set to specify a number of auto-rejoin attempts, it proceeds to make the specified number of attempts to rejoin the group while in super read only mode. If the member does not have any auto-rejoin attempts

specified, or if it has exhausted the specified number of attempts, it follows the action specified by the system variable `group_replication_exit_state_action`.

Warning

When you have a symmetric group, with just two members for example (S0,S2), if there is a network partition and there is no majority, after the configured timeout all members enter the `ERROR` state.

For more information on using this option, see Section 18.7.7.2, “Unreachable Majority Timeout”.

- `group_replication_view_change_uuid`

Command-Line Format	`--group-replication-view-change-uuid=value`
Introduced	8.0.26
System Variable	`group_replication_view_change_uuid`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String
Default Value	`AUTOMATIC`

Note

This system variable is a group-wide configuration setting, and a full reboot of the replication group is required for a change to take effect.

`group_replication_view_change_uuid` specifies an alternative UUID to use as the UUID part of the identifier in the GTIDs for view change events generated by the group. The alternative UUID makes these internally generated transactions easy to distinguish from transactions received by the group from clients. This can be useful if your setup allows for failover between groups, and you need to identify and discard transactions that were specific to the backup group. The default value for this system variable is `AUTOMATIC`, meaning that the GTIDs for view change events use the group name specified by the `group_replication_group_name` system variable, as transactions from clients do. Group members at a release that does not have this system variable are treated as having the value `AUTOMATIC`.

The alternative UUID must be different from the group name specified by the `group_replication_group_name` system variable, and it must be different from the server UUID of any group member. It must also be different from any UUIDs used in the GTIDs that are applied to anonymous transactions on replication channels anywhere in this topology, using the `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` option of the `CHANGE REPLICATION SOURCE TO` statement.

This system variable is a group-wide configuration setting. It must have the same value on all group members, cannot be changed while Group Replication is running, and requires a full reboot of the group (a bootstrap by a server with `group_replication_bootstrap_group=ON`) in order for the value change to take effect. For instructions to safely bootstrap a group where transactions have been executed and certified, see Section 18.5.2, “Restarting a Group”.

If the group has a value set for this system variable, and a joining member has a different value set for the system variable, the joining member cannot join the group until the value is changed to match. If the group members have a value set for this system variable, and the joining member does not support the system variable, it cannot join the group.

Group Replication Status Variable

This section describes the status variable which provides information about Group Replication. The variable has the following meaning:

- `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.

  Warning

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

  See Finding the Primary.

18.10 Frequently Asked Questions

This section provides answers to frequently asked questions.

What is the maximum number of MySQL servers in a group?

A group can consist of maximum 9 servers. Attempting to add another server to a group with 9 members causes the request to join to be refused. This limit has been identified from testing and benchmarking as a safe boundary where the group performs reliably on a stable local area network.

How are servers in a group connected?

Servers in a group connect to the other servers in the group by opening a peer-to-peer TCP connection. These connections are only used for internal communication and message passing between servers in the group. This address is configured by the `group_replication_local_address` variable.

What is the group_replication_bootstrap_group option used for?

The bootstrap flag instructs a member to *create* a group and act as the initial seed server. The second member joining the group needs to ask the member that bootstrapped the group to dynamically change the configuration in order for it to be added to the group.

A member needs to bootstrap the group in two scenarios. When the group is originally created, or when shutting down and restarting the entire group.

How do I set credentials for the distributed recovery process?

You can set the user credentials permanently as the credentials for the `group_replication_recovery` channel, using a `CHANGE REPLICATION SOURCE TO` statement (from MySQL 8.0.23) or `CHANGE MASTER TO` statement (before MySQL 8.0.23). Alternatively, from MySQL 8.0.21, you can specify them on the `START GROUP_REPLICATION` statement each time Group Replication is started.

User credentials set using `CHANGE REPLICATION SOURCE TO` | `CHANGE MASTER TO` are stored in plain text in the replication metadata repositories on the server, but user credentials specified on `START GROUP_REPLICATION` are saved in memory only, and are removed by a `STOP GROUP_REPLICATION` statement or server shutdown. Using `START GROUP_REPLICATION` to specify the user credentials therefore helps to secure the Group Replication servers against unauthorized access. However, this method is not compatible with starting Group Replication automatically, as specified by the `group_replication_start_on_boot` system variable. For more information, see Section 18.6.3.1, “Secure User Credentials for Distributed Recovery”.

Can I scale-out my write-load using Group Replication?

Not directly, but MySQL Group replication is a shared nothing full replication solution, where all servers in the group replicate the same amount of data. Therefore if one member in the group writes N bytes to storage as the result of a transaction commit operation, then roughly N bytes are written to storage on other members as well, because the transaction is replicated everywhere.

However, given that other members do not have to do the same amount of processing that the original member had to do when it originally executed the transaction, they apply the changes faster. Transactions are replicated in a format that is used to apply row transformations only, without having to re-execute transactions again (row-based format).

Furthermore, given that changes are propagated and applied in row-based format, this means that they are received in an optimized and compact format, and likely reducing the number of IO operations required when compared to the originating member.

To summarize, you can scale-out processing, by spreading conflict free transactions throughout different members in the group. And you can likely scale-out a small fraction of your IO operations, since remote servers receive only the necessary changes to read-modify-write changes to stable storage.

Does Group Replication require more network bandwidth and CPU, when compared to simple replication and under the same workload?

Some additional load is expected because servers need to be constantly interacting with each other for synchronization purposes. It is difficult to quantify how much more data. It also depends on the size of the group (three servers puts less stress on the bandwidth requirements than nine servers in the group).

Also the memory and CPU footprint are larger, because more complex work is done for the server synchronization part and for the group messaging.

Can I deploy Group Replication across wide-area networks?

Yes, but the network connection between each member *must* be reliable and have suitable performance. Low latency, high bandwidth network connections are a requirement for optimal performance.

If network bandwidth alone is an issue, then Section 18.7.4, “Message Compression” can be used to lower the bandwidth required. However, if the network drops packets, leading to re-transmissions and higher end-to-end latency, throughput and latency are both negatively affected.

Warning

When the network round-trip time (RTT) between any group members is 5 seconds or more you could encounter problems as the built-in failure detection mechanism could be incorrectly triggered.

Do members automatically rejoin a group in case of temporary connectivity problems?

This depends on the reason for the connectivity problem. If the connectivity problem is transient and the reconnection is quick enough that the failure detector is not aware of it, then the server may not be removed from the group. If it is a "long" connectivity problem, then the failure detector eventually suspects a problem and the server is removed from the group.

From MySQL 8.0, two settings are available to increase the chances of a member remaining in or rejoining a group:

- `group_replication_member_expel_timeout` increases the time between the creation of a suspicion (which happens after an initial 5-second detection period) and the expulsion of the member. You can set a waiting period of up to 1 hour. From MySQL 8.0.21, a waiting period of 5 seconds is set by default.
- `group_replication_autorejoin_tries` makes a member try to rejoin the group after an expulsion or unreachable majority timeout. The member makes the specified number of auto-rejoin attempts five minutes apart. From MySQL 8.0.21, this feature is activated by default and the member makes three auto-rejoin attempts.

If a server is expelled from the group and any auto-rejoin attempts do not succeed, you need to join it back again. In other words, after a server is removed explicitly from the group you need to rejoin it manually (or have a script doing it automatically).

When is a member excluded from a group?

If the member becomes silent, the other members remove it from the group configuration. In practice this may happen when the member has crashed or there is a network disconnection.

The failure is detected after a given timeout elapses for a given member and a new configuration without the silent member in it is created.

What happens when one node is significantly lagging behind?

There is no method for defining policies for when to expel members automatically from the group. You need to find out why a member is lagging behind and fix that or remove the member from the group. Otherwise, if the server is so slow that it triggers the flow control, then the entire group slows down as well. The flow control can be configured according to the your needs.

Upon suspicion of a problem in the group, is there a special member responsible for triggering a reconfiguration?

No, there is no special member in the group in charge of triggering a reconfiguration.

Any member can suspect that there is a problem. All members need to (automatically) agree that a given member has failed. One member is in charge of expelling it from the group, by triggering a reconfiguration. Which member is responsible for expelling the member is not something you can control or set.

Can I use Group Replication for sharding?

Group Replication is designed to provide highly available replica sets; data and writes are duplicated on each member in the group. For scaling beyond what a single system can provide, you need an orchestration and sharding framework built around a number of Group Replication sets, where each replica set maintains and manages a given shard or partition of your total dataset. This type of setup, often called a “sharded cluster”, allows you to scale reads and writes linearly and without limit.

How do I use Group Replication with SELinux?

If SELinux is enabled, which you can verify using `sestatus -v`, then you need to enable the use of the Group Replication communication port. See Setting the TCP Port Context for Group Replication.

How do I use Group Replication with iptables?

If `iptables` is enabled, then you need to open up the Group Replication port for communication between the machines. To see the current rules in place on each machine, issue `iptables -L`. Assuming the port configured is 33061, enable communication over the necessary port by issuing `iptables -A INPUT -p tcp --dport 33061 -j ACCEPT`.

How do I recover the relay log for a replication channel used by a group member?

The replication channels used by Group Replication behave in the same way as replication channels used in asynchronous source to replica replication, and as such rely on the relay log. In the event of a change of the `relay_log` variable, or when the option is not set and the host name changes, there is a chance of errors. See Section 17.2.4.1, "The Relay Log" for a recovery procedure in this situation. Alternatively, another way of fixing the issue specifically in Group Replication is to issue a `STOP GROUP_REPLICATION` statement and then a `START GROUP_REPLICATION` statement to restart the instance. The Group Replication plugin creates the `group_replication_applier` channel again.

Why does Group Replication use two bind addresses?

Group Replication uses two bind addresses in order to split network traffic between the SQL address, used by clients to communicate with the member, and the `group_replication_local_address`, used internally by the group members to communicate. For example, assume a server with two network interfaces assigned to the network addresses `203.0.113.1` and `198.51.100.179`. In such a situation you could use `203.0.113.1:33061` for the internal group network address by setting `group_replication_local_address=203.0.113.1:33061`. Then you could use `198.51.100.179` for `hostname` and `3306` for the `port`. Client SQL applications would then connect to the member at `198.51.100.179:3306`. This enables you to configure different rules on the different networks. Similarly, the internal group communication can be separated from the network connection used for client applications, for increased security.

How does Group Replication use network addresses and hostnames?

Group Replication uses network connections between members and therefore its functionality is directly impacted by how you configure hostnames and ports. For example, Group Replication's distributed recovery process creates a connection to an existing group member using the server's hostname and port. When a member joins a group it receives the group membership information, using the network address information that is listed at `performance_schema.replication_group_members`. One of the members listed in that table is selected as the donor of the missing data from the group to the joining member.

This means that any value you configure using a hostname, such as the SQL network address or the group seeds address, must be a fully qualified name and resolvable by each member of the group. You can ensure this for example through DNS, or correctly configured `/etc/hosts` files, or other local processes. If a you want to configure the `MEMBER_HOST` value on a server, specify it using the `--report-host` option on the server before joining it to the group.

Important

The assigned value is used directly and is not affected by the `skip_name_resolve` system variable.

To configure `MEMBER_PORT` on a server, specify it using the `report_port` system variable.

Why did the auto increment setting on the server change?

When Group Replication is started on a server, the value of `auto_increment_increment` is changed to the value of `group_replication_auto_increment_increment`, which defaults to 7, and the value of `auto_increment_offset` is changed to the server ID. The changes are reverted when Group Replication is stopped. These settings avoid the selection of duplicate auto-increment values for writes on group members, which causes rollback of transactions. The default auto increment value of 7 for Group Replication represents a balance between the number of usable values and the permitted maximum size of a replication group (9 members).

The changes are only made and reverted if `auto_increment_increment` and `auto_increment_offset` each have their default value of 1. If their values have already been

modified from the default, Group Replication does not alter them. From MySQL 8.0, the system variables are also not modified when Group Replication is in single-primary mode, where only one server writes.

How do I find the primary?

If the group is operating in single-primary mode, it can be useful to find out which member is the primary. See Finding the Primary

Chapter 19 MySQL Shell

MySQL Shell is an advanced client and code editor for MySQL Server. In addition to the provided SQL functionality, similar to `mysql`, MySQL Shell provides scripting capabilities for JavaScript and Python and includes APIs for working with MySQL. MySQL Shell is a component that you can install separately.

The following discussion briefly describes MySQL Shell's capabilities. For more information, see the MySQL Shell manual, available at https://dev.mysql.com/doc/mysql-shell/en/.

MySQL Shell includes the following APIs implemented in JavaScript and Python which you can use to develop code that interacts with MySQL.

- The X DevAPI enables developers to work with both relational and document data when MySQL Shell is connected to a MySQL server using the X Protocol. This enables you to use MySQL as a Document Store, sometimes referred to as “using NoSQL”. For more information, see Chapter 20, *Using MySQL as a Document Store*. For documentation on the concepts and usage of X DevAPI, which is implemented in MySQL Shell, see X DevAPI User Guide.
- The AdminAPI enables database administrators to work with InnoDB Cluster, which provides an integrated solution for high availability and scalability using InnoDB based MySQL databases, without requiring advanced MySQL expertise. The AdminAPI also includes support for InnoDB ReplicaSet, which enables you to administer a set of MySQL instances running asynchronous GTID-based replication in a similar way to InnoDB Cluster. Additionally, the AdminAPI makes administration of MySQL Router easier, including integration with both InnoDB Cluster and InnoDB ReplicaSet. See MySQL AdminAPI.

MySQL Shell is available in two editions, the Community Edition and the Commercial Edition. The Community Edition is available free of charge. The Commercial Edition provides additional Enterprise features at low cost.

Chapter 20 Using MySQL as a Document Store

Table of Contents

This chapter introduces an alternative way of working with MySQL as a document store, sometimes referred to as “using NoSQL”. If your intention is to use MySQL in a traditional (SQL) way, this chapter is probably not relevant to you.

Traditionally, relational databases such as MySQL have usually required a schema to be defined before documents can be stored. The features described in this section enable you to use MySQL as a document store, which is a schema-less, and therefore schema-flexible, storage system for documents. For example, when you create documents describing products, you do not need to know and define all possible attributes of any products before storing and operating with the documents. This differs from working with a relational database and storing products in a table, when all columns of the table must be known and defined before adding any products to the database. The features described in this chapter enable you to choose how you configure MySQL, using only the document store model, or combining the flexibility of the document store model with the power of the relational model.

To use MySQL as a document store, you use the following server features:

- X Plugin enables MySQL Server to communicate with clients using X Protocol, which is a prerequisite for using MySQL as a document store. X Plugin is enabled by default in MySQL Server as of MySQL 8.0. For instructions to verify X Plugin installation and to configure and monitor X Plugin, see Section 20.5, “X Plugin”.
- X Protocol supports both CRUD and SQL operations, authentication via SASL, allows streaming (pipelining) of commands and is extensible on the protocol and the message layer. Clients compatible with X Protocol include MySQL Shell and MySQL 8.0 Connectors.
- Clients that communicate with a MySQL Server using X Protocol can use X DevAPI to develop applications. X DevAPI offers a modern programming interface with a simple yet powerful design which provides support for established industry standard concepts. This chapter explains how to get started using either the JavaScript or Python implementation of X DevAPI in MySQL Shell as a client. See X DevAPI User Guide for in-depth tutorials on using X DevAPI.

20.1 Interfaces to a MySQL Document Store

To work with MySQL as a document store, you use dedicated components and a choice of clients that support communicating with the MySQL server to develop document based applications.

- The following MySQL products support X Protocol and enable you to use X DevAPI in your chosen language to develop applications that communicate with a MySQL Server functioning as a document store:
  - MySQL Shell (which provides implementations of X DevAPI in JavaScript and Python)
  - Connector/C++
  - Connector/J
  - Connector/Node.js
  - Connector/NET
  - Connector/Python
- MySQL Shell is an interactive interface to MySQL supporting JavaScript, Python, or SQL modes. You can use MySQL Shell to prototype applications, execute queries and update data. Installing MySQL Shell has instructions to download and install MySQL Shell.
- The quick-start guides (tutorials) in this chapter help you to get started using MySQL Shell with MySQL as a document store.

  The quick-start guide for JavaScript is here: Section 20.3, “JavaScript Quick-Start Guide: MySQL Shell for Document Store”.

  The quick-start guide for Python is here: Section 20.4, “Python Quick-Start Guide: MySQL Shell for Document Store”.
- The *MySQL Shell User Guide* at MySQL Shell 8.0 provides detailed information about configuring and using MySQL Shell.

20.2 Document Store Concepts

This section explains the concepts introduced as part of using MySQL as a document store.

- JSON Document
- Collection
- CRUD Operations

JSON Document

A JSON document is a data structure composed of key-value pairs and is the fundamental structure for using MySQL as document store. For example, the world_x schema (installed later in this chapter) contains this document:

```
{
    "GNP": 4834,
    "_id": "00005de917d80000000000000023",
    "Code": "BWA",
    "Name": "Botswana",
    "IndepYear": 1966,
    "geography": {
        "Region": "Southern Africa",
        "Continent": "Africa",
```

```
        "SurfaceArea": 581730
    },
    "government": {
        "HeadOfState": "Festus G. Mogae",
        "GovernmentForm": "Republic"
    },
    "demographics": {
        "Population": 1622000,
        "LifeExpectancy": 39.29999923706055
    }
}
```

This document shows that the values of keys can be simple data types, such as integers or strings, but can also contain other documents, arrays, and lists of documents. For example, the `geography` key's value consists of multiple key-value pairs. A JSON document is represented internally using the MySQL binary JSON object, through the `JSON` MySQL datatype.

The most important differences between a document and the tables known from traditional relational databases are that the structure of a document does not have to be defined in advance, and a collection can contain multiple documents with different structures. Relational tables on the other hand require that their structure be defined, and all rows in the table must contain the same columns.

