5

MySQL Database Administration

Introduction

For the most part, MySQL is low maintenance software. Once you have it installed and set up, there aren't a lot of administrative demands. Nonetheless, it is not maintenance free. This chapter provides an overview of the most typical administrative tasks. These range from configuring your server, to backing it up and running periodic maintenance on it.

MySQL Configuration

As an administrator, you will need to understand how the MySQL server and clients are configured and how to modify that configuration.

The configuration of mysqld, the MySQL server, is controlled from the command line or from one or more options files. The options files simply provide an convenient way for you to specify command line options. The full set of options for the mysqld server and other utilities are documented in Chapter 20 – MySQL Programs and Utilities.

An options file might look like this:

```
# Example mysql options file.
#
# These options go to all clients
[client]

password       = my_password
port            = 3306
socket          = /var/lib/mysql/mysql.sock
```



```
# These options go to the mysqld server
[mysqld]
port            = 3306
socket          = /var/lib/mysql/mysql.sock
skip-locking
set-variable    = max_allowed_packet=1M
```

These seems straightforward, but lets spend a few moments to dissect this file. The first three lines look like:

```
# Example mysql options file.
#
# These options go to all clients
```

Lines starting with the pound (#) character are comment lines, and are ignored. You may also use a semi-colon (;) to indicate a comment.

The next line,

```
[client]
```

is puzzling. This specifies a "group". All options following this line apply to the group mentioned. In our case, we have specified that the following options apply to the "client" group, meaning that the following options will apply to all MySQL client programs. Let look at the next section. The line

```
password       = my_password
```

is equivalent to the command line option `--password=my_password`. Since we specified a group of `client`, this option will be passed to all client programs.

> Specifying the password in the client group is an ideal way to specify your password to all clients so you don't have to type it every time you connect. However, if you do this, make some the option file is not readable by others.

In general, any command line option `--option=value` can be specified with a line in the format `option=value`. The next two lines

```
port            = 3306
socket          = /var/lib/mysql/mysql.sock
```

provide two more examples of this. Next we encounter

```
# These options go to the mysqld server
[mysqld]
```

which specifies a group of `mysqld`. All options following this line will apply only to the `mysqld` program, and no others. A group specification continues to apply until the end of the file is reached or another group is encountered. Following this we have some more option settings which look similar to the client section. However, there are a few differences worth mentioning. The line

```
skip-locking
```


is an option even though it is not in the `option=value` format. This is equivalent to the command line option `--skip-locking`. So, any command line option `--option` can be specified by simply listing the option name.

What about this line?

```
set-variable    = max_allowed_packet=1M
```

This is the same as specifying `--set-variable max_allowed_packet=1M` on the command line. This is the format you use to set a mysqld variable. (See Chapter for more information on system variables.)

So know you know how to set up a configuration file for MySQL. But once you've created one, how does MySQL find it? On UNIX, MySQL reads options from the following locations:

1. `/etc/my.cnf`
   This is the global options file. This is read by all MySQL installations on this host.
2. `DATADIR/my.cnf`
   This provides a server specific options file. DATADIR is the MySQL data directory.
3. `--defaults-extra-file`
   A command line option specifying an additional options file to be read.
4. `$HOME/.my.cnf`
   A user specific options file. This is particularly useful for customizing the operation of clients.

On Windows, the files are read from

1. `My.ini` in the windows system folder
2. `C:\my.cnf`
3. `C:\mysql\data\my.cnf`
4. `--defaults-extra-file`

Option files are read in the order specified above. If any option appears in multiple files, the last one takes precedence over the others. Options specified on the command line take precedence over options specified in an option file. Using this scheme, you can provide a default behavior in `/etc/my.cnf` that can be overridden for a particular server or by an individual user.

Some options can be specified using Environment variables. In this case, options specified in an option file or on the command line take precedence over the Environment variables. See Chapter 18 for more information on MySQL Environment variables.

Starting and Stopping the Server

One of the first things you'll want to do as administrator is to get MySQL set up so that it will automatically start when the OS boots, and shutdown cleanly along with the OS. On UNIX, you have a couple of options: the mysql.server script or safe_mysqld.

mysql.server

The `mysql.server` script is intended for use with the SVR4 style startup/shutdown mechanism. It is available in the `support-files` directory of your installation (usually `/usr/local/mysql/support-files`). When properly installed, it will automatically start and stop MySQL for you along with the OS.

