

Utilising MySQL Replication – Part one.

Table of contents

Overview	2
1. Introduction to MySQL	2
2. Introduction to MySQL Replication	2
3. Setting up basic replication	3
3.1 Server Identification	4
3.2 Enable binary logging	4
3.3 Creating permissions	5
3.4 Setting up the slave	5
3.5 Starting and monitoring replication	6
3.6 Enabling/disabling replication for databases	7
4. Usage scenarios	8
4.1 Backups	8
4.2 Spreading the load	9
4.3 Master/Master, hot standby	10
5. Conclusion	11

Overview

This paper describes the principles of MySQL replication and how it can be used to further your data availability and reliability. MySQL is one of the most widely used relational databases on the internet, due to its low cost of entry, its ease of use, and its compelling enterprise features. This paper looks at implementing a replication set-up with MySQL version 5.0, and the different usage scenarios to achieve various end results.

1. Introduction to MySQL

Due to the rise of the interactive web over the last decade it is hard to imagine a web site that does not use a database of some kind. Whether it be for storing and retrieving the content of web pages, holding product information for an e-shop, or storing visitor details, databases step in to fulfil this need.

More often than not MySQL is the chosen database. The term "LAMP Stack" was coined a few years ago to describe the fundamental basics needed for a web server. The M stands for MySQL, that's how pervasive it is.

However, such is its widespread use that many users do not know some of the more advanced features it is capable of, or how to make them work for their solution. One of these features is replication, a tool with almost endless possibilities for your data.

2. Introduction to MySQL Replication

MySQL, like all SQL compliant databases, uses 'statements' to affect the data within the database. These can be reduced down to four essential statement types, typified by CRUD:

C - Create

R - Retrieve

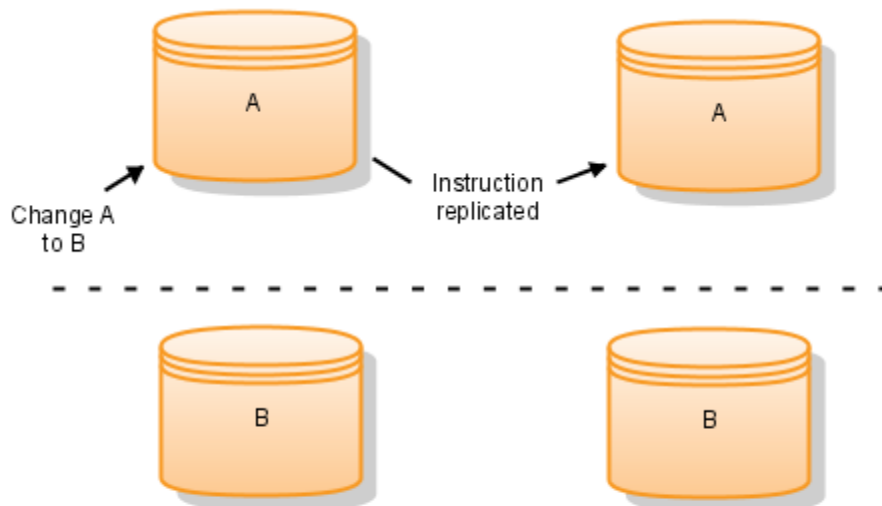
U - Update

D - Delete

All but a 'Retrieve' statement affects the data somehow, be it creating a new user on your portal, updating a customers telephone number, or deleting an old contact.

MySQL can be instructed to actually store these affecting statements in a log file, essentially as a list of instructions on how to repeat the same changes.

If two machines running MySQL had the same databases with the same data, and the first machine was instructed to record all its data changes in a log file, the second machine can be instructed to replay these from the first machine. Once the second machine has finished replaying these instructions, both servers will once again have the same databases with the same data.



The server generating this log file, known as the binary log, is referred to as the replication Master. All machines that replay another machine's binary log is known as a replication Slave. As you will see, these terms are contextual, as you may have machines that are both replication masters and replication slaves at the same time.

3. Setting up basic replication



We will now look at how to set up a basic Master Slave relationship between two MySQL servers.

3.1 Server Identification

As these two machines will be working together on the network, we need to give them some form of identification for them to recognise each other.