Collection

A collection is a container that is used to store JSON documents in a MySQL database. Applications usually run operations against a collection of documents, for example to find a specific document.

CRUD Operations

The four basic operations that can be issued against a collection are Create, Read, Update and Delete (CRUD). In terms of MySQL this means:

- Create a new document (insertion or addition)
- Read one or more documents (queries)
- Update one or more documents
- Delete one or more documents

20.3 JavaScript Quick-Start Guide: MySQL Shell for Document Store

This quick-start guide provides instructions to begin prototyping document store applications interactively with MySQL Shell. The guide includes the following topics:

- Introduction to MySQL functionality, MySQL Shell, and the `world_x` example schema.
- Operations to manage collections and documents.
- Operations to manage relational tables.
- Operations that apply to documents within tables.

To follow this quick-start guide you need a MySQL server with X Plugin installed, the default in 8.0, and MySQL Shell to use as the client. MySQL Shell 8.0 provides more in-depth information about MySQL Shell. The Document Store is accessed using X DevAPI, and MySQL Shell provides this API in both JavaScript and Python.

Related Information

- MySQL Shell 8.0 provides more in-depth information about MySQL Shell.

- See Installing MySQL Shell and Section 20.5, “X Plugin” for more information about the tools used in this quick-start guide.
- X DevAPI User Guide provides more examples of using X DevAPI to develop applications which use Document Store.
- A Python quick-start guide is also available.

20.3.1 MySQL Shell

This quick-start guide assumes a certain level of familiarity with MySQL Shell. The following section is a high level overview, see the MySQL Shell documentation for more information. MySQL Shell is a unified scripting interface to MySQL Server. It supports scripting in JavaScript and Python. JavaScript is the default processing mode.

Start MySQL Shell

After you have installed and started MySQL server, connect MySQL Shell to the server instance. You need to know the address of the MySQL server instance you plan to connect to. To be able to use the instance as a Document Store, the server instance must have X Plugin installed and you should connect to the server using X Protocol. For example to connect to the instance `ds1.example.com` on the default X Protocol port of 33060 use the network string *`user`*`@ds1.example.com:33060`.

Tip

If you connect to the instance using classic MySQL protocol, for example by using the default `port` of 3306 instead of the `mysqlx_port`, you *cannot* use the Document Store functionality shown in this tutorial. For example the `db` global object is not populated. To use the Document Store, always connect using X Protocol.

If MySQL Shell is not already running, open a terminal window and issue:

```
mysqlsh user@ds1.example.com:33060/world_x
```

Alternatively, if MySQL Shell is already running use the `\connect` command by issuing:

```
\connect user@ds1.example.com:33060/world_x
```

You need to specify the address of the MySQL server instance which you want to connect MySQL Shell to. For example in the previous example:

- *`user`* represents the user name of your MySQL account.
- `ds1.example.com` is the hostname of the server instance running MySQL. Replace this with the hostname of the MySQL server instance you are using as a Document Store.
- The default schema for this session is `world_x`. For instructions on setting up the `world_x` schema, see Section 20.3.2, “Download and Import world_x Database”.

For more information, see Section 4.2.5, “Connecting to the Server Using URI-Like Strings or Key-Value Pairs”.

Once MySQL Shell opens, the `mysql-js>` prompt indicates that the active language for this session is JavaScript.

```
mysql-js>
```

MySQL Shell supports input-line editing as follows:

- **left-arrow** and **right-arrow** keys move horizontally within the current input line.
- **up-arrow** and **down-arrow** keys move up and down through the set of previously entered lines.

- **Backspace** deletes the character before the cursor and typing new characters enters them at the cursor position.
- **Enter** sends the current input line to the server.

Get Help for MySQL Shell

Type `mysqlsh --help` at the prompt of your command interpreter for a list of command-line options.

```
mysqlsh --help
```

Type `\help` at the MySQL Shell prompt for a list of available commands and their descriptions.

```
mysql-js> \help
```

Type `\help` followed by a command name for detailed help about an individual MySQL Shell command. For example, to view help on the `\connect` command, issue:

```
mysql-js> \help \connect
```

Quit MySQL Shell

To quit MySQL Shell, issue the following command:

```
mysql-js> \quit
```

Related Information

- See Interactive Code Execution for an explanation of how interactive code execution works in MySQL Shell.
- See Getting Started with MySQL Shell to learn about session and connection alternatives.

20.3.2 Download and Import world_x Database

As part of this quick-start guide, an example schema is provided which is referred to as the `world_x` schema. Many of the examples demonstrate Document Store functionality using this schema. Start your MySQL server so that you can load the `world_x` schema, then follow these steps:

1. Download world_x-db.zip.
2. Extract the installation archive to a temporary location such as `/tmp/`. Unpacking the archive results in a single file named `world_x.sql`.
3. Import the `world_x.sql` file to your server. You can either:
   - Start MySQL Shell in SQL mode and import the file by issuing:

     ```
     mysqlsh -u root --sql --file /tmp/world_x-db/world_x.sql
     Enter password: ****
     ```

   - Set MySQL Shell to SQL mode while it is running and source the schema file by issuing:

     ```
     \sql
     Switching to SQL mode... Commands end with ;
     \source /tmp/world_x-db/world_x.sql
     ```

   Replace `/tmp/` with the path to the `world_x.sql` file on your system. Enter your password if prompted. A non-root account can be used as long as the account has privileges to create new schemas.

The world_x Schema

The `world_x` example schema contains the following JSON collection and relational tables:

- Collection
  - `countryinfo`: Information about countries in the world.
- Tables
  - `country`: Minimal information about countries of the world.
  - `city`: Information about some of the cities in those countries.
  - `countrylanguage`: Languages spoken in each country.

Related Information

- MySQL Shell Sessions explains session types.

20.3.3 Documents and Collections

When you are using MySQL as a Document Store, collections are containers within a schema that you can create, list, and drop. Collections contain JSON documents that you can add, find, update, and remove.

The examples in this section use the `countryinfo` collection in the `world_x` schema. For instructions on setting up the `world_x` schema, see Section 20.3.2, “Download and Import world_x Database”.

Documents

In MySQL, documents are represented as JSON objects. Internally, they are stored in an efficient binary format that enables fast lookups and updates.

- Simple document format for JavaScript:

```
{field1: "value", field2 : 10, "field 3": null}
```

An array of documents consists of a set of documents separated by commas and enclosed within `[` and `]` characters.

- Simple array of documents for JavaScript:

```
[{"Name": "Aruba", "Code:": "ABW"}, {"Name": "Angola", "Code:": "AGO"}]
```

MySQL supports the following JavaScript value types in JSON documents:

- numbers (integer and floating point)
- strings
- boolean (False and True)
- null
- arrays of more JSON values
- nested (or embedded) objects of more JSON values

Collections

Collections are containers for documents that share a purpose and possibly share one or more indexes. Each collection has a unique name and exists within a single schema.

The term schema is equivalent to a database, which means a group of database objects as opposed to a relational schema, used to enforce structure and constraints over data. A schema does not enforce conformity on the documents in a collection.

In this quick-start guide:

- Basic objects include:

Object form	Description
`db`	`db` is a global variable assigned to the current active schema. When you want to run operations against the schema, for example to retrieve a collection, you use methods available for the `db` variable.
`db.getCollections()`	db.getCollections() returns a list of collections in the schema. Use the list to get references to collection objects, iterate over them, and so on.

- Basic operations scoped by collections include:

Operation form	Description
`db.`*`name`*`.add()`	The add() method inserts one document or a list of documents into the named collection.
`db.`*`name`*`.find()`	The find() method returns some or all documents in the named collection.
`db.`*`name`*`.modify()`	The modify() method updates documents in the named collection.
`db.`*`name`*`.remove()`	The remove() method deletes one document or a list of documents from the named collection.

Related Information

- See Working with Collections for a general overview.
- CRUD EBNF Definitions provides a complete list of operations.

20.3.3.1 Create, List, and Drop Collections

In MySQL Shell, you can create new collections, get a list of the existing collections in a schema, and remove an existing collection from a schema. Collection names are case-sensitive and each collection name must be unique.

Confirm the Schema

To show the value that is assigned to the schema variable, issue:

```
mysql-js> db
```

If the schema value is not `Schema:world_x`, then set the `db` variable by issuing:

```
mysql-js> \use world_x
```

Create a Collection

To create a new collection in an existing schema, use the `db` object's `createCollection()` method. The following example creates a collection called `flags` in the `world_x` schema.

```
mysql-js> db.createCollection("flags")
```

The method returns a collection object.

```
<Collection:flags>
```

List Collections

To display all collections in the `world_x` schema, use the `db` object's `getCollections()` method. Collections returned by the server you are currently connected to appear between brackets.

```
mysql-js> db.getCollections()
[
    <Collection:countryinfo>,
    <Collection:flags>
]
```

Drop a Collection

To drop an existing collection from a schema, use the `db` object's `dropCollection()` method. For example, to drop the `flags` collection from the current schema, issue:

```
mysql-js> db.dropCollection("flags")
```

The `dropCollection()` method is also used in MySQL Shell to drop a relational table from a schema.

Related Information

- See Collection Objects for more examples.

20.3.3.2 Working with Collections

To work with the collections in a schema, use the `db` global object to access the current schema. In this example we are using the `world_x` schema imported previously, and the `countryinfo` collection. Therefore, the format of the operations you issue is `db.collection_name.operation`, where *`collection_name`* is the name of the collection which the operation is executed against. In the following examples, the operations are executed against the `countryinfo` collection.

Add a Document

Use the `add()` method to insert one document or a list of documents into an existing collection. Insert the following document into the `countryinfo` collection. As this is multi-line content, press **Enter** twice to insert the document.

```
mysql-js> db.countryinfo.add(
 {
    GNP: .6,
    IndepYear: 1967,
    Name: "Sealand",
    Code: "SEA",
    demographics: {
        LifeExpectancy: 79,
        Population: 27
    },
    geography: {
        Continent: "Europe",
        Region: "British Islands",
        SurfaceArea: 193
    },
    government: {
        GovernmentForm: "Monarchy",
        HeadOfState: "Michael Bates"
    }
  }
)
```

The method returns the status of the operation. You can verify the operation by searching for the document. For example:

```
mysql-js> db.countryinfo.find("Name = 'Sealand'")
{
    "GNP": 0.6,
```

```
    "_id": "00005e2ff4af00000000000000f4",
    "Name": "Sealand",
    "Code:": "SEA",
    "IndepYear": 1967,
    "geography": {
        "Region": "British Islands",
        "Continent": "Europe",
        "SurfaceArea": 193
    },
    "government": {
        "HeadOfState": "Michael Bates",
        "GovernmentForm": "Monarchy"
    },
    "demographics": {
        "Population": 27,
        "LifeExpectancy": 79
    }
}
```

Note that in addition to the fields specified when the document was added, there is one more field, the `_id`. Each document requires an identifier field called `_id`. The value of the `_id` field must be unique among all documents in the same collection. In MySQL 8.0.11 and higher, document IDs are generated by the server, not the client, so MySQL Shell does not automatically set an `_id` value. A MySQL server at 8.0.11 or higher sets an `_id` value if the document does not contain the `_id` field. A MySQL server at an earlier 8.0 release or at 5.7 does not set an `_id` value in this situation, so you must specify it explicitly. If you do not, MySQL Shell returns error 5115 `Document is missing a required field`. For more information see Understanding Document IDs.

Related Information

- See CollectionAddFunction for the full syntax definition.
- See Understanding Document IDs.

20.3.3.3 Find Documents

You can use the `find()` method to query for and return documents from a collection in a schema. MySQL Shell provides additional methods to use with the `find()` method to filter and sort the returned documents.

MySQL provides the following operators to specify search conditions: `OR` (`||`), `AND` (`&&`), `XOR`, `IS`, `NOT`, `BETWEEN`, `IN`, `LIKE`, `!=`, `<>`, `>`, `>=`, `<`, `<=`, `&`, `|`, `<<`, `>>`, `+`, `-`, `*`, `/`, `~`, and `%`.

Find All Documents in a Collection

To return all documents in a collection, use the `find()` method without specifying search conditions. For example, the following operation returns all documents in the `countryinfo` collection.

```
mysql-js> db.countryinfo.find()
[
     {
          "GNP": 828,
          "Code:": "ABW",
          "Name": "Aruba",
          "IndepYear": null,
          "geography": {
              "Continent": "North America",
              "Region": "Caribbean",
              "SurfaceArea": 193
          },
          "government": {
              "GovernmentForm": "Nonmetropolitan Territory of The Netherlands",
              "HeadOfState": "Beatrix"
          }
          "demographics": {
              "LifeExpectancy": 78.4000015258789,
              "Population": 103000
```

```
          },
          ...
      }
 ]
240 documents in set (0.00 sec)
```

The method produces results that contain operational information in addition to all documents in the collection.

An empty set (no matching documents) returns the following information:

```
Empty set (0.00 sec)
```

Filter Searches

You can include search conditions with the `find()` method. The syntax for expressions that form a search condition is the same as that of traditional MySQL Chapter 12, *Functions and Operators*. You must enclose all expressions in quotes. For the sake of brevity, some of the examples do not display output.

A simple search condition could consist of the `Name` field and a value we know is in a document. The following example returns a single document:

```
mysql-js> db.countryinfo.find("Name = 'Australia'")
[
    {
        "GNP": 351182,
        "Code:": "AUS",
        "Name": "Australia",
        "IndepYear": 1901,
        "geography": {
            "Continent": "Oceania",
            "Region": "Australia and New Zealand",
            "SurfaceArea": 7741220
        },
        "government": {
            "GovernmentForm": "Constitutional Monarchy, Federation",
            "HeadOfState": "Elisabeth II"
        }
        "demographics": {
            "LifeExpectancy": 79.80000305175781,
            "Population": 18886000
        },
    }
]
```

The following example searches for all countries that have a GNP higher than $500 billion. The `countryinfo` collection measures GNP in units of million.

```
mysql-js> db.countryinfo.find("GNP > 500000")
...[output removed]
10 documents in set (0.00 sec)
```

The Population field in the following query is embedded within the demographics object. To access the embedded field, use a period between demographics and Population to identify the relationship. Document and field names are case-sensitive.

```
mysql-js> db.countryinfo.find("GNP > 500000 and demographics.Population < 100000000")
...[output removed]
6 documents in set (0.00 sec)
```

Arithmetic operators in the following expression are used to query for countries with a GNP per capita higher than $30000. Search conditions can include arithmetic operators and most MySQL functions.

Note

Seven documents in the `countryinfo` collection have a population value of zero. Therefore warning messages appear at the end of the output.

```
mysql-js> db.countryinfo.find("GNP*1000000/demographics.Population > 30000")
...[output removed]
9 documents in set, 7 warnings (0.00 sec)
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
```

You can separate a value from the search condition by using the `bind()` method. For example, instead of specifying a hard-coded country name as the condition, substitute a named placeholder consisting of a colon followed by a name that begins with a letter, such as *country*. Then use the `bind(placeholder, value)` method as follows:

```
mysql-js> db.countryinfo.find("Name = :country").bind("country", "Italy")
{
    "GNP": 1161755,
    "_id": "00005de917d80000000000000006a",
    "Code": "ITA",
    "Name": "Italy",
    "Airports": [],
    "IndepYear": 1861,
    "geography": {
        "Region": "Southern Europe",
        "Continent": "Europe",
        "SurfaceArea": 301316
    },
    "government": {
        "HeadOfState": "Carlo Azeglio Ciampi",
        "GovernmentForm": "Republic"
    },
    "demographics": {
        "Population": 57680000,
        "LifeExpectancy": 79
    }
}
1 document in set (0.01 sec)
```

Tip

Within a program, binding enables you to specify placeholders in your expressions, which are filled in with values before execution and can benefit from automatic escaping, as appropriate.

Always use binding to sanitize input. Avoid introducing values in queries using string concatenation, which can produce invalid input and, in some cases, can cause security issues.

You can use placeholders and the `bind()` method to create saved searches which you can then call with different values. For example to create a saved search for a country:

```
mysql-js> var myFind = db.countryinfo.find("Name = :country")
mysql-js> myFind.bind('country', 'France')
{
    "GNP": 1424285,
    "_id": "00005de917d80000000000000048",
    "Code": "FRA",
    "Name": "France",
    "IndepYear": 843,
    "geography": {
        "Region": "Western Europe",
        "Continent": "Europe",
        "SurfaceArea": 551500
    },
    "government": {
        "HeadOfState": "Jacques Chirac",
        "GovernmentForm": "Republic"
```

```
    },
    "demographics": {
        "Population": 59225700,
        "LifeExpectancy": 78.80000305175781
    }
}
1 document in set (0.0028 sec)

mysql-js> myFind.bind('country', 'Germany')
{
    "GNP": 2133367,
    "_id": "00005de917d80000000000000038",
    "Code": "DEU",
    "Name": "Germany",
    "IndepYear": 1955,
    "geography": {
        "Region": "Western Europe",
        "Continent": "Europe",
        "SurfaceArea": 357022
    },
    "government": {
        "HeadOfState": "Johannes Rau",
        "GovernmentForm": "Federal Republic"
    },
    "demographics": {
        "Population": 82164700,
        "LifeExpectancy": 77.4000015258789
    }
}

1 document in set (0.0026 sec)
```

Project Results

You can return specific fields of a document, instead of returning all the fields. The following example returns the GNP and Name fields of all documents in the `countryinfo` collection matching the search conditions.

Use the `fields()` method to pass the list of fields to return.

```
mysql-js> db.countryinfo.find("GNP > 5000000").fields(["GNP", "Name"])
[
    {
        "GNP": 8510700,
        "Name": "United States"
    }
]
1 document in set (0.00 sec)
```

In addition, you can alter the returned documents—adding, renaming, nesting and even computing new field values—with an expression that describes the document to return. For example, alter the names of the fields with the following expression to return only two documents.

```
mysql-js> db.countryinfo.find().fields(
mysqlx.expr('{"Name": upper(Name), "GNPPerCapita": GNP*1000000/demographics.Population}')).limit(2)
{
    "Name": "ARUBA",
    "GNPPerCapita": 8038.834951456311
}
{
    "Name": "AFGHANISTAN",
    "GNPPerCapita": 263.0281690140845
}
```

Limit, Sort, and Skip Results

You can apply the `limit()`, `sort()`, and `skip()` methods to manage the number and order of documents returned by the `find()` method.