> If you installed MySQL on Linux using the RPM package, `mysql.server` may have already been installed on your system. The RPM installer renames `mysql.server` to `mysql` when it copies it to `/etc/rc.d/init.d`. If the file `/etc/rc.d/init.d/mysql` is already there, you are already set up to automatically start and stop MySQL.

The procedure for installing mysql.server on a RedHat Linux system is as follows (you must be logged in as the `root` user to do this):

```
# cp mysql.server /etc/rc.d/init.d
```

This copies the the mysql.server into the init.d location.

```
# ln -s /etc/rc.d/init.d/mysql.server /etc/rc.d/rc3.d/S99mysql
```

This sets it up so that you'll start the MySQL server when you enter run level three, which corresponds to the directory `rc3.d`. Run level three is typically used for full multi-user mode. If your system uses a different run level for multi-user mode, you need to link that directory instead.

```
# ln -s /etc/rc.d/init.d/mysql.server /etc/rc.d/rc0.d/S01mysql
```

Run level zero (`rc0.d`) is system halt. Scripts linked here will be called when the system shuts down. Now you are all set. You may want to test it by rebooting your server.

The above example assumes RedHat Linux as the target system. The location of the init files varies depending upon which UNIX you use. For example, Solaris places them in `/etc/init.d`. You'll need to tailor the commands to the location of the init files on your system.

safe_msyqld

On a non-SVR4 UNIX system, you can use `safe_mysqld` to automatically start MySQL at boot. `safe_mysqld` is found in the bin directory of your MySQL installation (usually `/usr/local/mysql/bin`).

In order to use it, you'll need to figure out how your system starts processes at boot time. Often, there is an `/etc/rc.local` file which can be modified to call `safe_mysqld`. If you are unsure what file to modify, talk to your system administrator to get help. Once you have identified the file, adding a section to invoke `safe_mysqld`.

It is also possible to start the MySQL server by directly invoking mysqld. This is NOT recommended however. Safe_mysqld and mysql.server were designed for this purpose and have a number of advantages, such as:

- determining the location of the server and invoking it with the right options
- logging run-time information to a log file
- monitoring the server and restarting it if necessary

Windows NT/2000

You can automatically start and stop the MySQL server under Windows NT or 2000 if it is installed as an NT Service. To do this, open up a MS-DOS window, and type

```
C:\mysql\bin\mysqld-nt --install
```

This is covered in greater detail in Chapter 3 – Installation. Please refer to that chapter if you need more information about installing MySQL as a service.

Once MySQL is installed as a service, it can be controlled like any other NT service.

Log Files

The MySQL server can produce a number of log files that you might find helpful:

- Error Log
- Query Log
- Binary Log
- Slow Query Log

By default, the log files are written to the data directory. The details of these are each outlined below.

Another Log file is available: The update Log. This has been obsoleted by the Binary log. If you are still using the update log (invoked by

option `--log-update`), we recommend that you switch to that format instead.

The Error Log

The Error Log contains the redirected output from the safe_mysqld script. On UNIX, it is a file called *`hostname`*`.err`. In windows, it is called `mysql.err`.

This file contains an entry for everytime the server is started and stopped, including an entry for every time the server is restarted because the server died. Critical errors and warnings about tables that need to be checked or repaired also appear here.

The Query Log

The query log contains the details of all connections and queries to the MySQL server. This can be useful for debugging a client application. It will log the SQL commands exactly as they are received by the server.

The query log can be enabled using the `--log[=file]` option. If no filename is given, it defaults to hostname.log. If no directory is given, it defaults to the data directory.

The Binary Log

The binary log writes contains all SQL commands that update data. Only statements that actually change data are logged. So for example, if I perform a delete that doesn't affect any rows, it will not be logged. Update statements that set a column to the same value are not logged either. Updates are logged in execution order.