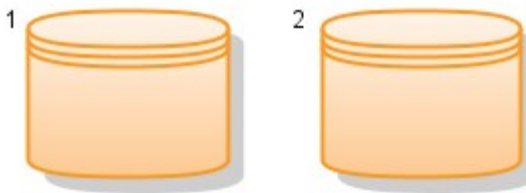
Within the `[mysqld]` section of the Master's mysql configuration file, add the line:

```
server-id = 1
report-host = mysql1
```

Within the `[mysqld]` section of the Slave's MySQL configuration file, add the line:

```
server-id = 2
report-host = mysql2
```

These can be any numerical values, though as you see we've adopted a simple approach of 1 & 2 to label our two servers.



3.2 Enable Binary Logging

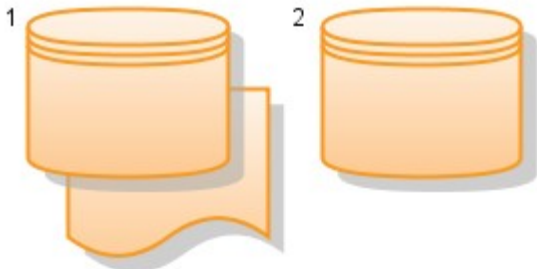
The next step is to start the master server logging statements.

Within the `[mysqld]` section of the Master's MySQL configuration file, add the line:

```
log_bin
```

This creates a binary log file in the default location on your server. You can change this location by adding "`= <LOCATION>`" to the end of that line, where `<LOCATION>` is the specified place for the log file.

If you restart the MySQL service now, it will start storing transactions to that log.

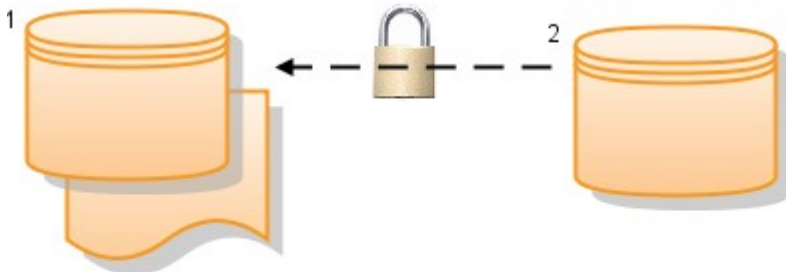


3.3 Creating the permissions

Before the Slave can connect to the Master, we need to tell the Master to allow connections for this purpose. This can be done by issuing the following statements within the Master's MySQL shell.

```
GRANT SLAVE PERMS  
FLUSH PRIVILEGES;
```

We now have user credentials that we can use to connect to the Master.



3.4 Setting up the Slave

The next step is to get an exact replica of the data on the Master. As the binary log stores transactions, the slave still needs the same starting data to apply these transactions to. As such, we need to get a copy of the data from the Master and on to the Slave prior to starting replication.

The easiest method of doing this is with the ``mysqldump`` command. This creates a plain text version of the database in statement format. Like the binary log, this dump file can be replayed on the Slave which in turn recreates the data of the Master perfectly on the Slave.

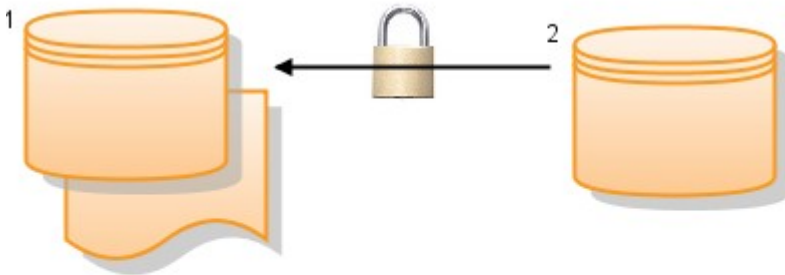
Once this is done, we should have two identical datasets on the Master and the Slave.

The next thing is to find out from the Master which is the current log file being used to store the transactions, and the position in the log file of where to start. This can be found by running the command ``SHOW MASTER STATUS;`` within the Master's MySQL shell.

Obtaining the dump file and the position at the time of the dump can be done in one step, by using the ``--master-data`` option on the ``mysqldump`` command:

```
mysqldump --master-data=2
```

This will store the log file and position of the master at the exact time that the dump is taken. Simply open the file and look in the first few lines to find this.



3.5 Starting and monitoring replication

Everything is now ready to start replication between the Master and the Slave. We have the following information:

Host (the network address of the Master server)

Username (the username to use for the replication connection)

Password (the password to use for the replication connection)

Logfile (the binary log file to read transactions from)

Logposition (the current position within the logfile)

We can now fill out the following statement to run on the Slave server:

```
CHANGE MASTER TO MASTER_HOST='Host',  
MASTER_USER='Username',  
MASTER_PASSWORD='Password',  
MASTER_LOG_FILE='Logfile',  
MASTER_LOG_POS=Logposition;
```

6

This provides the slave with all the information it needs to start replicating.