To specify the number of documents included in a result set, append the `limit()` method with a value to the `find()` method. The following query returns the first five documents in the `countryinfo` collection.

```
mysql-js> db.countryinfo.find().limit(5)
... [output removed]
5 documents in set (0.00 sec)
```

To specify an order for the results, append the `sort()` method to the `find()` method. Pass to the `sort()` method a list of one or more fields to sort by and, optionally, the descending (`desc`) or ascending (`asc`) attribute as appropriate. Ascending order is the default order type.

For example, the following query sorts all documents by the IndepYear field and then returns the first eight documents in descending order.

```
mysql-js> db.countryinfo.find().sort(["IndepYear desc"]).limit(8)
... [output removed]
8 documents in set (0.00 sec)
```

By default, the `limit()` method starts from the first document in the collection. You can use the `skip()` method to change the starting document. For example, to ignore the first document and return the next eight documents matching the condition, pass to the `skip()` method a value of 1.

```
mysql-js> db.countryinfo.find().sort(["IndepYear desc"]).limit(8).skip(1)
... [output removed]
8 documents in set (0.00 sec)
```

Related Information

- The MySQL Reference Manual provides detailed documentation on functions and operators.
- See CollectionFindFunction for the full syntax definition.

20.3.3.4 Modify Documents

You can use the `modify()` method to update one or more documents in a collection. The X DevAPI provides additional methods for use with the `modify()` method to:

- Set and unset fields within documents.
- Append, insert, and delete arrays.
- Bind, limit, and sort the documents to be modified.

Set and Unset Document Fields

The `modify()` method works by filtering a collection to include only the documents to be modified and then applying the operations that you specify to those documents.

In the following example, the `modify()` method uses the search condition to identify the document to change and then the `set()` method replaces two values within the nested demographics object.

```
mysql-js> db.countryinfo.modify("Code = 'SEA'").set(
"demographics", {"LifeExpectancy": 78, "Population": 28})
```

After you modify a document, use the `find()` method to verify the change.

To remove content from a document, use the `modify()` and `unset()` methods. For example, the following query removes the GNP from a document that matches the search condition.

```
mysql-js> db.countryinfo.modify("Name = 'Sealand'").unset("GNP")
```

Use the `find()` method to verify the change.

```
mysql-js> db.countryinfo.find("Name = 'Sealand'")
```

```
{
    "_id": "00005e2ff4af00000000000000f4",
    "Name": "Sealand",
    "Code:": "SEA",
    "IndepYear": 1967,
    "geography": {
        "Region": "British Islands",
        "Continent": "Europe",
        "SurfaceArea": 193
    },
    "government": {
        "HeadOfState": "Michael Bates",
        "GovernmentForm": "Monarchy"
    },
    "demographics": {
        "Population": 27,
        "LifeExpectancy": 79
    }
}
```

Append, Insert, and Delete Arrays

To append an element to an array field, or insert, or delete elements in an array, use the `arrayAppend()`, `arrayInsert()`, or `arrayDelete()` methods. The following examples modify the `countryinfo` collection to enable tracking of international airports.

The first example uses the `modify()` and `set()` methods to create a new Airports field in all documents.

Caution

Use care when you modify documents without specifying a search condition; doing so modifies all documents in the collection.

```
mysql-js> db.countryinfo.modify("true").set("Airports", [])
```

With the Airports field added, the next example uses the `arrayAppend()` method to add a new airport to one of the documents. *$.Airports* in the following example represents the Airports field of the current document.

```
mysql-js> db.countryinfo.modify("Name = 'France'").arrayAppend("$.Airports", "ORY")
```

Use `find()` to see the change.

```
mysql-js> db.countryinfo.find("Name = 'France'")
{
    "GNP": 1424285,
    "_id": "00005de917d80000000000000048",
    "Code": "FRA",
    "Name": "France",
    "Airports": [
        "ORY"
    ],
    "IndepYear": 843,
    "geography": {
        "Region": "Western Europe",
        "Continent": "Europe",
        "SurfaceArea": 551500
    },
    "government": {
        "HeadOfState": "Jacques Chirac",
        "GovernmentForm": "Republic"
    },
    "demographics": {
        "Population": 59225700,
        "LifeExpectancy": 78.80000305175781
    }
}
```

To insert an element at a different position in the array, use the `arrayInsert()` method to specify which index to insert in the path expression. In this case, the index is 0, or the first element in the array.

```
mysql-js> db.countryinfo.modify("Name = 'France'").arrayInsert("$.Airports[0]", "CDG")
```

To delete an element from the array, you must pass to the `arrayDelete()` method the index of the element to be deleted.

```
mysql-js> db.countryinfo.modify("Name = 'France'").arrayDelete("$.Airports[1]")
```

Related Information

- The MySQL Reference Manual provides instructions to help you search for and modify JSON values.
- See CollectionModifyFunction for the full syntax definition.

20.3.3.5 Remove Documents

You can use the `remove()` method to delete some or all documents from a collection in a schema. The X DevAPI provides additional methods for use with the `remove()` method to filter and sort the documents to be removed.

Remove Documents Using Conditions

The following example passes a search condition to the `remove()` method. All documents matching the condition are removed from the `countryinfo` collection. In this example, one document matches the condition.

```
mysql-js> db.countryinfo.remove("Code = 'SEA'")
```

Remove the First Document

To remove the first document in the `countryinfo` collection, use the `limit()` method with a value of 1.

```
mysql-js> db.countryinfo.remove("true").limit(1)
```

Remove the Last Document in an Order

The following example removes the last document in the `countryinfo` collection by country name.

```
mysql-js> db.countryinfo.remove("true").sort(["Name desc"]).limit(1)
```

Remove All Documents in a Collection

You can remove all documents in a collection. To do so, use the `remove("true")` method without specifying a search condition.

Caution

Use care when you remove documents without specifying a search condition. This action deletes all documents from the collection.

Alternatively, use the `db.drop_collection('countryinfo')` operation to delete the `countryinfo` collection.

Related Information

- See CollectionRemoveFunction for the full syntax definition.
- See Section 20.3.2, “Download and Import world_x Database” for instructions to recreate the `world_x` schema.

20.3.3.6 Create and Drop Indexes

Indexes are used to find documents with specific field values quickly. Without an index, MySQL must begin with the first document and then read through the entire collection to find the relevant fields. The larger the collection, the more this costs. If a collection is large and queries on a specific field are common, then consider creating an index on a specific field inside a document.

For example, the following query performs better with an index on the Population field:

```
mysql-js> db.countryinfo.find("demographics.Population < 100")
...[output removed]
8 documents in set (0.00 sec)
```

The `createIndex()` method creates an index that you can define with a JSON document that specifies which fields to use. This section is a high level overview of indexing. For more information see Indexing Collections.

Add a Nonunique Index

To create a nonunique index, pass an index name and the index information to the `createIndex()` method. Duplicate index names are prohibited.

The following example specifies an index named `popul`, defined against the `Population` field from the `demographics` object, indexed as an `Integer` numeric value. The final parameter indicates whether the field should require the `NOT NULL` constraint. If the value is `false`, the field can contain `NULL` values. The index information is a JSON document with details of one or more fields to include in the index. Each field definition must include the full document path to the field, and specify the type of the field.

```
mysql-js> db.countryinfo.createIndex("popul", {fields:
[{field: '$.demographics.Population', type: 'INTEGER'}]})
```

Here, the index is created using an integer numeric value. Further options are available, including options for use with GeoJSON data. You can also specify the type of index, which has been omitted here because the default type “index” is appropriate.

Add a Unique Index

To create a unique index, pass an index name, the index definition, and the index type “unique” to the `createIndex()` method. This example shows a unique index created on the country name (`"Name"`), which is another common field in the `countryinfo` collection to index. In the index field description, `"TEXT(40)"` represents the number of characters to index, and `"required": True` specifies that the field is required to exist in the document.

```
mysql-js> db.countryinfo.createIndex("name",
{"fields": [{"field": "$.Name", "type": "TEXT(40)", "required": true}], "unique": true})
```

Drop an Index

To drop an index, pass the name of the index to drop to the `dropIndex()` method. For example, you can drop the “popul” index as follows:

```
mysql-js> db.countryinfo.dropIndex("popul")
```

Related Information

- See Indexing Collections for more information.
- See Defining an Index for more information on the JSON document that defines an index.
- See Collection Index Management Functions for the full syntax definition.

20.3.4 Relational Tables

You can also use X DevAPI to work with relational tables. In MySQL, each relational table is associated with a particular storage engine. The examples in this section use `InnoDB` tables in the `world_x` schema.

Confirm the Schema

To show the schema that is assigned to the `db` global variable, issue `db`.

```
mysql-js> db
<Schema:world_x>
```

If the returned value is not `Schema:world_x`, set the `db` variable as follows:

```
mysql-js> \use world_x
Schema `world_x` accessible through db.
```

Show All Tables

To display all relational tables in the `world_x` schema, use the `getTables()` method on the `db` object.

```
mysql-js> db.getTables()
{
    "city": <Table:city>,
    "country": <Table:country>,
    "countrylanguage": <Table:countrylanguage>
}
```

Basic Table Operations

Basic operations scoped by tables include:

Operation form	Description
`db.name.insert()`	The insert() method inserts one or more records into the named table.
`db.name.select()`	The select() method returns some or all records in the named table.
`db.name.update()`	The update() method updates records in the named table.
`db.name.delete()`	The delete() method deletes one or more records from the named table.

Related Information

- See Working with Relational Tables for more information.
- CRUD EBNF Definitions provides a complete list of operations.
- See Section 20.3.2, “Download and Import world_x Database” for instructions on setting up the `world_x` schema sample.

20.3.4.1 Insert Records into Tables

You can use the `insert()` method with the `values()` method to insert records into an existing relational table. The `insert()` method accepts individual columns or all columns in the table. Use one or more `values()` methods to specify the values to be inserted.

Insert a Complete Record

To insert a complete record, pass to the `insert()` method all columns in the table. Then pass to the `values()` method one value for each column in the table. For example, to add a new record to the city table in the `world_x` schema, insert the following record and press **Enter** twice.

```
mysql-js> db.city.insert("ID", "Name", "CountryCode", "District", "Info").values(
None, "Olympia", "USA", "Washington", '{"Population": 5000}')
```

The city table has five columns: ID, Name, CountryCode, District, and Info. Each value must match the data type of the column it represents.

Insert a Partial Record

The following example inserts values into the ID, Name, and CountryCode columns of the city table.

```
mysql-js> db.city.insert("ID", "Name", "CountryCode").values(
None, "Little Falls", "USA").values(None, "Happy Valley", "USA")
```

When you specify columns using the `insert()` method, the number of values must match the number of columns. In the previous example, you must supply three values to match the three columns specified.

Related Information

- See TableInsertFunction for the full syntax definition.

20.3.4.2 Select Tables

You can use the `select()` method to query for and return records from a table in a database. The X DevAPI provides additional methods to use with the `select()` method to filter and sort the returned records.

MySQL provides the following operators to specify search conditions: `OR` (`||`), `AND` (`&&`), `XOR`, `IS`, `NOT`, `BETWEEN`, `IN`, `LIKE`, `!=`, `<>`, `>`, `>=`, `<`, `<=`, `&`, `|`, `<<`, `>>`, `+`, `-`, `*`, `/`, `~`, and `%`.

Select All Records

To issue a query that returns all records from an existing table, use the `select()` method without specifying search conditions. The following example selects all records from the city table in the `world_x` database.

Note

Limit the use of the empty `select()` method to interactive statements. Always use explicit column-name selections in your application code.

```
mysql-js> db.city.select()
+------+------------+-------------+------------+-------------------------+
| ID   | Name       | CountryCode | District   | Info                    |
+------+------------+-------------+------------+-------------------------+
|    1 | Kabul      | AFG         | Kabol      |{"Population": 1780000}  |
|    2 | Qandahar   | AFG         | Qandahar   |{"Population": 237500}   |
|    3 | Herat      | AFG         | Herat      |{"Population": 186800}   |
...    ...          ...           ...          ...
| 4079 | Rafah      | PSE         | Rafah      |{"Population": 92020}    |
+------+------- ----+-------------+------------+-------------------------+
4082 rows in set (0.01 sec)
```

An empty set (no matching records) returns the following information:

```
Empty set (0.00 sec)
```

Filter Searches

To issue a query that returns a set of table columns, use the `select()` method and specify the columns to return between square brackets. This query returns the Name and CountryCode columns from the city table.

```
mysql-js> db.city.select(["Name", "CountryCode"])
```

```
+------------------+-------------+
| Name             | CountryCode |
+------------------+-------------+
| Kabul            | AFG         |
| Qandahar         | AFG         |
| Herat            | AFG         |
| Mazar-e-Sharif   | AFG         |
| Amsterdam        | NLD         |
...                ...
| Rafah            | PSE         |
| Olympia          | USA         |
| Little Falls     | USA         |
| Happy Valley     | USA         |
+------------------+-------------+
4082 rows in set (0.00 sec)
```

To issue a query that returns rows matching specific search conditions, use the `where()` method to include those conditions. For example, the following example returns the names and country codes of the cities that start with the letter Z.

```
mysql-js> db.city.select(["Name", "CountryCode"]).where("Name like 'Z%'")
+------------------+-------------+
| Name             | CountryCode |
+------------------+-------------+
| Zaanstad         | NLD         |
| Zoetermeer       | NLD         |
| Zwolle           | NLD         |
| Zenica           | BIH         |
| Zagazig          | EGY         |
| Zaragoza         | ESP         |
| Zamboanga        | PHL         |
| Zahedan          | IRN         |
| Zanjan           | IRN         |
| Zabol            | IRN         |
| Zama             | JPN         |
| Zhezqazghan      | KAZ         |
| Zhengzhou        | CHN         |
...                ...
| Zeleznogorsk     | RUS         |
+------------------+-------------+
59 rows in set (0.00 sec)
```

You can separate a value from the search condition by using the `bind()` method. For example, instead of using "Name = 'Z%' " as the condition, substitute a named placeholder consisting of a colon followed by a name that begins with a letter, such as *name*. Then include the placeholder and value in the `bind()` method as follows:

```
mysql-js> db.city.select(["Name", "CountryCode"]).
              where("Name like :name").bind("name", "Z%")
```

Tip

Within a program, binding enables you to specify placeholders in your expressions, which are filled in with values before execution and can benefit from automatic escaping, as appropriate.

Always use binding to sanitize input. Avoid introducing values in queries using string concatenation, which can produce invalid input and, in some cases, can cause security issues.

Project Results

To issue a query using the `AND` operator, add the operator between search conditions in the `where()` method.

```
mysql-js> db.city.select(["Name", "CountryCode"]).where(
"Name like 'Z%' and CountryCode = 'CHN'")
+---------------+-------------+
```

```
| Name           | CountryCode |
+----------------+-------------+
| Zhengzhou      | CHN         |
| Zibo           | CHN         |
| Zhangjiakou    | CHN         |
| Zhuzhou        | CHN         |
| Zhangjiang     | CHN         |
| Zigong         | CHN         |
| Zaozhuang      | CHN         |
...              ...
| Zhangjiagang   | CHN         |
+----------------+-------------+
22 rows in set (0.01 sec)
```

To specify multiple conditional operators, you can enclose the search conditions in parenthesis to change the operator precedence. The following example demonstrates the placement of `AND` and `OR` operators.

```
mysql-js> db.city.select(["Name", "CountryCode"]).
where("Name like 'Z%' and (CountryCode = 'CHN' or CountryCode = 'RUS')")
+-------------------+-------------+
| Name              | CountryCode |
+-------------------+-------------+
| Zhengzhou         | CHN         |
| Zibo              | CHN         |
| Zhangjiakou       | CHN         |
| Zhuzhou           | CHN         |
...                 ...
| Zeleznogorsk      | RUS         |
+-------------------+-------------+
29 rows in set (0.01 sec)
```

Limit, Order, and Offset Results

You can apply the `limit()`, `orderBy()`, and `offSet()` methods to manage the number and order of records returned by the `select()` method.

To specify the number of records included in a result set, append the `limit()` method with a value to the `select()` method. For example, the following query returns the first five records in the country table.

```
mysql-js> db.country.select(["Code", "Name"]).limit(5)
+------+-------------+
| Code | Name        |
+------+-------------+
| ABW  | Aruba       |
| AFG  | Afghanistan |
| AGO  | Angola      |
| AIA  | Anguilla    |
| ALB  | Albania     |
+------+-------------+
5 rows in set (0.00 sec)
```

To specify an order for the results, append the `orderBy()` method to the `select()` method. Pass to the `orderBy()` method a list of one or more columns to sort by and, optionally, the descending (`desc`) or ascending (`asc`) attribute as appropriate. Ascending order is the default order type.

For example, the following query sorts all records by the Name column and then returns the first three records in descending order .

```
mysql-js> db.country.select(["Code", "Name"]).orderBy(["Name desc"]).limit(3)
+------+------------+
| Code | Name       |
+------+------------+
| ZWE  | Zimbabwe   |
| ZMB  | Zambia     |
| YUG  | Yugoslavia |
+------+------------+
3 rows in set (0.00 sec)
```

By default, the `limit()` method starts from the first record in the table. You can use the `offset()` method to change the starting record. For example, to ignore the first record and return the next three records matching the condition, pass to the `offset()` method a value of 1.

```
mysql-js> db.country.select(["Code", "Name"]).orderBy(["Name desc"]).limit(3).offset(1)
+------+------------+
| Code | Name       |
+------+------------+
| ZMB  | Zambia     |
| YUG  | Yugoslavia |
| YEM  | Yemen      |
+------+------------+
3 rows in set (0.00 sec)
```

Related Information

- The MySQL Reference Manual provides detailed documentation on functions and operators.
- See TableSelectFunction for the full syntax definition.

