**Crash, Recovery and Replication Strategy in MySQL**

Hedaya Khalif, Nada Elkelani, Melissa Tran

hkhalif@uwo.ca , nelkelan@uwo.ca, mtran247@uwo.ca

The University of Western Ontario

Department of Computer Science

**1 Background information**

In this project, we will be discussing replication using a MySQL DBMS. Replication is when data from a database can be replicated from one server to another. Replication can copy all the data in a server, or only specific databases depending on how the ‘change master’ replication command would be configured.[[1]](#endnote-1) MySQL replication has both master and slave servers. Masters are the servers that allow for write operations, whereas slave servers can only read.[[2]](#endnote-2) We will also discuss how certain actions make the system inconsistent, for example data should never be inserted into slave servers for databases that are replicated over from master servers.

* 1. **Asynchronous Replication**

Asynchronous replication is the default type of replication in MySQL. In this method, master servers do not have to wait on the slave to completely catch up for the database to continue to grow. Because slaves can be ‘behind the master’, a replication lag value is recorded so that an administrator can keep track of how many seconds the slave needs to catch up to its master.[[3]](#endnote-3) For example, a master server has an ‘x’ number of records in database ‘cs4411’. The slave currently has less than ‘x’ number of records but is constantly updating and catching up to the master. Replication lag can be found in the ‘show slave’ command in MySQL, showing how many seconds behind the slave server is to the master server.

**Advantages of Asynchronous Replication**: Writes are done in the master locally, therefore write latency is low causing less delays and multiple slaves can easily be added to read from the same master, without causing a great effect to the latency. Asynchronous replication is a more favourable option for companies that want to run applications with a lot of reading properties, as multiple read-only slaves can be configured to scale applications to be very large.[[4]](#endnote-4)

* 1. **Synchronous Replication**

With synchronous replication, the master server waits on the slave servers as data is not shown in the master until all the data has been successfully replicated over to at least one slave, with no replication lag. Unlike Asynchronous data replication, Synchronous data replication in MySQL is not the default replication setting and must be configured first in the my.cnf file.[[5]](#endnote-5)

**Advantages of Synchronous Replication:** This strategy is high in reliability and a good option for companies that cannot afford to lose data.

**1.3 Permissions**

In order for the slave replica to connect to it’s master, the slave needs a mysql username and password. When this user is created, it must be granted the privilege ‘replication slave’. The purpose of this permission is to allow the slave user to connect and receive updates from the MySQL master servers binary logs. Binary logs hold all transactions and history done on databases, so slave servers need access to this information to copy over data.[[6]](#endnote-6)

**1.4 Master - Slave Replication Strategy**

This replication strategy consists of one master and one slave (Fig.1). If the slave writes into its databases that are replicated from the master, some problems may be encountered. When the slave writes, the data will technically exist in the master but will not be visible. When data is duplicated in databases on primary keys, replication will break because the primary keys need to be unique. Therefore, if the data written by the slave is a duplicate entry to data already present in the master server, replication will require recovery. This is important to note, as this replication strategy writes only one way. [[7]](#endnote-7)

Fig.1. Master-slave replication

**1.5 Master - Master Replication Strategy**

This replication strategy has no slave, and two masters (Fig.2). Both servers are replicated from one another and writing can be done from both ends. The benefits to this strategy is that writing loads from users and applications are more balanced, which can increase performance. Writing loads are distributed, meaning it is a lot faster if 500 users use two separate servers as they would not have to use much resources, therefore increasing performance. It is important to note that once a server is participating in a master-master relationship, it cannot participate in another master-master relationship.[[8]](#endnote-8) For example, if master1 and master2 have a master-master relationship, then master1 and master3 cannot also have a master-master relationship since master1 already has one with master2.

Fig.2. Master-master replication

**1.6 Multi-source Replication Strategy**

In this replication strategy, one slave can be connected to multiple masters (Fig.3). For example, master 1 has database x and master 2 has database y. Slave 1 wants to replicate both database x and database y. In MySQL, it is possible for slave 1 to have data replicated to it by both master 1 and master 2. When the slave status is shown in slave 1, there will appear two different channels. This is the result of two different ‘change master’ commands that have to be applied to slave.[[9]](#endnote-9)

Fig.3. Multi-source replication

**1.7 Recovery**

If the master server crashes, then replication may break, impacting the system’s reliability. This can happen via network crash, system crash, duplicate entries, etc. In this case data will no longer be replicated over to the slave, making the data inconsistent between servers.

