# MySQL Replication

## Replication Overview:

* + Replication enables copying data from one MySQL database server (source) to one or more MySQL database servers (replica). This is a process that allows you to easily maintain multiple copies of a MySQL data by having them copied automatically from a master to a slave database.
  + MySQL replication tracks updates performed in a MySQL master logs and updates one or more slave servers. The mechanism involved in MySQL replication requires binary logging to be active.
    - In asynchronous replication, the slave does not need to be active all the time.
    - MySQL replication can be used to scale-out queries among different servers.
    - MySQL replication is a good solution for performing live backups.
    - As data is replicated to the replica and the replica can pause the replication process, it is possible to run the backup service on the replica without corrupting the corresponding source data.
    - Long distance data distribution.
  + Each update performed in a database is replicated from the master to the slaves.
  + Only slaves keep track of their replication status.
  + One may set at what point the replication begins

Slave 1

Master DB

Slave 2

* **Different methods of replication:**
  + The traditional method –
    - Based on replicating events from the source's binary log
    - Requires the log files and positions to be synchronized between the source and replica.
  + Global Transaction Identifiers (GTIDs) –
    - Unique identifier associated with each transaction performed on the origin server (source).
    - This identifier is unique not only to the server it originated from, but across all servers in a given replication topology.
* **Different types of synchronization:** 
  + Asynchronous replication –
    - One server act as the source, while one or more other servers act as replicas.
    - Writes data to the primary storage array first and then, depending on the implementation approach, commits data to be replicated to memory

1, Write Request

Replica Host

2. Replication Request

Source Host

Primary DB

Target DB

4.ACK

3. ACK

* + Synchronous replication –
    - Write data to primary storage and the replica simultaneously. As such, the primary copy and the replica should always remain synchronized.

3. Replication Request

Primary DB

2.ACK

1, Write Request

Source Host

Replica Host

Target DB

4. ACK

* + Semi-synchronous –
    - Falls between asynchronous and fully synchronous replication.
    - The source waits until at least one replica has received and logged the events (the required number of replicas is configurable), and then commits the transaction.

## Configuring Replication

* **Setting Up Binary Log File Position Based Replication:**
  + This section describes replication between MySQL servers based on the binary log file position method, where the MySQL instance operating as the source (where the database changes originate) writes updates and changes as “events” to the binary log.
  + Replicas are configured to read the binary log from the source and to execute the events in the binary log on the replica's local database.
  + If required, you can configure the replica to process only events that apply to particular databases or tables.
  + Unless you specify otherwise, all events in the source's binary log are executed on the replica.
  + It is the responsibility of the replica to decide which statements in the binary log should be executed.
  + Each replica receives a copy of the entire contents of the binary log.
  + Also, because each replica records the current position within the binary log, it is possible for replicas to be disconnected, reconnect and then resume processing.
  + Each replica keeps a record of the binary log coordinates: the file name and position within the file that it has read and processed from the source.
  + This means that multiple replicas can be connected to the source and executing different parts of the same binary log.
  + Because the replicas control this process, individual replicas can be connected and disconnected from the server without affecting the source's operation.
* **Methods to Setup MySQL BinLog Based Replication:**

Step 1: Enable Binary Logging and Configure Master

* Each server whether it is a master or slave of the replication structure needs to be configured with a unique server id.
* This needs to be accomplished using log-bin & server-id options available in the configuration file at {mysqld] section.
* This step is about enabling binary log and configuring a unique server ID on the master.
* You will need to shut down the MySQL server to edit the my.cnf or my.ini file.
* Here is how you can add log-bin & server-id options to enable the binary logging

[mysqld] log-bin=mysql-bin server-id=1

* Once the changes are done to the configuration file, you will need to restart the server.

Step 2: User Creation for Replication

* We will need to have the slave user accounts defined on the master server by granting ‘replication slave’ permissions.
* To create a new user account, use the CREATE USER command

mysql> CREATE USER 'repl'@'%.example.com' IDENTIFIED BY 'password';

* Newly or existing created accounts should have replication privileges granted

mysql> GRANT REPLICATION SLAVE ON \*.\* TO ‘username’@’%. domain.com’;

Step 3: Getting Binary Log Coordinates from the Replication Master

* To start the replication process, we should know the master’s current coordinates within its binary log to configure the slave.

[Warning]

This procedure uses FLUSH TABLES WITH READ LOCK, which blocks COMMIT operations for InnoDB tables.

* To find the master binary log coordinates, follow below steps:
  + Get started with a session on the master. With the command-line client, execute the FLUSH TABLES WITH READ LOCK statement:

mysql> FLUSH TABLES WITH READ LOCK;

* + This will flush all tables and block write statements.
  + Execute the SHOW MASTER STATUS statement in another session to obtain the current binary log file name and position:

mysql > SHOW MASTER STATUS;

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

| File | Position | Binlog\_Do\_DB | Binlog\_Ignore\_DB |

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

| mysql-bin.000003 | 73 | test | manual,mysql |

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

* + These are replication coordinates in the form of bin log file & position as shown in the above example. These values will need to be used while configuring the slaves.