20.3.4.3 Update Tables

You can use the `update()` method to modify one or more records in a table. The `update()` method works by filtering a query to include only the records to be updated and then applying the operations you specify to those records.

To replace a city name in the city table, pass to the `set()` method the new city name. Then, pass to the `where()` method the city name to locate and replace. The following example replaces the city Peking with Beijing.

```
mysql-js> db.city.update().set("Name", "Beijing").where("Name = 'Peking'")
```

Use the `select()` method to verify the change.

```
mysql-js> db.city.select(["ID", "Name", "CountryCode", "District", "Info"]).where("Name = 'Beijing'")
+------+-----------+-------------+----------+----------------------------+
| ID   | Name      | CountryCode | District | Info                       |
+------+-----------+-------------+----------+----------------------------+
| 1891 | Beijing   | CHN         | Peking   | {"Population": 7472000}    |
+------+-----------+-------------+----------+----------------------------+
1 row in set (0.00 sec)
```

Related Information

- See TableUpdateFunction for the full syntax definition.

20.3.4.4 Delete Tables

You can use the `delete()` method to remove some or all records from a table in a database. The X DevAPI provides additional methods to use with the `delete()` method to filter and order the records to be deleted.

Delete Records Using Conditions

The following example passes search conditions to the `delete()` method. All records matching the condition are deleted from the city table. In this example, one record matches the condition.

```
mysql-js> db.city.delete().where("Name = 'Olympia'")
```

Delete the First Record

To delete the first record in the city table, use the `limit()` method with a value of 1.

```
mysql-js> db.city.delete().limit(1)
```

Delete All Records in a Table

You can delete all records in a table. To do so, use the `delete()` method without specifying a search condition.

Caution

Use care when you delete records without specifying a search condition; doing so deletes all records from the table.

Drop a Table

The `dropCollection()` method is also used in MySQL Shell to drop a relational table from a database. For example, to drop the `citytest` table from the `world_x` database, issue:

```
mysql-js> session.dropCollection("world_x", "citytest")
```

Related Information

- See TableDeleteFunction for the full syntax definition.
- See Section 20.3.2, “Download and Import world_x Database” for instructions to recreate the `world_x` database.

20.3.5 Documents in Tables

In MySQL, a table may contain traditional relational data, JSON values, or both. You can combine traditional data with JSON documents by storing the documents in columns having a native `JSON` data type.

Examples in this section use the city table in the `world_x` schema.

city Table Description

The city table has five columns (or fields).

```
+---------------+------------+-------+-------+---------+------------------+
| Field         | Type       | Null  | Key   | Default | Extra            |
+---------------+------------+-------+-------+---------+------------------+
| ID            | int(11)    | NO    | PRI   | null    | auto_increment   |
| Name          | char(35)   | NO    |       |         |                  |
| CountryCode   | char(3)    | NO    |       |         |                  |
| District      | char(20)   | NO    |       |         |                  |
| Info          | json       | YES   |       | null    |                  |
+---------------+------------+-------+-------+---------+------------------+
```

Insert a Record

To insert a document into the column of a table, pass to the `values()` method a well-formed JSON document in the correct order. In the following example, a document is passed as the final value to be inserted into the Info column.

```
mysql-js> db.city.insert().values(
None, "San Francisco", "USA", "California", '{"Population":830000}')
```

Select a Record

You can issue a query with a search condition that evaluates document values in the expression.

```
mysql-js> db.city.select(["ID", "Name", "CountryCode", "District", "Info"]).where(
"CountryCode = :country and Info->'$.Population' > 1000000").bind(
'country', 'USA')
+------+---------------+-------------+---------------+----------------------------+
| ID   | Name          | CountryCode | District      | Info                       |
```

```
+------+---------------+-------------+---------------+-----------------------------+
| 3793 | New York      | USA         | New York      | {"Population": 8008278}     |
| 3794 | Los Angeles   | USA         | California    | {"Population": 3694820}     |
| 3795 | Chicago       | USA         | Illinois      | {"Population": 2896016}     |
| 3796 | Houston       | USA         | Texas         | {"Population": 1953631}     |
| 3797 | Philadelphia  | USA         | Pennsylvania  | {"Population": 1517550}     |
| 3798 | Phoenix       | USA         | Arizona       | {"Population": 1321045}     |
| 3799 | San Diego     | USA         | California    | {"Population": 1223400}     |
| 3800 | Dallas        | USA         | Texas         | {"Population": 1188580}     |
| 3801 | San Antonio   | USA         | Texas         | {"Population": 1144646}     |
+------+---------------+-------------+---------------+-----------------------------+
9 rows in set (0.01 sec)
```

Related Information

- See Working with Relational Tables and Documents for more information.
- See Section 11.5, “The JSON Data Type” for a detailed description of the data type.

20.4 Python Quick-Start Guide: MySQL Shell for Document Store

This quick-start guide provides instructions to begin prototyping document store applications interactively with MySQL Shell. The guide includes the following topics:

- Introduction to MySQL functionality, MySQL Shell, and the `world_x` example schema.
- Operations to manage collections and documents.
- Operations to manage relational tables.
- Operations that apply to documents within tables.

To follow this quick-start guide you need a MySQL server with X Plugin installed, the default in 8.0, and MySQL Shell to use as the client. MySQL Shell includes X DevAPI, implemented in both JavaScript and Python, which enables you to connect to the MySQL server instance using X Protocol and use the server as a Document Store.

Related Information

- MySQL Shell 8.0 provides more in-depth information about MySQL Shell.
- See Installing MySQL Shell and Section 20.5, “X Plugin” for more information about the tools used in this quick-start guide.
- See Supported Languages for more information about the languages MySQL Shell supports.
- X DevAPI User Guide provides more examples of using X DevAPI to develop applications which use MySQL as a Document Store.
- A JavaScript quick-start guide is also available.

20.4.1 MySQL Shell

This quick-start guide assumes a certain level of familiarity with MySQL Shell. The following section is a high level overview, see the MySQL Shell documentation for more information. MySQL Shell is a unified scripting interface to MySQL Server. It supports scripting in JavaScript and Python. JavaScript is the default processing mode.

Start MySQL Shell

After you have installed and started MySQL server, connect MySQL Shell to the server instance. You need to know the address of the MySQL server instance you plan to connect to. To be able to use the instance as a Document Store, the server instance must have X Plugin installed and you should

connect to the server using X Protocol. For example to connect to the instance `ds1.example.com` on the default X Protocol port of 33060 use the network string *`user`*`@ds1.example.com:33060`.

Tip

If you connect to the instance using classic MySQL protocol, for example by using the default `port` of 3306 instead of the `mysqlx_port`, you *cannot* use the Document Store functionality shown in this tutorial. For example the `db` global object is not populated. To use the Document Store, always connect using X Protocol.

If MySQL Shell is not already running, open a terminal window and issue:

```
mysqlsh user@ds1.example.com:33060/world_x
```

Alternatively, if MySQL Shell is already running use the `\connect` command by issuing:

```
\connect user@ds1.example.com:33060/world_x
```

You need to specify the address of the MySQL server instance which you want to connect MySQL Shell to. For example in the previous example:

- *`user`* represents the user name of your MySQL account.
- `ds1.example.com` is the hostname of the server instance running MySQL. Replace this with the hostname of the MySQL server instance you are using as a Document Store.
- The default schema for this session is `world_x`. For instructions on setting up the `world_x` schema, see Section 20.4.2, “Download and Import world_x Database”.

For more information, see Section 4.2.5, “Connecting to the Server Using URI-Like Strings or Key-Value Pairs”.

Once MySQL Shell opens, the `mysql-js>` prompt indicates that the active language for this session is JavaScript. To switch MySQL Shell to Python mode, use the `\py` command.

```
mysql-js> \py
Switching to Python mode...
mysql-py>
```

MySQL Shell supports input-line editing as follows:

- **left-arrow** and **right-arrow** keys move horizontally within the current input line.
- **up-arrow** and **down-arrow** keys move up and down through the set of previously entered lines.
- **Backspace** deletes the character before the cursor and typing new characters enters them at the cursor position.
- **Enter** sends the current input line to the server.

Get Help for MySQL Shell

Type `mysqlsh --help` at the prompt of your command interpreter for a list of command-line options.

```
mysqlsh --help
```

Type `\help` at the MySQL Shell prompt for a list of available commands and their descriptions.

```
mysql-py> \help
```

Type `\help` followed by a command name for detailed help about an individual MySQL Shell command. For example, to view help on the `\connect` command, issue:

```
mysql-py> \help \connect
```

Quit MySQL Shell

To quit MySQL Shell, issue the following command:

```
mysql-py> \quit
```

Related Information

- See Interactive Code Execution for an explanation of how interactive code execution works in MySQL Shell.
- See Getting Started with MySQL Shell to learn about session and connection alternatives.

20.4.2 Download and Import world_x Database

As part of this quick-start guide, an example schema is provided which is referred to as the `world_x` schema. Many of the examples demonstrate Document Store functionality using this schema. Start your MySQL server so that you can load the `world_x` schema, then follow these steps:

1. Download world_x-db.zip.
2. Extract the installation archive to a temporary location such as `/tmp/`. Unpacking the archive results in a single file named `world_x.sql`.
3. Import the `world_x.sql` file to your server. You can either:
   - Start MySQL Shell in SQL mode and import the file by issuing:

     ```
     mysqlsh -u root --sql --file /tmp/world_x-db/world_x.sql
     Enter password: ****
     ```

   - Set MySQL Shell to SQL mode while it is running and source the schema file by issuing:

     ```
     \sql
     Switching to SQL mode... Commands end with ;
     \source /tmp/world_x-db/world_x.sql
     ```

   Replace `/tmp/` with the path to the `world_x.sql` file on your system. Enter your password if prompted. A non-root account can be used as long as the account has privileges to create new schemas.

The world_x Schema

The `world_x` example schema contains the following JSON collection and relational tables:

- Collection
  - `countryinfo`: Information about countries in the world.
- Tables
  - `country`: Minimal information about countries of the world.
  - `city`: Information about some of the cities in those countries.
  - `countrylanguage`: Languages spoken in each country.

Related Information

- MySQL Shell Sessions explains session types.

20.4.3 Documents and Collections

When you are using MySQL as a Document Store, collections are containers within a schema that you can create, list, and drop. Collections contain JSON documents that you can add, find, update, and remove.

The examples in this section use the `countryinfo` collection in the `world_x` schema. For instructions on setting up the `world_x` schema, see Section 20.4.2, “Download and Import world_x Database”.

Documents

In MySQL, documents are represented as JSON objects. Internally, they are stored in an efficient binary format that enables fast lookups and updates.

- Simple document format for Python:

```
{"field1": "value", "field2" : 10, "field 3": null}
```

An array of documents consists of a set of documents separated by commas and enclosed within `[` and `]` characters.

- Simple array of documents for Python:

```
[{"Name": "Aruba", "Code:": "ABW"}, {"Name": "Angola", "Code:": "AGO"}]
```

MySQL supports the following Python value types in JSON documents:

- numbers (integer and floating point)
- strings
- boolean (False and True)
- None
- arrays of more JSON values
- nested (or embedded) objects of more JSON values

Collections

Collections are containers for documents that share a purpose and possibly share one or more indexes. Each collection has a unique name and exists within a single schema.

The term schema is equivalent to a database, which means a group of database objects as opposed to a relational schema, used to enforce structure and constraints over data. A schema does not enforce conformity on the documents in a collection.

In this quick-start guide:

- Basic objects include:

Object form	Description
`db`	`db` is a global variable assigned to the current active schema. When you want to run operations against the schema, for example to retrieve a collection, you use methods available for the `db` variable.
`db.get_collections()`	db.get_collections() returns a list of collections in the schema. Use the list to get references to collection objects, iterate over them, and so on.

- Basic operations scoped by collections include:

Operation form	Description
`db.name.add()`	The add() method inserts one document or a list of documents into the named collection.
`db.name.find()`	The find() method returns some or all documents in the named collection.
`db.name.modify()`	The modify() method updates documents in the named collection.
`db.name.remove()`	The remove() method deletes one document or a list of documents from the named collection.

Related Information

- See Working with Collections for a general overview.
- CRUD EBNF Definitions provides a complete list of operations.

20.4.3.1 Create, List, and Drop Collections

In MySQL Shell, you can create new collections, get a list of the existing collections in a schema, and remove an existing collection from a schema. Collection names are case-sensitive and each collection name must be unique.

Confirm the Schema

To show the value that is assigned to the schema variable, issue:

```
mysql-py> db
```

If the schema value is not `Schema:world_x`, then set the `db` variable by issuing:

```
mysql-py> \use world_x
```

Create a Collection

To create a new collection in an existing schema, use the `db` object's `createCollection()` method. The following example creates a collection called `flags` in the `world_x` schema.

```
mysql-py> db.create_collection("flags")
```

The method returns a collection object.

```
<Collection:flags>
```

List Collections

To display all collections in the `world_x` schema, use the `db` object's `get_collections()` method. Collections returned by the server you are currently connected to appear between brackets.

```
mysql-py> db.get_collections()
[
    <Collection:countryinfo>,
    <Collection:flags>
]
```

Drop a Collection

To drop an existing collection from a schema, use the `db` object's `drop_collection()` method. For example, to drop the `flags` collection from the current schema, issue:

```
mysql-py> db.drop_collection("flags")
```

The `drop_collection()` method is also used in MySQL Shell to drop a relational table from a schema.

Related Information

- See Collection Objects for more examples.

20.4.3.2 Working with Collections

To work with the collections in a schema, use the `db` global object to access the current schema. In this example we are using the `world_x` schema imported previously, and the `countryinfo` collection. Therefore, the format of the operations you issue is `db.collection_name.operation`, where *`collection_name`* is the name of the collection which the operation is executed against. In the following examples, the operations are executed against the `countryinfo` collection.

Add a Document

Use the `add()` method to insert one document or a list of documents into an existing collection. Insert the following document into the `countryinfo` collection. As this is multi-line content, press **Enter** twice to insert the document.

```
mysql-py> db.countryinfo.add(
 {
    "GNP": .6,
    "IndepYear": 1967,
    "Name": "Sealand",
    "Code:": "SEA",
    "demographics": {
        "LifeExpectancy": 79,
        "Population": 27
    },
    "geography": {
        "Continent": "Europe",
        "Region": "British Islands",
        "SurfaceArea": 193
    },
    "government": {
        "GovernmentForm": "Monarchy",
        "HeadOfState": "Michael Bates"
    }
  }
)
```

The method returns the status of the operation. You can verify the operation by searching for the document. For example:

```
mysql-py> db.countryinfo.find("Name = 'Sealand'")
{
    "GNP": 0.6,
    "_id": "00005e2ff4af00000000000000f4",
    "Name": "Sealand",
    "Code:": "SEA",
    "IndepYear": 1967,
    "geography": {
        "Region": "British Islands",
        "Continent": "Europe",
        "SurfaceArea": 193
    },
    "government": {
        "HeadOfState": "Michael Bates",
        "GovernmentForm": "Monarchy"
    },
    "demographics": {
        "Population": 27,
        "LifeExpectancy": 79
```

```
    }
}
```

Note that in addition to the fields specified when the document was added, there is one more field, the `_id`. Each document requires an identifier field called `_id`. The value of the `_id` field must be unique among all documents in the same collection. In MySQL 8.0.11 and higher, document IDs are generated by the server, not the client, so MySQL Shell does not automatically set an `_id` value. A MySQL server at 8.0.11 or higher sets an `_id` value if the document does not contain the `_id` field. A MySQL server at an earlier 8.0 release or at 5.7 does not set an `_id` value in this situation, so you must specify it explicitly. If you do not, MySQL Shell returns error 5115 `Document is missing a required field`. For more information see Understanding Document IDs.

Related Information

- See CollectionAddFunction for the full syntax definition.
- See Understanding Document IDs.

20.4.3.3 Find Documents

You can use the `find()` method to query for and return documents from a collection in a schema. MySQL Shell provides additional methods to use with the `find()` method to filter and sort the returned documents.

MySQL provides the following operators to specify search conditions: `OR` (`||`), `AND` (`&&`), `XOR`, `IS`, `NOT`, `BETWEEN`, `IN`, `LIKE`, `!=`, `<>`, `>`, `>=`, `<`, `<=`, `&`, `|`, `<<`, `>>`, `+`, `-`, `*`, `/`, `~`, and `%`.

Find All Documents in a Collection

To return all documents in a collection, use the `find()` method without specifying search conditions. For example, the following operation returns all documents in the `countryinfo` collection.

```
mysql-py> db.countryinfo.find()
[
     {
          "GNP": 828,
          "Code:": "ABW",
          "Name": "Aruba",
          "IndepYear": null,
          "geography": {
              "Continent": "North America",
              "Region": "Caribbean",
              "SurfaceArea": 193
          },
          "government": {
              "GovernmentForm": "Nonmetropolitan Territory of The Netherlands",
              "HeadOfState": "Beatrix"
          }
          "demographics": {
              "LifeExpectancy": 78.4000015258789,
              "Population": 103000
          },
          ...
      }
 ]
240 documents in set (0.00 sec)
```

The method produces results that contain operational information in addition to all documents in the collection.

An empty set (no matching documents) returns the following information:

```
Empty set (0.00 sec)
```

Filter Searches

You can include search conditions with the `find()` method. The syntax for expressions that form a search condition is the same as that of traditional MySQL Chapter 12, *Functions and Operators*. You must enclose all expressions in quotes. For the sake of brevity, some of the examples do not display output.