All that remains now is to instruct the Slave server to start replicating. This is done by running the following statement in the Slave server's MySQL shell:

```
START SLAVE;
```

And replication should now be working.



You can check on the status of replication by running the following command in the Slave's MySQL shell:

```
SHOW SLAVE STATUS;
```

This outputs all the variables that reflect the replication status. You can use this command to see if there are any problems, or if replication is broken.

3.6 Enabling/disabling replication for databases

This set-up will replicate all statements on the Master over to the Slave. It is possible to have a finer grained control over what exactly is replicated.

In the Master's configuration file, use the directives:

```
binlog_do_db = database_name  
binlog_ignore_db = database_name
```

to create a fine grained filter of databases to log transactions for.

Alternatively, use:

```
replicate_do_table = database_name  
replicate_ignore_table = database_name
```

on the Slave server to determine if transactions for a named database should be ignored, or if they should be replicated.

It is combinations of these filters that allow some of the more complex Usage Scenarios outlined next in this document.

4 Usage scenarios

It is helpful to identify how this replication is used in the real world and the benefits associated with it.

4.1 Backups

Getting good, coherent backups from a database can be troublesome. A relation database like MySQL contains relation data - data that contains integral relationships to other data in the database. A backup starts copying data from the beginning of a database, and works through to the end. If a database is written to when the backup is only halfway through, the writes could take place all over the database, but that won't be captured by the backup which has already copied half the database.

As such, you end up with no data integrity - essentially, you have a corrupt backup.

The option is to 'lock' the database for the duration of the backup, stopping all writes. This may not be a problem for small datasets, but for large datasets this process can take any number of minutes, sometimes hours; and all that time your database cannot receive any writes.

These kind of backups are often attempted in the middle of the night when use of the database is low, but that's not an option for those that require 24 hour access to the database. Also, it means that only one backup a day is taken. If something went wrong, an entire days worth of data would be lost.

Instead, a Slave server could be set up, and all backups can be taken from the Slave, as frequently as was needed. The Slave can be instructed to temporarily stop replicating, the database can be backed up, then the slave started again. The replication process is intelligent enough to know where it last stopped, and will resume from that point, meaning your slave can catch up to the Master server as soon as it physically can, ready to take the next backup.

ForLinux provides various backup solutions for your entire business, as well as specialist tools for MySQL backups, including fully coherent backups of a live machine in a matter of seconds.

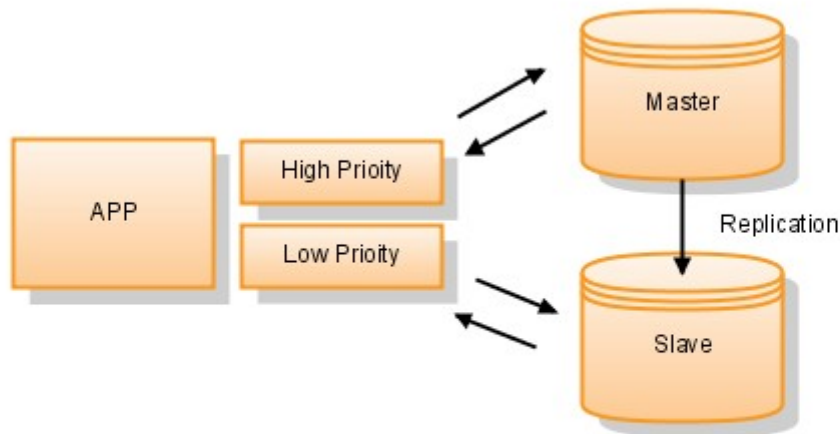
4.2 Spreading the load

Demands on a database can often lead to it being under strain, and demand invariably always increases, increasing the strain until the database is virtually useless. Often the solution is thought to be buying more expensive and exotic hardware to deal with the problem.

Instead, you could split the database demands at their fundamental basics: 'reads' and 'writes'.