Step 4: Creating a Data Snapshot

* We can create a dump of all the tables and or databases at the master server and import the same on the slave by using mysqldump. This will need to be performed before the replication task gets started.
* In this command, the data is dumped into the file ‘repl\_dbdump.db’ with –master-data option:

mysqldump --all-databases --master-data > dbdump.db

* If you want to exclude any tables in the database, use –ignore-table option.
* If you want to name only specific databases which you want to be dumped, use the –databases option

Step 5: Replication Slave Configuration

* Similar to the master, the slaves will also need to be set up with a unique server id so that it does not conflict. The same process will need to be followed to edit the configuration file as indicated in Step 1 with regard to server id.
* For example:

[mysqld]

server-id=2

* Once you are done with making the changes to the configuration file, restart the respective slave. There is no need to activate binary logging on the slave but if you want to, then it can also be used as a master to other slaves.

Step 6: Master Configuration on the Slave

* To configure the slave to communicate with the master for replication, set up the slave with the required connection information. The following statements will need to be executed to accomplish the same.

mysql> CHANGE MASTER TO

-> MASTER\_HOST='source\_host\_name',

-> MASTER\_USER='replication\_user\_name',

-> MASTER\_PASSWORD='replication\_password',

-> MASTER\_LOG\_FILE='recorded\_log\_file\_name',

-> MASTER\_LOG\_POS=recorded\_log\_position;

* The following step would be needed depending on whether you have a dump to import or not.

Step 7: Replication Setup between a New Master and Slaves

* If you have no snapshot of a previous database to import, then a new master is to be used for the replication.
* By starting the slave & executing CHANGE MASTER TO statement, you need to set up replication between a new master and the slave.

Next, perform the above step on each slave.

Step 8: Replication Setup with Existing Data

* When setting up replication with existing data, the snapshot needs to be sent from the master to the slave. This has to be performed before the replication process is started.
* By considering that we have used mysqldump
  + Start the slave, using the –skip-slave-start option. This will not trigger the replication
  + Import the dump file:

shell> mysql < fulldb.dump

* + With this, the slave connects to the master to replicate data updates.
  + The data available on slaves can be utilized continuously for analytics purposes. With this approach, the binary log will need to be observed for any changes in the data as per the events and replicate only those events based on the position id.
* **Common Replication Administration Tasks:**
  + **Checking Replication Status:**
  + If you are using the SHOW SLAVE STATUS statement to check on the status of an individual replica, the statement provides the following information:

mysql> SHOW SLAVE STATUS\G

* + The key fields from the status report to examine are:
    - Slave\_IO\_State: The current status of the replica.
    - Slave\_IO\_Running: Whether the I/O thread for reading the source's binary log is running. Normally, you want this to be Yes unless you have not yet started replication or have explicitly stopped it with STOP SLAVE.
    - Slave\_SQL\_Running: Whether the SQL thread for executing events in the relay log is running. As with the I/O thread, this should normally be Yes.
    - Last\_IO\_Error, Last\_SQL\_Error: The last errors registered by the I/O and SQL threads when processing the relay log. Ideally these should be blank, indicating no errors.
    - Seconds\_Behind\_Master: The number of seconds that the replication SQL thread is behind processing the source's binary log. A high number (or an increasing one) can indicate that the replica is unable to handle events from the source in a timely fashion.
  + You can check the status of connected replicas using SHOW PROCESSLIST to examine the list of running processes. Replica connections have Binlog Dump in the Command field:

mysql> SHOW PROCESSLIST \G;

* + For replicas that were started with the --report-host option and are connected to the source, the SHOW SLAVE HOSTS statement on the source shows basic information about the replicas.
  + **Pausing Replication on the Replica:**
    - Stop/Start Slave:

mysql> <STOP/START> SLAVE;

* + - To stop/start a particular thread, specify the thread type:

mysql> <STOP/START> SLAVE IO\_THREAD;

mysql> <STOP/START> SLAVE SQL\_THREAD;

* + **Skipping Transactions:**
    - If replication stops due to an issue with an event in a replicated transaction, you can resume replication by skipping the failed transaction on the replica.
    - Before skipping a transaction, ensure that the replication I/O thread is stopped as well as the replication SQL thread.
    - Before skipping the transaction and restarting the replica, check these points: Is the transaction that stopped replication from an unknown or untrusted source? If so, investigate the cause in case there are any security considerations that indicate the replica should not be restarted.
    - Does the transaction that stopped replication need to be applied on the replica? If so, either make the appropriate corrections and reapply the transaction, or manually reconcile the data on the replica.
    - Did the transaction that stopped replication need to be applied on the source? If not, undo the transaction manually on the server where it originally took place.
    - *Skipping Transactions With GTIDs:*

SET GTID\_NEXT='aaa-bbb-ccc-ddd:N';

BEGIN;

COMMIT;

SET GTID\_NEXT='AUTOMATIC';