A simple search condition could consist of the `Name` field and a value we know is in a document. The following example returns a single document:

```
mysql-py> db.countryinfo.find("Name = 'Australia'")
[
    {
        "GNP": 351182,
        "Code:": "AUS",
        "Name": "Australia",
        "IndepYear": 1901,
        "geography": {
            "Continent": "Oceania",
            "Region": "Australia and New Zealand",
            "SurfaceArea": 7741220
        },
        "government": {
            "GovernmentForm": "Constitutional Monarchy, Federation",
            "HeadOfState": "Elisabeth II"
        }
        "demographics": {
            "LifeExpectancy": 79.80000305175781,
            "Population": 18886000
        },
    }
]
```

The following example searches for all countries that have a GNP higher than $500 billion. The `countryinfo` collection measures GNP in units of million.

```
mysql-py> db.countryinfo.find("GNP > 500000")
...[output removed]
10 documents in set (0.00 sec)
```

The Population field in the following query is embedded within the demographics object. To access the embedded field, use a period between demographics and Population to identify the relationship. Document and field names are case-sensitive.

```
mysql-py> db.countryinfo.find("GNP > 500000 and demographics.Population < 100000000")
...[output removed]
6 documents in set (0.00 sec)
```

Arithmetic operators in the following expression are used to query for countries with a GNP per capita higher than $30000. Search conditions can include arithmetic operators and most MySQL functions.

Note

Seven documents in the `countryinfo` collection have a population value of zero. Therefore warning messages appear at the end of the output.

```
mysql-py> db.countryinfo.find("GNP*1000000/demographics.Population > 30000")
...[output removed]
9 documents in set, 7 warnings (0.00 sec)
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
```

You can separate a value from the search condition by using the `bind()` method. For example, instead of specifying a hard-coded country name as the condition, substitute a named placeholder

consisting of a colon followed by a name that begins with a letter, such as *country*. Then use the `bind(placeholder, value)` method as follows:

```
mysql-py> db.countryinfo.find("Name = :country").bind("country", "Italy")
{
    "GNP": 1161755,
    "_id": "00005de917d8000000000000006a",
    "Code": "ITA",
    "Name": "Italy",
    "Airports": [],
    "IndepYear": 1861,
    "geography": {
        "Region": "Southern Europe",
        "Continent": "Europe",
        "SurfaceArea": 301316
    },
    "government": {
        "HeadOfState": "Carlo Azeglio Ciampi",
        "GovernmentForm": "Republic"
    },
    "demographics": {
        "Population": 57680000,
        "LifeExpectancy": 79
    }
}
1 document in set (0.01 sec)
```

Tip

Within a program, binding enables you to specify placeholders in your expressions, which are filled in with values before execution and can benefit from automatic escaping, as appropriate.

Always use binding to sanitize input. Avoid introducing values in queries using string concatenation, which can produce invalid input and, in some cases, can cause security issues.

You can use placeholders and the `bind()` method to create saved searches which you can then call with different values. For example to create a saved search for a country:

```
mysql-py> myFind = db.countryinfo.find("Name = :country")
mysql-py> myFind.bind('country', 'France')
{
    "GNP": 1424285,
    "_id": "00005de917d80000000000000048",
    "Code": "FRA",
    "Name": "France",
    "IndepYear": 843,
    "geography": {
        "Region": "Western Europe",
        "Continent": "Europe",
        "SurfaceArea": 551500
    },
    "government": {
        "HeadOfState": "Jacques Chirac",
        "GovernmentForm": "Republic"
    },
    "demographics": {
        "Population": 59225700,
        "LifeExpectancy": 78.80000305175781
    }
}
1 document in set (0.0028 sec)

mysql-py> myFind.bind('country', 'Germany')
{
    "GNP": 2133367,
    "_id": "00005de917d80000000000000038",
    "Code": "DEU",
    "Name": "Germany",
```

```
    "IndepYear": 1955,
    "geography": {
        "Region": "Western Europe",
        "Continent": "Europe",
        "SurfaceArea": 357022
    },
    "government": {
        "HeadOfState": "Johannes Rau",
        "GovernmentForm": "Federal Republic"
    },
    "demographics": {
        "Population": 82164700,
        "LifeExpectancy": 77.4000015258789
    }
}

1 document in set (0.0026 sec)
```

Project Results

You can return specific fields of a document, instead of returning all the fields. The following example returns the GNP and Name fields of all documents in the `countryinfo` collection matching the search conditions.

Use the `fields()` method to pass the list of fields to return.

```
mysql-py> db.countryinfo.find("GNP > 5000000").fields(["GNP", "Name"])
[
    {
        "GNP": 8510700,
        "Name": "United States"
    }
]
1 document in set (0.00 sec)
```

In addition, you can alter the returned documents—adding, renaming, nesting and even computing new field values—with an expression that describes the document to return. For example, alter the names of the fields with the following expression to return only two documents.

```
mysql-py> db.countryinfo.find().fields(
mysqlx.expr('{"Name": upper(Name), "GNPPerCapita": GNP*1000000/demographics.Population}')).limit(2)
{
    "Name": "ARUBA",
    "GNPPerCapita": 8038.834951456311
}
{
    "Name": "AFGHANISTAN",
    "GNPPerCapita": 263.0281690140845
}
```

Limit, Sort, and Skip Results

You can apply the `limit()`, `sort()`, and `skip()` methods to manage the number and order of documents returned by the `find()` method.

To specify the number of documents included in a result set, append the `limit()` method with a value to the `find()` method. The following query returns the first five documents in the `countryinfo` collection.

```
mysql-py> db.countryinfo.find().limit(5)
... [output removed]
5 documents in set (0.00 sec)
```

To specify an order for the results, append the `sort()` method to the `find()` method. Pass to the `sort()` method a list of one or more fields to sort by and, optionally, the descending (`desc`) or ascending (`asc`) attribute as appropriate. Ascending order is the default order type.

For example, the following query sorts all documents by the IndepYear field and then returns the first eight documents in descending order.

```
mysql-py> db.countryinfo.find().sort(["IndepYear desc"]).limit(8)
... [output removed]
8 documents in set (0.00 sec)
```

By default, the `limit()` method starts from the first document in the collection. You can use the `skip()` method to change the starting document. For example, to ignore the first document and return the next eight documents matching the condition, pass to the `skip()` method a value of 1.

```
mysql-py> db.countryinfo.find().sort(["IndepYear desc"]).limit(8).skip(1)
... [output removed]
8 documents in set (0.00 sec)
```

Related Information

- The MySQL Reference Manual provides detailed documentation on functions and operators.
- See CollectionFindFunction for the full syntax definition.

20.4.3.4 Modify Documents

You can use the `modify()` method to update one or more documents in a collection. The X DevAPI provides additional methods for use with the `modify()` method to:

- Set and unset fields within documents.
- Append, insert, and delete arrays.
- Bind, limit, and sort the documents to be modified.

Set and Unset Document Fields

The `modify()` method works by filtering a collection to include only the documents to be modified and then applying the operations that you specify to those documents.

In the following example, the `modify()` method uses the search condition to identify the document to change and then the `set()` method replaces two values within the nested demographics object.

```
mysql-py> db.countryinfo.modify("Code = 'SEA'").set(
"demographics", {"LifeExpectancy": 78, "Population": 28})
```

After you modify a document, use the `find()` method to verify the change.

To remove content from a document, use the `modify()` and `unset()` methods. For example, the following query removes the GNP from a document that matches the search condition.

```
mysql-py> db.countryinfo.modify("Name = 'Sealand'").unset("GNP")
```

Use the `find()` method to verify the change.

```
mysql-py> db.countryinfo.find("Name = 'Sealand'")
{
    "_id": "00005e2ff4af00000000000000f4",
    "Name": "Sealand",
    "Code:": "SEA",
    "IndepYear": 1967,
    "geography": {
        "Region": "British Islands",
        "Continent": "Europe",
        "SurfaceArea": 193
    },
    "government": {
        "HeadOfState": "Michael Bates",
        "GovernmentForm": "Monarchy"
    },
```

```
    "demographics": {
        "Population": 27,
        "LifeExpectancy": 79
    }
}
```

Append, Insert, and Delete Arrays

To append an element to an array field, or insert, or delete elements in an array, use the `array_append()`, `array_insert()`, or `array_delete()` methods. The following examples modify the `countryinfo` collection to enable tracking of international airports.

The first example uses the `modify()` and `set()` methods to create a new Airports field in all documents.

Caution

Use care when you modify documents without specifying a search condition; doing so modifies all documents in the collection.

```
mysql-py> db.countryinfo.modify("true").set("Airports", [])
```

With the Airports field added, the next example uses the `array_append()` method to add a new airport to one of the documents. *$.Airports* in the following example represents the Airports field of the current document.

```
mysql-py> db.countryinfo.modify("Name = 'France'").array_append("$.Airports", "ORY")
```

Use `find()` to see the change.

```
mysql-py> db.countryinfo.find("Name = 'France'")
{
    "GNP": 1424285,
    "_id": "00005de917d80000000000000048",
    "Code": "FRA",
    "Name": "France",
    "Airports": [
        "ORY"
    ],
    "IndepYear": 843,
    "geography": {
        "Region": "Western Europe",
        "Continent": "Europe",
        "SurfaceArea": 551500
    },
    "government": {
        "HeadOfState": "Jacques Chirac",
        "GovernmentForm": "Republic"
    },
    "demographics": {
        "Population": 59225700,
        "LifeExpectancy": 78.80000305175781
    }
}
```

To insert an element at a different position in the array, use the `array_insert()` method to specify which index to insert in the path expression. In this case, the index is 0, or the first element in the array.

```
mysql-py> db.countryinfo.modify("Name = 'France'").array_insert("$.Airports[0]", "CDG")
```

To delete an element from the array, you must pass to the `array_delete()` method the index of the element to be deleted.

```
mysql-py> db.countryinfo.modify("Name = 'France'").array_delete("$.Airports[1]")
```

Related Information

- The MySQL Reference Manual provides instructions to help you search for and modify JSON values.

- See CollectionModifyFunction for the full syntax definition.

20.4.3.5 Remove Documents

You can use the `remove()` method to delete some or all documents from a collection in a schema. The X DevAPI provides additional methods for use with the `remove()` method to filter and sort the documents to be removed.

Remove Documents Using Conditions

The following example passes a search condition to the `remove()` method. All documents matching the condition are removed from the `countryinfo` collection. In this example, one document matches the condition.

```
mysql-py> db.countryinfo.remove("Code = 'SEA'")
```

Remove the First Document

To remove the first document in the `countryinfo` collection, use the `limit()` method with a value of 1.

```
mysql-py> db.countryinfo.remove("true").limit(1)
```

Remove the Last Document in an Order

The following example removes the last document in the `countryinfo` collection by country name.

```
mysql-py> db.countryinfo.remove("true").sort(["Name desc"]).limit(1)
```

Remove All Documents in a Collection

You can remove all documents in a collection. To do so, use the `remove("true")` method without specifying a search condition.

Caution

Use care when you remove documents without specifying a search condition. This action deletes all documents from the collection.

Alternatively, use the `db.drop_collection('countryinfo')` operation to delete the `countryinfo` collection.

Related Information

- See CollectionRemoveFunction for the full syntax definition.
- See Section 20.4.2, “Download and Import world_x Database” for instructions to recreate the `world_x` schema.

20.4.3.6 Create and Drop Indexes

Indexes are used to find documents with specific field values quickly. Without an index, MySQL must begin with the first document and then read through the entire collection to find the relevant fields. The larger the collection, the more this costs. If a collection is large and queries on a specific field are common, then consider creating an index on a specific field inside a document.

For example, the following query performs better with an index on the Population field:

```
mysql-py> db.countryinfo.find("demographics.Population < 100")
...[output removed]
8 documents in set (0.00 sec)
```

The `create_index()` method creates an index that you can define with a JSON document that specifies which fields to use. This section is a high level overview of indexing. For more information see Indexing Collections.

Add a Nonunique Index

To create a nonunique index, pass an index name and the index information to the `create_index()` method. Duplicate index names are prohibited.

The following example specifies an index named `popul`, defined against the `Population` field from the `demographics` object, indexed as an `Integer` numeric value. The final parameter indicates whether the field should require the `NOT NULL` constraint. If the value is `false`, the field can contain `NULL` values. The index information is a JSON document with details of one or more fields to include in the index. Each field definition must include the full document path to the field, and specify the type of the field.

```
mysql-py> db.countryinfo.createIndex("popul", {fields:
[{field: '$.demographics.Population', type: 'INTEGER'}]})
```

Here, the index is created using an integer numeric value. Further options are available, including options for use with GeoJSON data. You can also specify the type of index, which has been omitted here because the default type “index” is appropriate.

Add a Unique Index

To create a unique index, pass an index name, the index definition, and the index type “unique” to the `create_index()` method. This example shows a unique index created on the country name (`"Name"`), which is another common field in the `countryinfo` collection to index. In the index field description, `"TEXT(40)"` represents the number of characters to index, and `"required": True` specifies that the field is required to exist in the document.

```
mysql-py> db.countryinfo.create_index("name",
{"fields": [{"field": "$.Name", "type": "TEXT(40)", "required": True}], "unique": True})
```

Drop an Index

To drop an index, pass the name of the index to drop to the `drop_index()` method. For example, you can drop the “popul” index as follows:

```
mysql-py> db.countryinfo.drop_index("popul")
```

Related Information

- See Indexing Collections for more information.
- See Defining an Index for more information on the JSON document that defines an index.
- See Collection Index Management Functions for the full syntax definition.

20.4.4 Relational Tables

You can also use X DevAPI to work with relational tables. In MySQL, each relational table is associated with a particular storage engine. The examples in this section use `InnoDB` tables in the `world_x` schema.

Confirm the Schema

To show the schema that is assigned to the `db` global variable, issue `db`.

```
mysql-py> db
<Schema:world_x>
```

If the returned value is not `Schema:world_x`, set the `db` variable as follows:

```
mysql-py> \use world_x
Schema `world_x` accessible through db.
```

Show All Tables

To display all relational tables in the `world_x` schema, use the `get_tables()` method on the `db` object.

```
mysql-py> db.get_tables()
[
    <Table:city>,
    <Table:country>,
    <Table:countrylanguage>
]
```

Basic Table Operations

Basic operations scoped by tables include:

Operation form	Description
`db.name.insert()`	The insert() method inserts one or more records into the named table.
`db.name.select()`	The select() method returns some or all records in the named table.
`db.name.update()`	The update() method updates records in the named table.
`db.name.delete()`	The delete() method deletes one or more records from the named table.

Related Information

- See Working with Relational Tables for more information.
- CRUD EBNF Definitions provides a complete list of operations.
- See Section 20.4.2, “Download and Import world_x Database” for instructions on setting up the `world_x` schema sample.

20.4.4.1 Insert Records into Tables

You can use the `insert()` method with the `values()` method to insert records into an existing relational table. The `insert()` method accepts individual columns or all columns in the table. Use one or more `values()` methods to specify the values to be inserted.

Insert a Complete Record

To insert a complete record, pass to the `insert()` method all columns in the table. Then pass to the `values()` method one value for each column. For example, to add a new record to the city table in the `world_x` database, insert the following record and press **Enter** twice.

```
mysql-py> db.city.insert("ID", "Name", "CountryCode", "District", "Info").values(
None, "Olympia", "USA", "Washington", '{"Population": 5000}')
```

The city table has five columns: ID, Name, CountryCode, District, and Info. Each value must match the data type of the column it represents.

Insert a Partial Record

The following example inserts values into the ID, Name, and CountryCode columns of the city table.

```
mysql-py> db.city.insert("ID", "Name", "CountryCode").values(
```

```
None, "Little Falls", "USA").values(None, "Happy Valley", "USA")
```

When you specify columns using the `insert()` method, the number of values must match the number of columns. In the previous example, you must supply three values to match the three columns specified.

Related Information

- See TableInsertFunction for the full syntax definition.

20.4.4.2 Select Tables

You can use the `select()` method to query for and return records from a table in a database. The X DevAPI provides additional methods to use with the `select()` method to filter and sort the returned records.

MySQL provides the following operators to specify search conditions: `OR` (`||`), `AND` (`&&`), `XOR`, `IS`, `NOT`, `BETWEEN`, `IN`, `LIKE`, `!=`, `<>`, `>`, `>=`, `<`, `<=`, `&`, `|`, `<<`, `>>`, `+`, `-`, `*`, `/`, `~`, and `%`.

Select All Records

To issue a query that returns all records from an existing table, use the `select()` method without specifying search conditions. The following example selects all records from the city table in the `world_x` database.

Note

Limit the use of the empty `select()` method to interactive statements. Always use explicit column-name selections in your application code.

```
mysql-py> db.city.select()
+------+------------+-------------+------------+-------------------------+
| ID   | Name       | CountryCode | District   | Info                    |
+------+------------+-------------+------------+-------------------------+
|    1 | Kabul      | AFG         | Kabol      |{"Population": 1780000}  |
|    2 | Qandahar   | AFG         | Qandahar   |{"Population": 237500}   |
|    3 | Herat      | AFG         | Herat      |{"Population": 186800}   |
...    ...          ...           ...          ...
| 4079 | Rafah      | PSE         | Rafah      |{"Population": 92020}    |
+------+------- ----+-------------+------------+-------------------------+
4082 rows in set (0.01 sec)
```

An empty set (no matching records) returns the following information:

```
Empty set (0.00 sec)
```

Filter Searches

To issue a query that returns a set of table columns, use the `select()` method and specify the columns to return between square brackets. This query returns the Name and CountryCode columns from the city table.

```
mysql-py> db.city.select(["Name", "CountryCode"])
+-------------------+-------------+
| Name              | CountryCode |
+-------------------+-------------+
| Kabul             | AFG         |
| Qandahar          | AFG         |
| Herat             | AFG         |
| Mazar-e-Sharif    | AFG         |
| Amsterdam         | NLD         |
...                 ...
| Rafah             | PSE         |
| Olympia           | USA         |
| Little Falls      | USA         |
```

```
| Happy Valley      | USA         |
+-------------------+-------------+
4082 rows in set (0.00 sec)
```

To issue a query that returns rows matching specific search conditions, use the `where()` method to include those conditions. For example, the following example returns the names and country codes of the cities that start with the letter Z.

```
mysql-py> db.city.select(["Name", "CountryCode"]).where("Name like 'Z%'")
+-------------------+-------------+
| Name              | CountryCode |
+-------------------+-------------+
| Zaanstad          | NLD         |
| Zoetermeer        | NLD         |
| Zwolle            | NLD         |
| Zenica            | BIH         |
| Zagazig           | EGY         |
| Zaragoza          | ESP         |
| Zamboanga         | PHL         |
| Zahedan           | IRN         |
| Zanjan            | IRN         |
| Zabol             | IRN         |
| Zama              | JPN         |
| Zhezqazghan       | KAZ         |
| Zhengzhou         | CHN         |
...                 ...
| Zeleznogorsk      | RUS         |
+-------------------+-------------+
59 rows in set (0.00 sec)
```

You can separate a value from the search condition by using the `bind()` method. For example, instead of using "Name = 'Z%' " as the condition, substitute a named placeholder consisting of a colon followed by a name that begins with a letter, such as *name*. Then include the placeholder and value in the `bind()` method as follows:

```
mysql-py> db.city.select(["Name", "CountryCode"]).where(
"Name like :name").bind("name", "Z%")
```

Tip

Within a program, binding enables you to specify placeholders in your expressions, which are filled in with values before execution and can benefit from automatic escaping, as appropriate.