There are multiple methods of recovering data replication. When replication breaks, the error will be displayed in the MySQLl ‘show slave’ command beside the ‘last error’ variable.[[10]](#endnote-10) Error logs are documents that can show reasons as to why replication is broken , allowing for a rebuilding and recovery plan to be established.[[11]](#endnote-11) When replication is broken beyond repair, the final option is to rebuild the replication. If the replication is broken as a result of a bad query or duplicate entry, the query can be skipped to resolve the issue. To do this, we use a global variable contained in MySQL called the SQL\_SLAVE\_SKIP\_COUNTER.[[12]](#endnote-12) The command to repair the replication is as follows:

SET GLOBAL SQL\_SLAVE\_SKIP\_COUNTER = 1.

**2 Goals and Objective**

The goal of this report is to learn how to set up replication in a MySQL DBMS and to test out different replication strategies. We will record observations and reflect on how they react to different updates and inserts of data. We will be testing out three different replication strategies; master - slave, master - master and multi - source replication.

**3 Technologies, Techniques and Tools /Outcome**

**3.1 Setting Up The Environment**

To begin our testing, we set up three virtual machines using Oracle VM VirtualBox Manager. Two of these servers act as master servers, and one acts as only a slave server. In each of these three virtual machines, we have installed CentosOS 7 and MySQL version 5.7 (Fig.4).

Fig.4. Oracle VM VirtualBox Environment

An internal network adapter has been set up for each virtual machine, in order for the virtual boxes to be able to connect to one another. In order to make things easier to read for the set up of replication, The following commands were run on the servers to change the hostname of each server.

Sudo hostnamectl set-hostname master\_one;

Sudo hostnamectl set-hostname master\_two;

Sudo hostnamectl set-hostname slave;

Once our virtual machines were all set up, we were ready to begin our testing.

**3.2 Creating The Databases**

In order to test different replication strategies, we first created multiple databases within our servers to be able to test if the replication was working. Two different databases were created, one in Master\_one and the other in Master\_two. The database in Master\_one is called ‘Students’ and the database in Master\_two is called ‘Courses’. The following queries and data entries were executed in order to create the above databases.

**3.2.1 Master\_one, database ‘Students’**

mysql> create database Students;

Query OK, 1 row affected (0.00 sec)

mysql> use Students;

Database changed

mysql> show tables;

Empty set (0.00 sec)

mysql> create table `Names`(`studentNum` INT, `firstName` VARCHAR(20), `lastName` VARCHAR(20), PRIMARY KEY(`studentNum`));

Query OK, 0 rows affected (0.01 sec)

//Data inserted

mysql> INSERT INTO Names VALUES ('65453','Hedaya','Khalif');

Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO Names VALUES ('54242','sally','lane');

Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO Names VALUES ('12244','justin','thomson');

Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO Names VALUES ('898743','Melissa','Tran');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Names VALUES ('83594','Bob','smith');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Names VALUES ('22334','Anna','jane');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Names VALUES ('34355','alex','tin');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Names VALUES ('56343','Sarah','john');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Names VALUES ('54242','sally','lane');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Names VALUES ('12244','justin','thomson');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Names VALUES ('22334','Anna','jane');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Names VALUES ('34355','alex','tin');

Query OK, 1 row affected (0.01 sec)

**3.2.2 Master\_two, database ‘Courses’**

mysql> create database Courses;

Query OK, 1 row affected (0.00 sec)

mysql> use Courses;

Database changed

mysql> create table `Course`(`courseCode` INT, `teacherFirstName` VARCHAR(20), `teacherLastName` VARCHAR(20),`courseName` VARCHAR(20), PRIMARY KEY(`courseCode`));

Query OK, 0 rows affected (0.00 sec)

//Data Inserted

mysql> INSERT INTO Course VALUES ('4411','john','smith','Databases 2');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Course VALUES ('2212','justin','thomson','software eng');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Course VALUES ('2210','Anna','jane','algorthims');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Course VALUES ('2209','alex','tin','applied logics');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Course VALUES ('4412','Emily','young','distributed computing');

Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> INSERT INTO Course VALUES ('3307','Bob','smith','object-design');

Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO Course VALUES ('3319','Sarah','john','databases');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Course VALUES ('2211','sally','lane','file systems');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Course VALUES ('2212','justin','thomson','software eng');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Course VALUES ('2210','Anna','jane','algorithms');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Course VALUES ('2209','alex','tin','applied logics');

Query OK, 1 row affected (0.00 sec)

**3.3 Setting Up Replication**

**3.3.1 Master - Slave Replication**

The first strategy we implemented was the master-slave replication between server Master\_one and Slave. The first step in setting up replication between servers is making sure that the replication user in which Slave uses to read from Master\_one exists in the system. We created a user called replica\_slave in the Master\_one by executing the following command:

CREATE USER IF NOT EXISTS 'replica\_slave'@'%' IDENTIFIED BY ‘password123’;

GRANT REPLICATION SLAVE ON \*.\* TO 'replica\_slave'@'%';

Once the user was created, the next step to building the replication was to take a backup of the database ‘Students” found in Master\_one. In order to take this backup, we had to create a mysqldump command. Mysqldump is a command that is used to generate logical backups in MySQL. The following command was executed in the command line:

mysqldump -uroot -p -F -R --master-data=2 --opt --routines --triggers --no-create-db --single-transaction Students | gzip -c > full\_backup\_students.sql.gz

The -master-data-2 flag is important to the set up of replication, as this is the flag that will show the mysql binlog and position numbers that will be used to set up the replication. These values are important, as the binlog and position numbers will read the transactions on the master databases from the mysqldump and allow the slave to know its transactions. The database tables also had to be unlocked before any set up, so that they are accessible for replication.

Once our backup had completed successfully, we copied over the backup file that we created in Master\_one over to Slave using scp. Scp stands for secure copy protocol and is used to transfer files from localhost to a remote host. The following scp command was used to transfer the files between hosts:

scp -r root@master\_one:/tmp/full\_backup\_students.sql.gz .

Once our backup file successfully transfers over to our destination server (Slave) we are then able to begin the process of restoring the data into Mysql. We did this by using the following command :

gunzip -c full\_backup\_students.sql.gz | mysql -uroot -p

Once the database had been restored, we were ready to set up the replication. The replication is configured by executing a create master command in MySQL. The following command was executed to start the replication

CHANGE MASTER

TO MASTER\_HOST='master\_one',

MASTER\_PORT=3306,

MASTER\_USER=’replica\_slave',

MASTER\_PASSWORD='password123',

MASTER\_LOG\_FILE='mysql-bin.000001',

MASTER\_LOG\_POS=156;

start slave;

**3.3.2 Master - Master Replication**

The second replication strategy that we implemented was the master-master replication between Master\_one and Master\_two. Similar to the master - slave replication, a mysql user named replica\_master2 was created in Master\_one for Master\_two. Since we are building a two way replication between the servers, a user was also created in Master\_two for Master\_one called replica\_master1. A mysqldump backup file was created in order to establish the replication, taking a full backup of databases in  Master\_one.

Master-master replication is a two way replication strategy, so two “change master commands” were created to be executed in both Master\_one and Master\_two.

*Executed on master\_one:*

CHANGE MASTER

TO MASTER\_HOST='master\_two',

MASTER\_PORT=3306,

MASTER\_USER=’replica\_master1',

MASTER\_PASSWORD='password123',

MASTER\_LOG\_FILE='mysql-bin.000001',

MASTER\_LOG\_POS= 108470;

start slave;

*Executed in master\_two:*

CHANGE MASTER

TO MASTER\_HOST='master\_one',

MASTER\_PORT=3306,

MASTER\_USER='replica\_master2',

MASTER\_PASSWORD='password123',

MASTER\_LOG\_FILE='mysql-bin.000011',

MASTER\_LOG\_POS=690462;

start slave;

The details for the “change master command” in Master\_two were obtained from the mysqldump backup file, similar to the setup in the master-slave replication strategy. Once the command has been executed in Master\_two and the replication was set, we were then able to set up the replication going from Master\_two to Master\_one. The details from the change master command in Master\_one was obtained by executing ‘SHOW MASTER STATUS\G’ in Master\_two.

**3.3.3 Multi-Source Replication**

The final replication strategy that we implemented was the multi-source replication between Master\_one, Master\_two and Slave. Similar to how replication was set up in the master-slave replication strategy, two mysqldump files were created from the master servers for databases students and courses. Mysql users with replication slave privilege were also created in both Master\_one and Master\_two with the names slave1 and slave2 in order for the slave server to connect to both masters. The difference in setting up replication in the multi-source strategy is the way the ‘change master’ command is configured.[[13]](#endnote-13) Because the slave server has two masters, MySQL needs to be able to differentiate the two. The following is how the ‘change master’ commands were executed for the multi channel replication.