* + - *Skipping Transactions With SET GLOBAL sql\_slave\_skip\_counter*

SET GLOBAL sql\_slave\_skip\_counter = N

* + - *Skipping Transactions With CHANGE MASTER TO:*

CHANGE MASTER TO MASTER\_LOG\_FILE='source\_log\_name', MASTER\_LOG\_POS=source\_log\_pos;

## Replication Implementation

* **Two core types of replication format:**
  + **Statement Based Replication (SBR)**, which replicates entire SQL statements

{INSERT,t,(1,''Sun,999)};

{UPDATE,t,id ==1,(NULL,'Moon',NULL)};

CREATE TABLE t(id INT,name VARCHAR(12));

ALTER TABLE t ADD COLUMN id2 INT;

Advantage:

* + - There is always less data that is to be transferred between the master and the slave.
    - There is less space taken up in the update logs.
    - There is no need to deal with the row format.
    - Also, auditing the database is easy, because statements that made any changes to the data are all logged in the binary log.

Disadvantage:

* + - The single biggest disadvantage of statement-based replication is the data-inconsistency issue between the master and the slave that creeps up due to the way this kind of replication works. Because we are logging the SQL statements, it is also necessary to log context information, so that the updates produce the same results on the slave as they did originally on the master. But in some cases it is not possible to provide any such context. Any nondeterministic behavior, is not going to have any such context present and hence is difficult to replicate using statement-based replication.  
      Let me quote an example here from the MySQL manual:  
      *“For example, for INSERT … SELECT with no ORDER BY, the SELECT may return rows in a different order (which results in a row having different ranks, hence getting a different number in the AUTO\_INCREMENT column), depending on the choices made by the optimizers on the master and slave.”*
    - With statement-based replication, you are bound to encounter issues with replicating stored routines or triggers, and hence this kind of replication does not always work with stored routines and triggers.
    - There is a performance penalty in the case of *INSERT … SELECT*, because in the case of statement-based replication this kind of statement requires a greater number of row-level locks as compared to row-based replication.
    - There is a lot of execution context information that is required in order for the updates to produce the same results on the slave as they did originally on the master.
    - A statement that depends on UDFs or stored routines that are nondeterministic, cannot be replicated properly, since the value returned by such a UDF or stored routine is not always the same, for similar parameters.
  + **Row Based Replication (RBR)**, which replicates only the changed rows.

CREATE TABLE t(id INT,name VARCHAR(12));

ALTER TABLE t ADD COLUMN id2 INT;

Advantages:

{INSERT,t,(1,''Sun,999)};

{UPDATE,t,id ==1,(NULL,'Moon',NULL)};

* + - With row-based replication, each and every change can be replicated and hence this is the safest form of replication.
    - Because every row update is physically logged, hence there no need for any context information. The only thing that is needed is to know which record is being updated and what is the update that is being written to that record.
    - There are fewer row locks required on the master, which thus achieves high concurrency.
    - The problems with auto\_increment columns, timestamps, stored routines, and triggers don’t bother us with this kind of replication.
    - Statements that update very few rows are very fast.

Disadvantage:

* + - On a system that frequently updates large number of rows such as,
    - UPDATE products set status='sold' where product\_id BETWEEN 30000 and 50000;
    - row-based replication produces very large update logs and generates a lot of network traffic between the master and the slave.
    - This kind of replication requires a lot of awareness of the internal row format.
    - In cases of very large updates, the performance overhead associated with the increased I/O required to write large update logs could become unacceptable.
    - You cannot examine the logs to audit changes to the database, because SQL statements are not logged, instead the data is logged in binary format.
  + Though you might point out the fact that large updates produce large update logs in case of row-based replication, but those cases in real-world would be far and few and not very frequent.
  + Although both the types have advantages and disadvantages, but for me the advantage that row-based replication offers in terms of data consistency between master and slave, far outweighs any of the disadvantages.
  + Also, stored routines and triggers are increasingly being used after their introduction in MySQL and row-based replication allows us to use them without thinking about them being unsafe for replication.

## Replication Threads

* MySQL replication capabilities are implemented using three main threads, one on the source server and two on the replica:
  + Binary log dump thread.
    - The binary log dump thread acquires a lock on the source's binary log for reading each event that is to be sent to the replica.
    - The source creates a thread to send the binary log contents to a replica when the replica connects.
  + Replication I/O thread.
    - When a START SLAVE statement is issued on a replica server, the replica creates an I/O thread, which connects to the source and asks it to send the updates recorded in its binary logs.
    - The replication I/O thread reads the updates that the source's Binlog Dump thread sends (see previous item) and copies them to local files that comprise the replica's relay log.
  + Replication SQL thread.
    - The replica creates an SQL thread to read the relay log that is written by the replication I/O thread and execute the transactions contained in it.
* Monitoring Replication Main Threads:
  + The SHOW PROCESSLIST statement provides information that tells you what is happening on the source and on the replica regarding replication.

mysql> SHOW PROCESSLIST \G;

* Monitoring Replication Applier Worker Threads:
  + On a multithreaded replica, the Performance Schema tables replication\_applier\_status\_by\_coordinator and replication\_applier\_status\_by\_worker show status information for the replica's coordinator thread and applier worker threads respectively.