Always use binding to sanitize input. Avoid introducing values in queries using string concatenation, which can produce invalid input and, in some cases, can cause security issues.

Project Results

To issue a query using the `AND` operator, add the operator between search conditions in the `where()` method.

```
mysql-py> db.city.select(["Name", "CountryCode"]).where(
"Name like 'Z%' and CountryCode = 'CHN'")
+----------------+-------------+
| Name           | CountryCode |
+----------------+-------------+
| Zhengzhou      | CHN         |
| Zibo           | CHN         |
| Zhangjiakou    | CHN         |
| Zhuzhou        | CHN         |
| Zhangjiang     | CHN         |
| Zigong         | CHN         |
| Zaozhuang      | CHN         |
...              ...
| Zhangjiagang   | CHN         |
+----------------+-------------+
```

```
22 rows in set (0.01 sec)
```

To specify multiple conditional operators, you can enclose the search conditions in parenthesis to change the operator precedence. The following example demonstrates the placement of `AND` and `OR` operators.

```
mysql-py> db.city.select(["Name", "CountryCode"]).where(
"Name like 'Z%' and (CountryCode = 'CHN' or CountryCode = 'RUS')")
+-------------------+-------------+
| Name              | CountryCode |
+-------------------+-------------+
| Zhengzhou         | CHN         |
| Zibo              | CHN         |
| Zhangjiakou       | CHN         |
| Zhuzhou           | CHN         |
...                 ...
| Zeleznogorsk      | RUS         |
+-------------------+-------------+
29 rows in set (0.01 sec)
```

Limit, Order, and Offset Results

You can apply the `limit()`, `order_by()`, and `offset()` methods to manage the number and order of records returned by the `select()` method.

To specify the number of records included in a result set, append the `limit()` method with a value to the `select()` method. For example, the following query returns the first five records in the country table.

```
mysql-py> db.country.select(["Code", "Name"]).limit(5)
+------+-------------+
| Code | Name        |
+------+-------------+
| ABW  | Aruba       |
| AFG  | Afghanistan |
| AGO  | Angola      |
| AIA  | Anguilla    |
| ALB  | Albania     |
+------+-------------+
5 rows in set (0.00 sec)
```

To specify an order for the results, append the `order_by()` method to the `select()` method. Pass to the `order_by()` method a list of one or more columns to sort by and, optionally, the descending (`desc`) or ascending (`asc`) attribute as appropriate. Ascending order is the default order type.

For example, the following query sorts all records by the Name column and then returns the first three records in descending order .

```
mysql-py> db.country.select(["Code", "Name"]).order_by(["Name desc"]).limit(3)
+------+------------+
| Code | Name       |
+------+------------+
| ZWE  | Zimbabwe   |
| ZMB  | Zambia     |
| YUG  | Yugoslavia |
+------+------------+
3 rows in set (0.00 sec)
```

By default, the `limit()` method starts from the first record in the table. You can use the `offset()` method to change the starting record. For example, to ignore the first record and return the next three records matching the condition, pass to the `offset()` method a value of 1.

```
mysql-py> db.country.select(["Code", "Name"]).order_by(["Name desc"]).limit(3).offset(1)
+------+------------+
| Code | Name       |
+------+------------+
| ZMB  | Zambia     |
| YUG  | Yugoslavia |
```

```
| YEM  | Yemen      |
+------+------------+
3 rows in set (0.00 sec)
```

Related Information

- The MySQL Reference Manual provides detailed documentation on functions and operators.
- See TableSelectFunction for the full syntax definition.

20.4.4.3 Update Tables

You can use the `update()` method to modify one or more records in a table. The `update()` method works by filtering a query to include only the records to be updated and then applying the operations you specify to those records.

To replace a city name in the city table, pass to the `set()` method the new city name. Then, pass to the `where()` method the city name to locate and replace. The following example replaces the city Peking with Beijing.

```
mysql-py> db.city.update().set("Name", "Beijing").where("Name = 'Peking'")
```

Use the `select()` method to verify the change.

```
mysql-py> db.city.select(["ID", "Name", "CountryCode", "District", "Info"]).where("Name = 'Beijing'")
+------+-----------+-------------+----------+----------------------------+
| ID   | Name      | CountryCode | District | Info                       |
+------+-----------+-------------+----------+----------------------------+
| 1891 | Beijing   | CHN         | Peking   | {"Population": 7472000}    |
+------+-----------+-------------+----------+----------------------------+
1 row in set (0.00 sec)
```

Related Information

- See TableUpdateFunction for the full syntax definition.

20.4.4.4 Delete Tables

You can use the `delete()` method to remove some or all records from a table in a database. The X DevAPI provides additional methods to use with the `delete()` method to filter and order the records to be deleted.

Delete Records Using Conditions

The example that follows passes search conditions to the `delete()` method. All records matching the condition are deleted from the `city` table. In this example, one record matches the condition.

```
mysql-py> db.city.delete().where("Name = 'Olympia'")
```

Delete the First Record

To delete the first record in the city table, use the `limit()` method with a value of 1.

```
mysql-py> db.city.delete().limit(1)
```

Delete All Records in a Table

You can delete all records in a table. To do so, use the `delete()` method without specifying a search condition.

Caution

Use care when you delete records without specifying a search condition; doing so deletes all records from the table.

Drop a Table

The `drop_collection()` method is also used in MySQL Shell to drop a relational table from a database. For example, to drop the `citytest` table from the `world_x` database, issue:

```
mysql-py> db.drop_collection("citytest")
```

Related Information

- See TableDeleteFunction for the full syntax definition.
- See Section 20.4.2, “Download and Import world_x Database” for instructions to recreate the `world_x` database.

20.4.5 Documents in Tables

In MySQL, a table may contain traditional relational data, JSON values, or both. You can combine traditional data with JSON documents by storing the documents in columns having a native `JSON` data type.

Examples in this section use the city table in the `world_x` schema.

city Table Description

The city table has five columns (or fields).

```
+---------------+------------+-------+-------+---------+------------------+
| Field         | Type       | Null  | Key   | Default | Extra            |
+---------------+------------+-------+-------+---------+------------------+
| ID            | int(11)    | NO    | PRI   | null    | auto_increment   |
| Name          | char(35)   | NO    |       |         |                  |
| CountryCode   | char(3)    | NO    |       |         |                  |
| District      | char(20)   | NO    |       |         |                  |
| Info          | json       | YES   |       | null    |                  |
+---------------+------------+-------+-------+---------+------------------+
```

Insert a Record

To insert a document into the column of a table, pass to the `values()` method a well-formed JSON document in the correct order. In the following example, a document is passed as the final value to be inserted into the Info column.

```
mysql-py> db.city.insert().values(
None, "San Francisco", "USA", "California", '{"Population":830000}')
```

Select a Record

You can issue a query with a search condition that evaluates document values in the expression.

```
mysql-py> db.city.select(["ID", "Name", "CountryCode", "District", "Info"]).where(
"CountryCode = :country and Info->'$.Population' > 1000000").bind(
'country', 'USA')
+------+---------------+-------------+---------------+-----------------------------+
| ID   | Name          | CountryCode | District      | Info                        |
+------+---------------+-------------+---------------+-----------------------------+
| 3793 | New York      | USA         | New York      | {"Population": 8008278}     |
| 3794 | Los Angeles   | USA         | California    | {"Population": 3694820}     |
| 3795 | Chicago       | USA         | Illinois      | {"Population": 2896016}     |
| 3796 | Houston       | USA         | Texas         | {"Population": 1953631}     |
| 3797 | Philadelphia  | USA         | Pennsylvania  | {"Population": 1517550}     |
| 3798 | Phoenix       | USA         | Arizona       | {"Population": 1321045}     |
| 3799 | San Diego     | USA         | California    | {"Population": 1223400}     |
| 3800 | Dallas        | USA         | Texas         | {"Population": 1188580}     |
| 3801 | San Antonio   | USA         | Texas         | {"Population": 1144646}     |
+------+---------------+-------------+---------------+-----------------------------+
9 rows in set (0.01 sec)
```

Related Information

- See Working with Relational Tables and Documents for more information.
- See Section 11.5, “The JSON Data Type” for a detailed description of the data type.

20.5 X Plugin

This section explains how to use, configure and monitor X Plugin.

20.5.1 Checking X Plugin Installation

X Plugin is enabled by default in MySQL 8, therefore installing or upgrading to MySQL 8 makes the plugin available. You can verify X Plugin is installed on an instance of MySQL server by using the `SHOW plugins` statement to view the plugins list.

To use MySQL Shell to verify X Plugin is installed, issue:

```
$> mysqlsh -u user --sqlc -P 3306 -e "SHOW plugins"
```

To use MySQL Client to verify X Plugin is installed, issue:

```
$> mysql -u user -p -e "SHOW plugins"
```

An example result if X Plugin is installed is highlighted here:

```
+----------------------------+----------+--------------------+---------+---------+
| Name                       | Status   | Type               | Library | License |
+----------------------------+----------+--------------------+---------+---------+

...


| mysqlx                     | ACTIVE   | DAEMON             | NULL    | GPL     |

...

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

20.5.2 Disabling X Plugin

The X Plugin can be disabled at startup by either setting `mysqlx=0` in your MySQL configuration file, or by passing in either `--mysqlx=0` or `--skip-mysqlx` when starting the MySQL server.

Alternatively, use the `-DWITH_MYSQLX=OFF` CMake option to compile MySQL Server without X Plugin.

20.5.3 Using Encrypted Connections with X Plugin

This section explains how to configure X Plugin to use encrypted connections. For more background information, see Section 6.3, “Using Encrypted Connections”.

To enable configuring support for encrypted connections, X Plugin has `mysqlx_ssl_xxx` system variables, which can have different values from the `ssl_xxx` system variables used with MySQL Server. For example, X Plugin can have SSL key, certificate, and certificate authority files that differ from those used for MySQL Server. These variables are described at Section 20.5.6.2, “X Plugin Options and System Variables”. Similarly, X Plugin has its own `Mysqlx_ssl_xxx` status variables that correspond to the MySQL Server encrypted-connection `Ssl_xxx` status variables. See Section 20.5.6.3, “X Plugin Status Variables”.

At initialization, X Plugin determines its TLS context for encrypted connections as follows:

- If all `mysqlx_ssl_xxx` system variables have their default values, X Plugin uses the same TLS context as the MySQL Server main connection interface, which is determined by the values of the `ssl_xxx` system variables.

- If any `mysqlx_ssl_xxx` variable has a nondefault value, X Plugin uses the TLS context defined by the values of its own system variables. (This is the case if any `mysqlx_ssl_xxx` system variable is set to a value different from its default.)

This means that, on a server with X Plugin enabled, you can choose to have MySQL Protocol and X Protocol connections share the same encryption configuration by setting only the `ssl_xxx` variables, or have separate encryption configurations for MySQL Protocol and X Protocol connections by configuring the `ssl_xxx` and `mysqlx_ssl_xxx` variables separately.

To have MySQL Protocol and X Protocol connections use the same encryption configuration, set only the `ssl_xxx` system variables in `my.cnf`:

```
[mysqld]
ssl_ca=ca.pem
ssl_cert=server-cert.pem
ssl_key=server-key.pem
```

To configure encryption separately for MySQL Protocol and X Protocol connections, set both the `ssl_xxx` and `mysqlx_ssl_xxx` system variables in `my.cnf`:

```
[mysqld]
ssl_ca=ca1.pem
ssl_cert=server-cert1.pem
ssl_key=server-key1.pem

mysqlx_ssl_ca=ca2.pem
mysqlx_ssl_cert=server-cert2.pem
mysqlx_ssl_key=server-key2.pem
```

For general information about configuring connection-encryption support, see Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”. That discussion is written for MySQL Server, but the parameter names are similar for X Plugin. (The X Plugin `mysqlx_ssl_xxx` system variable names correspond to the MySQL Server `ssl_xxx` system variable names.)

The `tls_version` system variable that determines the permitted TLS versions for MySQL Protocol connections also applies to X Protocol connections. The permitted TLS versions for both types of connections are therefore the same.

Encryption per connection is optional, but a specific user can be required to use encryption for X Protocol and MySQL Protocol connections by including an appropriate `REQUIRE` clause in the `CREATE USER` statement that creates the user. For details, see Section 13.7.1.3, “CREATE USER Statement”. Alternatively, to require all users to use encryption for X Protocol and MySQL Protocol connections, enable the `require_secure_transport` system variable. For additional information, see Configuring Encrypted Connections as Mandatory.

20.5.4 Using X Plugin with the Caching SHA-2 Authentication Plugin

X Plugin supports MySQL user accounts created with the `caching_sha2_password` authentication plugin. For more information on this plugin, see Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”. You can use X Plugin to authenticate against such accounts using non-SSL connections with `SHA256_MEMORY` authentication and SSL connections with `PLAIN` authentication.

Although the `caching_sha2_password` authentication plugin holds an authentication cache, this cache is not shared with X Plugin, so X Plugin uses its own authentication cache for `SHA256_MEMORY` authentication. The X Plugin authentication cache stores hashes of user account passwords, and cannot be accessed using SQL. If a user account is modified or removed, the relevant entries are removed from the cache. The X Plugin authentication cache is maintained by the `mysqlx_cache_cleaner` plugin, which is enabled by default, and has no related system variables or status variables.

Before you can use non-SSL X Protocol connections to authenticate an account that uses the `caching_sha2_password` authentication plugin, the account must have authenticated at least once

over an X Protocol connection with SSL, to supply the password to the X Plugin authentication cache. Once this initial authentication over SSL has succeeded, non-SSL X Protocol connections can be used.

It is possible to disable the `mysqlx_cache_cleaner` plugin by starting the MySQL server with the option `--mysqlx_cache_cleaner=0`. If you do this, the X Plugin authentication cache is disabled, and therefore SSL must always be used for X Protocol connections when authenticating with `SHA256_MEMORY` authentication.

20.5.5 Connection Compression with X Plugin

From MySQL 8.0.19, X Plugin supports compression of messages sent over X Protocol connections. Connections can be compressed if the server and the client agree on a mutually supported compression algorithm. Enabling compression reduces the number of bytes sent over the network, but adds to the server and client an additional CPU cost for compression and decompression operations. The benefits of compression therefore occur primarily when there is low network bandwidth, network transfer time dominates the cost of compression and decompression operations, and result sets are large.

Note

Different MySQL clients implement support for connection compression differently; consult your client documentation for details. For example, for classic MySQL protocol connections, see Section 4.2.8, “Connection Compression Control”.

- Configuring Connection Compression for X Plugin
- Compressed Connection Characteristics for X Plugin
- Monitoring Connection Compression for X Plugin

Configuring Connection Compression for X Plugin

By default, X Plugin supports the zstd, LZ4, and Deflate compression algorithms. Compression with the Deflate algorithm is carried out using the zlib software library, so the `deflate_stream` compression algorithm setting for X Protocol connections is equivalent to the `zlib` setting for classic MySQL protocol connections.

On the server side, you can disallow any of the compression algorithms by setting the `mysqlx_compression_algorithms` system variable to include only those permitted. The algorithm names `zstd_stream`, `lz4_message`, and `deflate_stream` can be specified in any combination, and the order and lettercase are not important. If the system variable value is the empty string, no compression algorithms are permitted and connections are uncompressed.

The following table compares the characteristics of the different compression algorithms and shows their assigned priorities. By default, the server chooses the highest-priority algorithm permitted in common by the server and the client; clients may change the priorities as described later. The short form alias for the algorithms can be used by clients when specifying them.

Table 20.1 X Protocol Compression Algorithm Characteristics

Algorithm	Alias	Compression Ratio	Throughput	CPU Cost	Default Priority
`zsth_stream`	`zstd`	High	High	Medium	First
`lz4_message`	`lz4`	Low	High	Lowest	Second
`deflate_stream`	`deflate`	High	Low	Highest	Third

The X Protocol set of permitted compression algorithms (whether user-specified or default) is independent of the set of compression algorithms permitted by MySQL Server for classic MySQL protocol connections, which is specified by the `protocol_compression_algorithms` server

system variable. If you do not specify the `mysqlx_compression_algorithms` system variable, X Plugin does not fall back to using compression settings for classic MySQL protocol connections. Instead, its default is to permit all algorithms shown in Table 20.1, "X Protocol Compression Algorithm Characteristics". This is unlike the situation for the TLS context, where MySQL Server settings are used if the X Plugin system variables are not set, as described in Section 20.5.3, "Using Encrypted Connections with X Plugin". For information about compression for classic MySQL protocol connections, see Section 4.2.8, "Connection Compression Control".

On the client side, an X Protocol connection request can specify several parameters for compression control:

- The compression mode.
- The compression level (from MySQL 8.0.20).
- The list of permitted compression algorithms in priority order (from MySQL 8.0.22).

Note

Some clients or Connectors might not support a given compression-control feature. For example, specifying compression level for X Protocol connections is supported only by MySQL Shell, not by other MySQL clients or Connectors. See the documentation for specific products for details about supported features and how to use them.

The connection mode has these permitted values:

- `disabled`: The connection is uncompressed.
- `preferred`: The server and client negotiate to find a compression algorithm they both permit. If no common algorithm is available, the connection is uncompressed. This is the default mode if not specified explicitly.
- `required`: Compression algorithm negotiation occurs as for `preferred` mode, but if no common algorithm is available, the connection request terminates with an error.

In addition to agreeing on a compression algorithm for each connection, the server and client can agree on a compression level from the numeric range that applies to the agreed algorithm. As the compression level for an algorithm increases, the data compression ratio increases, which reduces the network bandwidth and transfer time needed to send the message to the client. However, the effort required for data compression also increases, taking up time and CPU and memory resources on the server. Increases in the compression effort do not have a linear relationship to increases in the compression ratio.

In MySQL 8.0.19, X Plugin always uses the library default compression level for each algorithm (3 for zstd, 0 for LZ4, and 6 for Deflate), and the client cannot negotiate this. From MySQL 8.0.20, the client can request a specific compression level during capability negotiations with the server for an X Protocol connection.

The default compression levels used by X Plugin from MySQL 8.0.20 have been selected through performance testing as being a good trade-off between compression time and network transit time. These defaults are not necessarily the same as the library default for each algorithm. They apply if the client does not request a compression level for the algorithm. The default compression levels are initially set to 3 for zstd, 2 for LZ4, and 3 for Deflate. You can adjust these settings using the `mysqlx_zstd_default_compression_level`, `mysqlx_lz4_default_compression_level`, and `mysqlx_deflate_default_compression_level` system variables.

To prevent excessive resource consumption on the server, X Plugin sets a maximum compression level that the server permits for each algorithm. If a client requests a compression level that exceeds this setting, the server uses its maximum permitted compression level (compression level requests by a client are supported only by MySQL Shell). The maximum

compression levels are initially set to 11 for zstd, 8 for LZ4, and 5 for Deflate. You can adjust these settings using the `mysqlx_zstd_max_client_compression_level`, `mysqlx_lz4_max_client_compression_level`, and `mysqlx_deflate_max_client_compression_level` system variables.

If the server and client permit more than one algorithm in common, the default priority order for choosing an algorithm during negotiation is shown in Table 20.1, "X Protocol Compression Algorithm Characteristics". From MySQL 8.0.22, for clients that support specifying compression algorithms, the connection request can include a list of algorithms permitted by the client, specified using the algorithm name or its alias. The order of these algorithms in the list is taken as a priority order by the server. The algorithm used in this case is the first of those in the client list that is also permitted on the server side. However, the option for compression algorithms is subject to the compression mode:

- If the compression mode is `disabled`, the compression algorithms option is ignored.
- If the compression mode is `preferred` but no algorithm permitted on the client side is permitted on the server side, the connection is uncompressed.
- If the compression mode is `required` but no algorithm permitted on the client side is permitted on the server side, an error occurs.

To monitor the effects of message compression, use the X Plugin status variables described in Monitoring Connection Compression for X Plugin. You can use these status variables to calculate the benefit of message compression with your current settings, and use that information to tune your settings.

Compressed Connection Characteristics for X Plugin

X Protocol connection compression operates with the following behaviors and boundaries:

- The `_stream` and `_message` suffixes in algorithm names refer to two different operational modes: In stream mode, all X Protocol messages in a single connection are compressed into a continuous stream and must be decompressed in the same manner—following the order they were compressed and without skipping any messages. In message mode, each message is compressed individually and independently, and need not be decompressed in the order in which they were compressed. Also, message mode does not require all compressed messages to be decompressed.
- Compression is not applied to any messages that are sent before authentication succeeds.
- Compression is not applied to control flow messages such as `Mysqlx.Ok`, `Mysqlx.Error`, and `Mysqlx.Sql.StmtExecuteOk` messages.
- All other X Protocol messages can be compressed if the server and client agree on a mutually permitted compression algorithm during capability negotiation. If the client does not request compression at that stage, neither the client nor the server applies compression to messages.
- When messages sent over X Protocol connections are compressed, the limit specified by the `mysqlx_max_allowed_packet` system variable still applies. The network packet must be smaller than this limit after the message payload has been decompressed. If the limit is exceeded, X Plugin returns a decompression error and closes the connection.
- The following points pertain to compression level requests by clients, which is supported only by MySQL Shell:
  - Compression levels must be specified by the client as an integer. If any other type of value is supplied, the connection closes with an error.
  - If a client specifies an algorithm but not a compression level, the server uses its default compression level for the algorithm.
  - If a client requests an algorithm compression level that exceeds the server maximum permitted level, the server uses the maximum permitted level.

- If a client requests an algorithm compression level that is less than the server minimum permitted level, the server uses the minimum permitted level.

Monitoring Connection Compression for X Plugin

You can monitor the effects of message compression using the X Plugin status variables. When message compression is in use, the session `Mysqlx_compression_algorithm` status variable shows which compression algorithm is in use for the current X Protocol connection, and `Mysqlx_compression_level` shows the compression level that was selected. These session status variables are available from MySQL 8.0.20.

From MySQL 8.0.19, X Plugin status variables can be used to calculate the efficiency of the compression algorithms that are selected (the data compression ratio), and the overall effect of using message compression. Use the session value of the status variables in the following calculations to see what the benefit of message compression was for a specific session with a known compression algorithm. Or use the global value of the status variables to check the overall benefit of message compression for your server across all sessions using X Protocol connections, including all the compression algorithms that have been used for those sessions, and all sessions that did not use message compression. You can then tune message compression by adjusting the permitted compression algorithms, maximum compression level, and default compression level, as described in Configuring Connection Compression for X Plugin.

When message compression is in use, the `Mysqlx_bytes_sent` status variable shows the total number of bytes sent out from the server, including compressed message payloads measured after compression, any items in compressed messages that were not compressed such as X Protocol headers, and any uncompressed messages. The `Mysqlx_bytes_sent_compressed_payload` status variable shows the total number of bytes sent as compressed message payloads, measured after compression, and the `Mysqlx_bytes_sent_uncompressed_frame` status variable shows the total number of bytes for those same message payloads but measured before compression. The compression ratio, which shows the efficiency of the compression algorithm, can therefore be calculated using the following expression:

```
mysqlx_bytes_sent_uncompressed_frame / mysqlx_bytes_sent_compressed_payload
```

The effectiveness of compression for X Protocol messages sent by the server can be calculated using the following expression:

```
(mysqlx_bytes_sent - mysqlx_bytes_sent_compressed_payload + mysqlx_bytes_sent_uncompressed_frame) / mysqlx_
```

For messages received by the server from clients, the `Mysqlx_bytes_received_compressed_payload` status variable shows the total number of bytes received as compressed message payloads, measured before decompression, and the `Mysqlx_bytes_received_uncompressed_frame` status variable shows the total number of bytes for those same message payloads but measured after decompression. The `Mysqlx_bytes_received` status variable includes compressed message payloads measured before decompression, any uncompressed items in compressed messages, and any uncompressed messages.

20.5.6 X Plugin Options and Variables

This section describes the command options and system variables that configure X Plugin, as well as the status variables available for monitoring purposes. If configuration values specified at startup time are incorrect, X Plugin could fail to initialize properly and the server does not load it. In this case, the server could also produce error messages for other X Plugin settings because it cannot recognize them.

20.5.6.1 X Plugin Option and Variable Reference

This table provides an overview of the command options, system variables, and status variables provided by X Plugin.

Table 20.2 X Plugin Option and Variable Reference

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
mysqlx	Yes	Yes				
Mysqlx_aborted_clients				Yes	Global	No
Mysqlx_address				Yes	Global	No
mysqlx_bind_address	Yes	Yes	Yes		Global	No
Mysqlx_bytes_received				Yes	Both	No
Mysqlx_bytes_received_compressed_payload				Yes	Both	No
Mysqlx_bytes_received_uncompressed_frame				Yes	Both	No
Mysqlx_bytes_sent				Yes	Both	No
Mysqlx_bytes_sent_compressed_payload				Yes	Both	No
Mysqlx_bytes_sent_uncompressed_frame				Yes	Both	No
Mysqlx_compression_algorithm				Yes	Session	No
mysqlx_compression_algorithms	Yes	Yes	Yes		Global	Yes
Mysqlx_compression_level				Yes	Session	No
mysqlx_connect_timeout	Yes	Yes	Yes		Global	Yes
Mysqlx_connection_accept_errors				Yes	Both	No
Mysqlx_connection_errors				Yes	Both	No
Mysqlx_connections_accepted				Yes	Global	No
Mysqlx_connections_closed				Yes	Global	No
Mysqlx_connections_rejected				Yes	Global	No
Mysqlx_crud_create_view				Yes	Both	No
Mysqlx_crud_delete				Yes	Both	No
Mysqlx_crud_drop_view				Yes	Both	No
Mysqlx_crud_find				Yes	Both	No
Mysqlx_crud_insert				Yes	Both	No
Mysqlx_crud_modify_view				Yes	Both	No
Mysqlx_crud_update				Yes	Both	No
mysqlx_deflate_default	Yes	Yes	Yes		Global	Yes
mysqlx_deflate_max_client_compression_level	Yes	Yes	Yes		Global	Yes
mysqlx_document_id_unique_prefix	Yes	Yes	Yes		Global	Yes
mysqlx_enable_hello_notice	Yes	Yes	Yes		Global	Yes
Mysqlx_errors_sent				Yes	Both	No
Mysqlx_errors_unknown_message_type				Yes	Both	No
Mysqlx_expect_close				Yes	Both	No
Mysqlx_expect_open				Yes	Both	No
mysqlx_idle_worker_thread_timeout	Yes	Yes	Yes		Global	Yes
Mysqlx_init_error				Yes	Both	No
mysqlx_interactive_timeout	Yes	Yes	Yes		Global	Yes
mysqlx_lz4_default_compression_level	Yes	Yes	Yes		Global	Yes
mysqlx_lz4_max_client_compression_level	Yes	Yes	Yes		Global	Yes
mysqlx_max_allowed_packet	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
mysqlx_max_connections	Yes	Yes	Yes		Global	Yes
Mysqlx_messages_sent				Yes	Both	No
mysqlx_min_worker_threads	Yes	Yes	Yes		Global	Yes
Mysqlx_notice_global_sent				Yes	Both	No
Mysqlx_notice_other_sent				Yes	Both	No
Mysqlx_notice_warning_sent				Yes	Both	No
Mysqlx_notified_by_group_replication				Yes	Both	No
Mysqlx_port				Yes	Global	No
mysqlx_port	Yes	Yes	Yes		Global	No
mysqlx_port_open_timeout	Yes	Yes	Yes		Global	No
mysqlx_read_timeout	Yes	Yes	Yes		Session	Yes
Mysqlx_rows_sent				Yes	Both	No
Mysqlx_sessions				Yes	Global	No
Mysqlx_sessions_accepted				Yes	Global	No
Mysqlx_sessions_closed				Yes	Global	No
Mysqlx_sessions_fatal_error				Yes	Global	No
Mysqlx_sessions_killed				Yes	Global	No
Mysqlx_sessions_rejected				Yes	Global	No
Mysqlx_socket				Yes	Global	No
mysqlx_socket	Yes	Yes	Yes		Global	No
Mysqlx_ssl_accept_renegotiates				Yes	Global	No
Mysqlx_ssl_accepts				Yes	Global	No
Mysqlx_ssl_active				Yes	Both	No
mysqlx_ssl_ca	Yes	Yes	Yes		Global	No
mysqlx_ssl_capath	Yes	Yes	Yes		Global	No
mysqlx_ssl_cert	Yes	Yes	Yes		Global	No
Mysqlx_ssl_cipher				Yes	Both	No
mysqlx_ssl_cipher	Yes	Yes	Yes		Global	No
Mysqlx_ssl_cipher_list				Yes	Both	No
mysqlx_ssl_crl	Yes	Yes	Yes		Global	No
mysqlx_ssl_crlpath	Yes	Yes	Yes		Global	No
Mysqlx_ssl_ctx_verify_depth				Yes	Both	No
Mysqlx_ssl_ctx_verify_mode				Yes	Both	No
Mysqlx_ssl_finished_accepts				Yes	Global	No
mysqlx_ssl_key	Yes	Yes	Yes		Global	No
Mysqlx_ssl_server_not_after				Yes	Global	No
Mysqlx_ssl_server_not_before				Yes	Global	No
Mysqlx_ssl_verify_depth				Yes	Global	No
Mysqlx_ssl_verify_mode				Yes	Global	No
Mysqlx_ssl_version				Yes	Both	No
Mysqlx_stmt_create_collection				Yes	Both	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
Mysqlx_stmt_create_collection_index				Yes	Both	No
Mysqlx_stmt_disable_notices				Yes	Both	No
Mysqlx_stmt_drop_collection				Yes	Both	No
Mysqlx_stmt_drop_collection_index				Yes	Both	No
Mysqlx_stmt_enable_notices				Yes	Both	No
Mysqlx_stmt_ensure_collection				Yes	Both	No
Mysqlx_stmt_execute_mysqlx				Yes	Both	No
Mysqlx_stmt_execute_sql				Yes	Both	No
Mysqlx_stmt_execute_xplugin				Yes	Both	No
Mysqlx_stmt_get_collection_options				Yes	Both	No
Mysqlx_stmt_kill_client				Yes	Both	No
Mysqlx_stmt_list_clients				Yes	Both	No
Mysqlx_stmt_list_notices				Yes	Both	No
Mysqlx_stmt_list_objects				Yes	Both	No
Mysqlx_stmt_modify_collection_options				Yes	Both	No
Mysqlx_stmt_ping				Yes	Both	No
mysqlx_wait_timeout	Yes	Yes	Yes		Session	Yes
Mysqlx_worker_threads				Yes	Global	No
Mysqlx_worker_threads_active				Yes	Global	No
mysqlx_write_timeout	Yes	Yes	Yes		Session	Yes
mysqlx_zstd_default_compression_level	Yes	Yes	Yes		Global	Yes
mysqlx_zstd_max_client_compression_level	Yes	Yes	Yes		Global	Yes

20.5.6.2 X Plugin Options and System Variables

To control activation of X Plugin, use this option:

- `--mysqlx[=value]`

Command-Line Format	`--mysqlx[=value]`
Type	Enumeration
Default Value	`ON`

Valid Values	`ON`
	`OFF`
	`FORCE`
	`FORCE_PLUS_PERMANENT`

This option controls how the server loads X Plugin at startup. In MySQL 8.0, X Plugin is enabled by default, but this option may be used to control its activation state.

The option value should be one of those available for plugin-loading options, as described in Section 5.6.1, “Installing and Uninstalling Plugins”.

If X Plugin is enabled, it exposes several system variables that permit control over its operation:

- `mysqlx_bind_address`

Command-Line Format	`--mysqlx-bind-address=addr`
System Variable	`mysqlx_bind_address`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	String
Default Value	`*`

The network address on which X Plugin listens for TCP/IP connections. This variable is not dynamic and can be configured only at startup. This is the X Plugin equivalent of the `bind_address` system variable; see that variable description for more information.

By default, X Plugin accepts TCP/IP connections on all server host IPv4 interfaces, and, if the server host supports IPv6, on all IPv6 interfaces. If `mysqlx_bind_address` is specified, its value must satisfy these requirements:

- Prior to MySQL 8.0.21, `mysqlx_bind_address` accepts a single address value, which may specify a single non-wildcard IP address (either IPv4 or IPv6), or a host name, or one of the wildcard address formats that permit listening on multiple network interfaces (`*`, `0.0.0.0`, or `::`).
- As of MySQL 8.0.21, `mysqlx_bind_address` accepts either a single value as just described, or a list of comma-separated values. When the variable names a list of multiple values, each value must specify a single non-wildcard IP address (either IPv4 or IPv6) or a host name. Wildcard address formats (`*`, `0.0.0.0`, or `::`) are not allowed in a list of values.
- As of MySQL 8.0.22, the value may include a network namespace specifier.

IP addresses can be specified as IPv4 or IPv6 addresses. For any value that is a host name, X Plugin resolves the name to an IP address and binds to that address. If a host name resolves to multiple IP addresses, X Plugin uses the first IPv4 address if there are any, or the first IPv6 address otherwise.

X Plugin treats different types of addresses as follows:

- If the address is `*`, X Plugin accepts TCP/IP connections on all server host IPv4 interfaces, and, if the server host supports IPv6, on all IPv6 interfaces. Use this address to permit both IPv4 and IPv6 connections for X Plugin. This value is the default. If the variable specifies a list of multiple values, this value is not permitted.
- If the address is `0.0.0.0`, X Plugin accepts TCP/IP connections on all server host IPv4 interfaces. If the variable specifies a list of multiple values, this value is not permitted.
- If the address is `::`, X Plugin accepts TCP/IP connections on all server host IPv4 and IPv6 interfaces. If the variable specifies a list of multiple values, this value is not permitted.
- If the address is an IPv4-mapped address, X Plugin accepts TCP/IP connections for that address, in either IPv4 or IPv6 format. For example, if X Plugin is bound to `::ffff:127.0.0.1`, a client such as MySQL Shell can connect using `--host=127.0.0.1` or `--host=::ffff:127.0.0.1`.
- If the address is a “regular” IPv4 or IPv6 address (such as `127.0.0.1` or `::1`), X Plugin accepts TCP/IP connections only for that IPv4 or IPv6 address.

These rules apply to specifying a network namespace for an address:

- 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.
- If the variable value specifies multiple addresses, it can include addresses in the global namespace, in named namespaces, or a mix.

For additional information about network namespaces, see Section 5.1.14, “Network Namespace Support”.

Important

Because X Plugin is not a mandatory plugin, it does not prevent server startup if there is an error in the specified address or list of addresses (as MySQL Server does for `bind_address` errors). With X Plugin, if one of the listed addresses cannot be parsed or if X Plugin cannot bind to it, the address is skipped, an error message is logged, and X Plugin attempts to bind to each of the remaining addresses. X Plugin's `Mysqlx_address` status variable displays only those addresses from the list for which the bind succeeded. If none of the listed addresses results in a successful bind, or if a single specified address fails, X Plugin logs the error message `ER_XPLUGIN_FAILED_TO_PREPARE_IO_INTERFACES` stating that X Protocol cannot be used. `mysqlx_bind_address` is not dynamic, so to fix any issues you must stop the server, correct the system variable value, and restart the server.

- `mysqlx_compression_algorithms`

Command-Line Format	`--mysqlx-compression-algorithms=value`
Introduced	8.0.19
System Variable	`mysqlx_compression_algorithms`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Set
Default Value	`deflate_stream,lz4_message,zstd_stream`

Valid Values	`deflate_stream`
	`lz4_message`
	`zstd_stream`

The compression algorithms that are permitted for use on X Protocol connections. By default, the Deflate, LZ4, and zstd algorithms are all permitted. To disallow any of the algorithms, set `mysqlx_compression_algorithms` to include only the ones you permit. The algorithm names `deflate_stream`, `lz4_message`, and `zstd_stream` can be specified in any combination, and the order and case are not important. If you set the system variable to the empty string, no compression algorithms are permitted and only uncompressed connections are used. Use the algorithm-specific

system variables to adjust the default and maximum compression level for each permitted algorithm. For more details, and information on how connection compression for X Protocol relates to the equivalent settings for MySQL Server, see Section 20.5.5, “Connection Compression with X Plugin”.

- `mysqlx_connect_timeout`

Command-Line Format	`--mysqlx-connect-timeout=#`
System Variable	`mysqlx_connect_timeout`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`30`
Minimum Value	`1`
Maximum Value	`1000000000`
Unit	seconds

The number of seconds X Plugin waits for the first packet to be received from newly connected clients. This is the X Plugin equivalent of `connect_timeout`; see that variable description for more information.

- `mysqlx_deflate_default_compression_level`

Command-Line Format	`--mysqlx_deflate_default_compression_level=#`
Introduced	8.0.20
System Variable	`mysqlx_deflate_default_compression_level`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`3`
Minimum Value	`1`
Maximum Value	`9`

The default compression level that the server uses for the Deflate algorithm on X Protocol connections. Specify the level as an integer from 1 (the lowest compression effort) to 9 (the highest effort). This level is used if the client does not request a compression level during capability negotiation. If you do not specify this system variable, the server uses level 3 as the default. For more information, see Section 20.5.5, “Connection Compression with X Plugin”.

- `mysqlx_deflate_max_client_compression_level`

Command-Line Format	`--mysqlx_deflate_max_client_compression_level=#`
Introduced	8.0.20
System Variable	`mysqlx_deflate_max_client_compression_level`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No

Type	Integer
Default Value	5
Minimum Value	1
Maximum Value	9

The maximum compression level that the server permits for the Deflate algorithm on X Protocol connections. The range is the same as for the default compression level for this algorithm. If the client requests a higher compression level than this, the server uses the level you set here. If you do not specify this system variable, the server sets a maximum compression level of 5.

- `mysqlx_document_id_unique_prefix`

Command-Line Format	`--mysqlx-document-id-unique-prefix=#`
System Variable	`mysqlx_document_id_unique_prefix`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	0
Minimum Value	0
Maximum Value	65535

Sets the first 4 bytes of document IDs generated by the server when documents are added to a collection. By setting this variable to a unique value per instance, you can ensure document IDs are unique across instances. See Understanding Document IDs.

- `mysqlx_enable_hello_notice`

Command-Line Format	`--mysqlx-enable-hello-notice[={OFF|ON}]`
System Variable	`mysqlx_enable_hello_notice`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Boolean
Default Value	ON

Controls messages sent to classic MySQL protocol clients that try to connect over X Protocol. When enabled, clients which do not support X Protocol that attempt to connect to the server X Protocol port receive an error explaining they are using the wrong protocol.

- `mysqlx_idle_worker_thread_timeout`

Command-Line Format	`--mysqlx-idle-worker-thread-timeout=#`
System Variable	`mysqlx_idle_worker_thread_timeout`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer

Default Value	60
Minimum Value	0
Maximum Value	3600
Unit	seconds

The number of seconds after which idle worker threads are terminated.

- mysqlx_interactive_timeout

Command-Line Format	--mysqlx-interactive-timeout=#
System Variable	mysqlx_interactive_timeout
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	28800
Minimum Value	1
Maximum Value	2147483
Unit	seconds

The default value of the mysqlx_wait_timeout session variable for interactive clients. (The number of seconds to wait for interactive clients to timeout.)

- mysqlx_lz4_default_compression_level

Command-Line Format	--mysqlx_lz4_default_compression_level=#
Introduced	8.0.20
System Variable	mysqlx_lz4_default_compression_level
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	2
Minimum Value	0
Maximum Value	16

The default compression level that the server uses for the LZ4 algorithm on X Protocol connections. Specify the level as an integer from 0 (the lowest compression effort) to 16 (the highest effort). This level is used if the client does not request a compression level during capability negotiation. If you do not specify this system variable, the server uses level 2 as the default. For more information, see Section 20.5.5, “Connection Compression with X Plugin”.

- mysqlx_lz4_max_client_compression_level

Command-Line Format	--mysqlx_lz4_max_client_compression_level=#
Introduced	8.0.20
System Variable	mysqlx_lz4_max_client_compression_level

Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	8
Minimum Value	0
Maximum Value	16

The maximum compression level that the server permits for the LZ4 algorithm on X Protocol connections. The range is the same as for the default compression level for this algorithm. If the client requests a higher compression level than this, the server uses the level you set here. If you do not specify this system variable, the server sets a maximum compression level of 8.

- mysqlx_max_allowed_packet

Command-Line Format	--mysqlx-max-allowed-packet=#
System Variable	mysqlx_max_allowed_packet
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	67108864
Minimum Value	512
Maximum Value	1073741824
Unit	bytes

The maximum size of network packets that can be received by X Plugin. This limit also applies when compression is used for the connection, so the network packet must be smaller than this size after the message has been decompressed. This is the X Plugin equivalent of max_allowed_packet; see that variable description for more information.

- mysqlx_max_connections

Command-Line Format	--mysqlx-max-connections=#
System Variable	mysqlx_max_connections
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	100
Minimum Value	1
Maximum Value	65535

The maximum number of concurrent client connections X Plugin can accept. This is the X Plugin equivalent of max_connections; see that variable description for more information.

For modifications to this variable, if the new value is smaller than the current number of connections, the new limit is taken into account only for new connections.

- mysqlx_min_worker_threads

Command-Line Format	`--mysqlx-min-worker-threads=#`
System Variable	`mysqlx_min_worker_threads`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`2`
Minimum Value	`1`
Maximum Value	`100`

The minimum number of worker threads used by X Plugin for handling client requests.

- `mysqlx_port`

Command-Line Format	`--mysqlx-port=port_num`
System Variable	`mysqlx_port`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`33060`
Minimum Value	`1`
Maximum Value	`65535`

The network port on which X Plugin listens for TCP/IP connections. This is the X Plugin equivalent of `port`; see that variable description for more information.

- `mysqlx_port_open_timeout`

Command-Line Format	`--mysqlx-port-open-timeout=#`
System Variable	`mysqlx_port_open_timeout`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`120`
Unit	seconds

The number of seconds X Plugin waits for a TCP/IP port to become free.

- `mysqlx_read_timeout`

Command-Line Format	`--mysqlx-read-timeout=#`
System Variable	`mysqlx_read_timeout`
Scope	Session

Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	30
Minimum Value	1
Maximum Value	2147483
Unit	seconds

The number of seconds that X Plugin waits for blocking read operations to complete. After this time, if the read operation is not successful, X Plugin closes the connection and returns a warning notice with the error code ER_IO_READ_ERROR to the client application.

- mysqlx_socket

Command-Line Format	--mysqlx-socket=file_name
System Variable	mysqlx_socket
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String
Default Value	/tmp/mysqlx.sock

The path to a Unix socket file which X Plugin uses for connections. This setting is only used by MySQL Server when running on Unix operating systems. Clients can use this socket to connect to MySQL Server using X Plugin.

The default mysqlx_socket path and file name is based on the default path and file name for the main socket file for MySQL Server, with the addition of an x appended to the file name. The default path and file name for the main socket file is /tmp/mysql.sock, therefore the default path and file name for the X Plugin socket file is /tmp/mysqlx.sock.

If you specify an alternative path and file name for the main socket file at server startup using the socket system variable, this does not affect the default for the X Plugin socket file. In this situation, if you want to store both sockets at a single path, you must set the mysqlx_socket system variable as well. For example in a configuration file:

```
socket=/home/sockets/mysqld/mysql.sock
mysqlx_socket=/home/sockets/xplugin/xplugin.sock
```

If you change the default path and file name for the main socket file at compile time using the MYSQL_UNIX_ADDR compile option, this does affect the default for the X Plugin socket file, which is formed by appending an x to the MYSQL_UNIX_ADDR file name. If you want to set a different default for the X Plugin socket file at compile time, use the MYSQLX_UNIX_ADDR compile option.

The MYSQLX_UNIX_PORT environment variable can also be used to set a default for the X Plugin socket file at server startup (see Section 4.9, “Environment Variables”). If you set this environment variable, it overrides the compiled MYSQLX_UNIX_ADDR value, but is overridden by the mysqlx_socket value.

- mysqlx_ssl_ca

Command-Line Format	--mysqlx-ssl-ca=file_name
System Variable	mysqlx_ssl_ca
Scope	Global

Dynamic	No
SET_VAR Hint Applies	No
Type	File name
Default Value	NULL

The mysqlx_ssl_ca system variable is like ssl_ca, except that it applies to X Plugin rather than the MySQL Server main connection interface. For information about configuring encryption support for X Plugin, see Section 20.5.3, “Using Encrypted Connections with X Plugin”.

- mysqlx_ssl_capath

Command-Line Format	--mysqlx-ssl-capath=dir_name
System Variable	mysqlx_ssl_capath
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Directory name
Default Value	NULL

The mysqlx_ssl_capath system variable is like ssl_capath, except that it applies to X Plugin rather than the MySQL Server main connection interface. For information about configuring encryption support for X Plugin, see Section 20.5.3, “Using Encrypted Connections with X Plugin”.

- mysqlx_ssl_cert

Command-Line Format	--mysqlx-ssl-cert=file_name
System Variable	mysqlx_ssl_cert
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	File name
Default Value	NULL

The mysqlx_ssl_cert system variable is like ssl_cert, except that it applies to X Plugin rather than the MySQL Server main connection interface. For information about configuring encryption support for X Plugin, see Section 20.5.3, “Using Encrypted Connections with X Plugin”.

- mysqlx_ssl_cipher

Command-Line Format	--mysqlx-ssl-cipher=name
System Variable	mysqlx_ssl_cipher
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String
Default Value	NULL

The mysqlx_ssl_cipher system variable is like ssl_cipher, except that it applies to X Plugin rather than the MySQL Server main connection interface. For information about configuring encryption support for X Plugin, see Section 20.5.3, “Using Encrypted Connections with X Plugin”.

- `mysqlx_ssl_crl`

Command-Line Format	`--mysqlx-ssl-crl=file_name`
System Variable	`mysqlx_ssl_crl`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	File name
Default Value	`NULL`

The `mysqlx_ssl_crl` system variable is like `ssl_crl`, except that it applies to X Plugin rather than the MySQL Server main connection interface. For information about configuring encryption support for X Plugin, see Section 20.5.3, “Using Encrypted Connections with X Plugin”.

- `mysqlx_ssl_crlpath`

Command-Line Format	`--mysqlx-ssl-crlpath=dir_name`
System Variable	`mysqlx_ssl_crlpath`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	Directory name
Default Value	`NULL`

The `mysqlx_ssl_crlpath` system variable is like `ssl_crlpath`, except that it applies to X Plugin rather than the MySQL Server main connection interface. For information about configuring encryption support for X Plugin, see Section 20.5.3, “Using Encrypted Connections with X Plugin”.

- `mysqlx_ssl_key`

Command-Line Format	`--mysqlx-ssl-key=file_name`
System Variable	`mysqlx_ssl_key`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	File name
Default Value	`NULL`

The `mysqlx_ssl_key` system variable is like `ssl_key`, except that it applies to X Plugin rather than the MySQL Server main connection interface. For information about configuring encryption support for X Plugin, see Section 20.5.3, “Using Encrypted Connections with X Plugin”.

- `mysqlx_wait_timeout`

Command-Line Format	`--mysqlx-wait-timeout=#`
System Variable	`mysqlx_wait_timeout`
Scope	Session
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer

Default Value	28800
Minimum Value	1
Maximum Value	2147483
Unit	seconds

The number of seconds that X Plugin waits for activity on a connection. After this time, if the read operation is not successful, X Plugin closes the connection. If the client is noninteractive, the initial value of the session variable is copied from the global `mysqlx_wait_timeout` variable. For interactive clients, the initial value is copied from the session `mysqlx_interactive_timeout`.

- `mysqlx_write_timeout`

Command-Line Format	`--mysqlx-write-timeout=#`
System Variable	`mysqlx_write_timeout`
Scope	Session
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	60
Minimum Value	1
Maximum Value	2147483
Unit	seconds

The number of seconds that X Plugin waits for blocking write operations to complete. After this time, if the write operation is not successful, X Plugin closes the connection.

- `mysqlx_zstd_default_compression_level`

Command-Line Format	`--mysqlx_zstd_default_compression_level=#`
Introduced	8.0.20
System Variable	`mysqlx_zstd_default_compression_level`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	3
Minimum Value	-131072
Maximum Value	22

The default compression level that the server uses for the zstd algorithm on X Protocol connections. For versions of the zstd library from 1.4.0, you can set positive values from 1 to 22 (the highest compression effort), or negative values which represent progressively lower effort. A value of 0 is converted to a value of 1. For earlier versions of the zstd library, you can only specify the value 3. This level is used if the client does not request a compression level during capability negotiation. If you do not specify this system variable, the server uses level 3 as the default. For more information, see Section 20.5.5, “Connection Compression with X Plugin”.

- `mysqlx_zstd_max_client_compression_level`

Command-Line Format	`--mysqlx_zstd_max_client_compression_level=#`
Introduced	8.0.20
System Variable	`mysqlx_zstd_max_client_compression_level`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`11`
Minimum Value	`-131072`
Maximum Value	`22`

The maximum compression level that the server permits for the zstd algorithm on X Protocol connections. The range is the same as for the default compression level for this algorithm. If the client requests a higher compression level than this, the server uses the level you set here. If you do not specify this system variable, the server sets a maximum compression level of 11.

20.5.6.3 X Plugin Status Variables

The X Plugin status variables have the following meanings.

- `Mysqlx_aborted_clients`

  The number of clients that were disconnected because of an input or output error.

- `Mysqlx_address`

  The network address or addresses for which X Plugin accepts TCP/IP connections. If multiple addresses were specified using the `mysqlx_bind_address` system variable, `Mysqlx_address` displays only those addresses for which the bind succeeded. If the bind has failed for every network address specified by `mysqlx_bind_address`, or if the `skip_networking` option has been used, the value of `Mysqlx_address` is `UNDEFINED`. If X Plugin startup is not yet complete, the value of `Mysqlx_address` is empty.

- `Mysqlx_bytes_received`

  The total number of bytes received through the network. If compression is used for the connection, this figure comprises compressed message payloads measured before decompression (`Mysqlx_bytes_received_compressed_payload`), any items in compressed messages that were not compressed such as X Protocol headers, and any uncompressed messages.

- `Mysqlx_bytes_received_compressed_payload`

  The number of bytes received as compressed message payloads, measured before decompression.

- `Mysqlx_bytes_received_uncompressed_frame`

  The number of bytes received as compressed message payloads, measured after decompression.

- `Mysqlx_bytes_sent`

  The total number of bytes sent through the network. If compression is used for the connection, this figure comprises compressed message payloads measured after compression (`Mysqlx_bytes_sent_compressed_payload`), any items in compressed messages that were not compressed such as X Protocol headers, and any uncompressed messages.

- `Mysqlx_bytes_sent_compressed_payload`

  The number of bytes sent as compressed message payloads, measured after compression.

- `Mysqlx_bytes_sent_uncompressed_frame`

  The number of bytes sent as compressed message payloads, measured before compression.

- `Mysqlx_compression_algorithm`

  (Session scope) The compression algorithm in use for the X Protocol connection for this session. The permitted compression algorithms are listed by the `mysqlx_compression_algorithms` system variable.

- `Mysqlx_compression_level`

  (Session scope) The compression level in use for the X Protocol connection for this session.

- `Mysqlx_connection_accept_errors`

  The number of connections which have caused accept errors.

- `Mysqlx_connection_errors`

  The number of connections which have caused errors.

- `Mysqlx_connections_accepted`

  The number of connections which have been accepted.

- `Mysqlx_connections_closed`

  The number of connections which have been closed.

- `Mysqlx_connections_rejected`

  The number of connections which have been rejected.

- `Mysqlx_crud_create_view`

  The number of create view requests received.

- `Mysqlx_crud_delete`

  The number of delete requests received.

- `Mysqlx_crud_drop_view`

  The number of drop view requests received.

- `Mysqlx_crud_find`

  The number of find requests received.

- `Mysqlx_crud_insert`

  The number of insert requests received.

- `Mysqlx_crud_modify_view`

  The number of modify view requests received.

- `Mysqlx_crud_update`

  The number of update requests received.