*Change master for replication between master\_one and slave:*

CHANGE MASTER

TO MASTER\_HOST='master\_one',

MASTER\_PORT=3306,

MASTER\_USER='slave1',

MASTER\_PASSWORD='password123',

MASTER\_LOG\_FILE='mysql-bin.000002',

MASTER\_LOG\_POS=;156

start slave;

*Change master for replication between master\_two and slave:*

CHANGE MASTER

TO MASTER\_HOST='master\_two',

MASTER\_PORT=3306,

MASTER\_USER='slave2',

MASTER\_PASSWORD='password123',

MASTER\_LOG\_FILE='mysql-bin.000012,

MASTER\_LOG\_POS= 155;

start slave;

**4 Testing**

Once replication was set up between master and slave servers (Fig.5), we were able to begin our testing.

Fig.5. Replication setup

**4.1 Test 1 - Data Insert In Master\_one**

We first tested by inserting data into Master\_one , and checked to see how the data behaves in both Master\_two and Slave. After inserting data into Master\_one, the data was now visible in the slave and Master\_two as shown by displaying the count number in the database. The following data was inserted into database Courses:

**Data Input:**

mysql> INSERT INTO Course VALUES ('3232','john','smith','course1');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Course VALUES ('2384','Anna','jane','course7');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Course VALUES ('2482','alex','tin','course8');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Course VALUES ('4343','Emily','young','course2');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Course VALUES ('2323','Bob','smith','course3');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Course VALUES ('3452','Sarah','john','course4');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Course VALUES ('2492','sally','lane','course5');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Course VALUES ('2940','justin','thomson','course6');

Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO Course VALUES ('2384','Anna','jane','course7');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Course VALUES ('2482','alex','tin','course8');

Query OK, 1 row affected (0.00 sec)

**Results:**

//Count in slave 1

//Before

mysql> select count(\*) from Course;

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

| count(\*) |

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

|        8 |

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

1 row in set (0.00 sec)

//After

mysql> select count(\*) from Course;

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

| count(\*) |

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

|       16 |

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

1 row in set (0.00 sec)

//Count in Master\_two

//Before

mysql> select count(\*) from Course;

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

| count(\*) |

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

|        8 |

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

1 row in set (0.00 sec)

//After

mysql> select count(\*) from Course;

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

| count(\*) |

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

|       16 |

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

1 row in set (0.00 sec)

**4.2 Test 2 - Data Insert In Slave**

Next, we tested what would happen if we inserted data in the slave. When we inserted data, the number of records in both the master servers did not change, as the data is not visible to be read by the masters. The following data had been inserted into the slave:

**Data Input:**

mysql> use Students

Database changed

mysql> INSERT INTO Names VALUES ('77876','Hannah','smith');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Names VALUES ('43225','Jennah','johnson');

Query OK, 1 row affected (0.00 sec)

**Result:**

//Count in slave

mysql> select count(\*) from Names;

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

| count(\*) |

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

|       11 |

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

1 row in set (0.00 sec)

//Count in both master\_one and master\_two

mysql> select count(\*) from Names;

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

| count(\*) |

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

|        9 |

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

1 row in set (0.00 sec)

**4.3 Test 3 - Data Insert in Master\_two**

Finally, we tested to see what would happen if we inserted the same data above again, but in the Master\_two server. Once inserted, the following error was displayed in the MySQL, and the replication as shown in the ‘SHOW SLAVE STATUS\G’ command is not running:

**Data Input:**

mysql> INSERT INTO Names VALUES ('77876','Hannah','smith');

ERROR 1062 (23000): Duplicate entry '77876' for key 'PRIMARY'

mysql> INSERT INTO Names VALUES ('43225','Jennah','johnson');

ERROR 1062 (23000): Duplicate entry '43225' for key 'PRIMARY'

**Result:**

Replication is broken in the slave as shown the following variables in the slave status:

          Slave\_IO\_Running: Yes

            Slave\_SQL\_Running: No

The error displays a duplicate entry on the primary key. In order to recover this, because it is only one duplicate entry, we are able to skip over this transaction as recorded in the binary logs. To do this we applied the SET GLOBAL SQL\_SLAVE\_SKIP\_COUNTER = 1 command in mysql and it fixed our issue. We can see replication is healthy again, as SLAVE\_IO\_RUNNING and SLAVE\_SQL\_RUNNING are showing as ‘yes’.