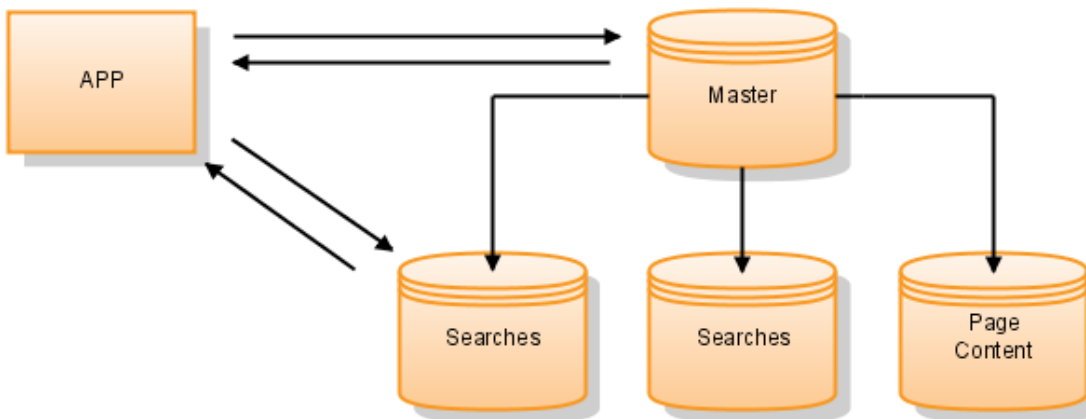
By replicating databases to another machine, you can then run your 'reads' on the Slave server, reducing demand on the Master to cope with the 'writes'.

Imagine the case of a web shop; A single database machine dealing with all the user accounts and purchases, updates to the product catalogue, the general page content, and the searches of the catalogue by the users.



As performance suffers due to increased use, the product and page content databases could be replicated to a second machine. You could then run all the general page content queries and the more intensive product searches on the Slave, freeing up the Master to deal quickly with the user accounting information.

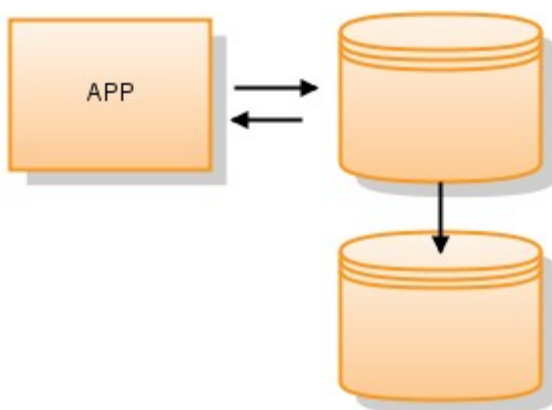
Of course, you may find that the demands on the slave end up being too much after a duration; so why not add another slave to split the searches over, and a third slave dedicated to the page content.

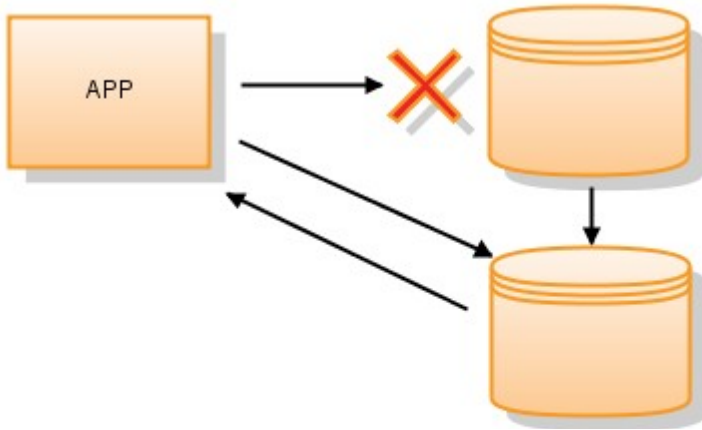


In the end, there is no end to the possibilities of how these approaches can be combined. It is all about what works for you and your data. ForLinux provides a consultation service to help you achieve an efficient combination of Masters and Slaves for the maximum use of your data.

4.3 Master/Master, hot standby

Another use for a slave would be a 'hot standby'. In the event of hardware failure on the Master, all queries could be sent to the Slave, providing an almost seamless fail-over process that keeps your web service online and working.





ForLinux provides an automated High-Availability solution that utilises this method to keep your databases online at all times.

5. Conclusion

This document has shown a few methods of utilising MySQL replication to enhance and increase your data availability, reduce load issues for strained databases, and even achieve High-Availability. The possibilities are limited only by your needs and technical ability. For further information or support with any aspect of MySQL please contact ForLinux.