     Slave\_IO\_Running: Yes

            Slave\_SQL\_Running: Yes

**5 Discussion**

In order to set up replication, high level knowledge of networking systems, linux systems and DBMS configurations is important. We found that replication could be set up in a variety of ways. The type of setup chosen would depend on the clients needs and the most efficient way to satisfy those needs. Replication within a database needs to be dealt with a high level of accuracy and consistency in order to avoid costly replication breaks resulting in data inconsistency. This was demonstrated in our testing when we tried inserting into Master\_two and got an error as replication was broken in the slave. The replication broke due to a duplicate entry.

Our testing only investigated a small part of the replication strategies available in MySQL. In the future we hope to implement more complex replication strategies and test those in regards to recovery. We also hope to learn more about the configs in the MySQL configuration file to see how replication can be consumed even more to satisfy the needs of a business.

**6 Citations**

1. Bojovic, Ivan. “MySQL Master-Slave Replication on the Same Machine.” *Toptal Engineering Blog*, 11 Jan. 2016, [**www.toptal.com/mysql/mysql-master-slave-replication-tutorial**](http://www.toptal.com/mysql/mysql-master-slave-replication-tutorial)**.** [↑](#endnote-ref-1)
2. Severalnines. “MySQL Replication for High Availability - Tutorial.” Severalnines, 1 Oct. 2020, severalnines.com/resources/database-management-tutorials/mysql-replication-high-availability-tutorial. [↑](#endnote-ref-2)
3. “Synchronous vs Asynchronous Replication.” CloudBasic, 19 May 2020, cloudbasic.net/white-papers/synchronous-vs-asynchronous-replication/#:%7E:text=Most%20synchronous%20replication%20products%20write,written%20to%20the%20primary%20storage. [↑](#endnote-ref-3)
4. “---.” CloudBasic, 19 May 2020, cloudbasic.net/white-papers/synchronous-vs-asynchronous-replication/#:%7E:text=The%20benefits%20of%20asynchronous%20replication,tolerate%20some%20degradation%20in%20connectivity. [↑](#endnote-ref-4)
5. Severalnines. “MySQL Replication for High Availability - Tutorial.” Severalnines, 1 Oct. 2020, severalnines.com/resources/database-management-tutorials/mysql-replication-high-availability-tutorial. [↑](#endnote-ref-5)
6. <https://dev.mysql.com/doc/refman/5.7/en/replication-options-binary-log.html> [↑](#endnote-ref-6)
7. Sivakumar, S. “MySQL Master Slave Replication: 7 Easy Steps.” *Learn | Hevo*, 30 June 2020, hevodata.com/learn/mysql-master-slave-replication. [↑](#endnote-ref-7)
8. Garg, Shruti. “MySQL Master Master Replication: Easy Tutorial.” Learn | Hevo, 19 Aug. 2020, hevodata.com/learn/mysql-master-master-replication. [↑](#endnote-ref-8)
9. <https://dev.mysql.com/doc/refman/5.7/en/replication-multi-source.html> [↑](#endnote-ref-9)
10. Sicam, Jaime. “How To Fix MySQL Replication After an Incompatible DDL Command.” *Percona Database Performance Blog*, 7 Aug. 2019, www.percona.com/blog/2018/10/01/how-to-fix-mysql-replication-after-an-incompatible-ddl-command. [↑](#endnote-ref-10)
11. MashaMSFT. “Find Errors with Transactional Replication - SQL Server.” Microsoft Docs, 1 July 2020, docs.microsoft.com/en-us/sql/relational-databases/replication/troubleshoot-tran-repl-errors?view=sql-server-ver15. [↑](#endnote-ref-11)
12. <https://dev.mysql.com/doc/refman/5.6/en/set-global-sql-slave-skip-counter.html> [↑](#endnote-ref-12)
13. Ahmed, Ibrar. “Multi-Master Replication Solutions for PostgreSQL.” *Percona Database Performance Blog*, 9 June 2020, www.percona.com/blog/2020/06/09/multi-master-replication-solutions-for-postgresql. [↑](#endnote-ref-13)