The binary log is very useful for journaling transactions since the last backup. If, for example, you backup your database once per day, and your database crashes in the middle of the day. You can restore the database up the last completed transaction, by

1. Restoring the database (see the section in this chapter for more information on database backup and restore)
2. Applying the transactions from all binary logs since the last backup

The binary log can be enabled using the `-log-bin[=file]` option. If no filename is provided, it defaults to *`hostname`*`-bin`. If no directory is geiven, it defaults to the data directory. MySQL appends a numeric index to the filename, so the actual filename ends up being *`hostname`*`-bin.`*`number`*. The index is used for rotating the files. MySQL will rotate to the next index:

- When the server is restarted
- When the server is refreshed (with mysqladmin refresh)

- When the logs are flushed (with mysqladmin flush-logs, or SQL "FLUSH LOGS")

MySQL also creates an index file which contains a list of all used binary log files. By default this is named hostname-bin.index. If you wish, you may specify the name and or location of the index file with the `--log-bin-index[=file]` option.

In order to read a binary log, you'll need a utility called `mysqlbinlog`. The example below illustrates the workings of the binary log. Assume we've started MySQL on a host called `odin` and specified `log-bin` in our global configuration file, `/etc/my.cnf`. In the data directory, we'll see:

```
$ cd /usr/local/mysql/data
$ ls -l
.
.
-rw-rw----   1 mysql    mysql          73 Aug  5 17:06 odin-bin.001
-rw-rw----   1 mysql    mysql          15 Aug  5 17:06 odin-bin.index
.
.
```

If we inspect `odin-bin.index`, we see

```
$ cat odin-bin.index
./odin-bin.001
$
```

Lets use `myslbinlog` to read the binary log:

```
$ mysqlbinlog odin-bin.001
# at 4
#010805 17:06:00 server id  1   Start: binlog v 1, server v 3.23.40-log created
010805 17:06:00
$
```

After we do an update to a database, and look again, we get some more information.

```
$ mysql
mysql> use test
.
.
mysql> insert into test (object_id, object_title) values (1, "test");
Query OK, 1 row affected (0.02 sec)
mysql> quit
Bye
$ mysqlbinlog odin-bin.001
# at 4
#010805 17:06:00 server id  1   Start: binlog v 1, server v 3.23.40-log created
010805 17:06:00
# at 73
#010805 17:39:38 server id  1   Query   thread_id=2     exec_time=0
error_code=0
use test;
SET TIMESTAMP=997058378;
insert into test (object_id, object_title) values (1, "test");
$
```

Now, lets flush the logs to see what happens.

```
$ mysqladmin -uroot -pblueshoes flush-logs
```

We now have new file called `odin-bin.002` and our index file has been updated.


```
$ ls -l
.
.
-rw-rw----   1 mysql    mysql         203 Aug  5 17:45 odin-bin.001
-rw-rw----   1 mysql    mysql          73 Aug  5 17:45 odin-bin.002
-rw-rw----   1 mysql    mysql          30 Aug  5 17:45 odin-bin.index
.
.
.
$ cat odin-bin.index
./odin-bin.001
./odin-bin.002
$
```

The binary logs can be played back into a server by piping the output from mysqlbinlog to a mysql session. For example

```
$ mysqlbinlog odin-bin.001 | mysql ...
```

There are a few other options you can use to control the binary logs. The option `--binlog-do-db=dbname` tells MySQL to only log updates for the specified database. The option `--binlog-ignore-db=dbname` tells MySQL to ignore the specified database for the purposes of binary logging.

The Slow Query Log

The slow query log contains all SQL commands that took longer than the variable long_query_time This can be used to identify problem queries, and expose parts of your database or application that need tuning.

The slow query log is enabled with the `-log-slow-queries[=file]` option. If no filename is provided, it defaults to *`hostname`*`-slow.log`. If no directory is given, it defaults to the data directory. The long_query_time can be set using the `--set-variable long_query_time=time` query (`time` is specified in seconds).

Log Rotation

No matter which log files you choose to enable, you'll have to worry about maintaining them so they don't fill up a file system.

If you are running RedHat Linux, you can use `mysql-log-rotate` for this. It can be found in the support-files directory of your installation. This uses the logrotate utility to automatically rotate your error log for you. For more information on logrotate, read the man page or refer to your RedHat documentation.

To install `mysql-log-rotate` on Linux, simply copy it to `/etc/logrotate.d`. You may wish to edit the script to rotate other logs that you have enabled as well. By default it only rotates the query log.

If you installed MySQL on Linux using the RPM package, `mysql-log-rotate` may have already been installed on your system. The

RPM installer renames `mysql-log-rotate` to `mysql` when it copies it to `/etc/logrotate.d/`. If the file `/etc/logrotate.d/mysql` is already there, it has already been installed.

On systems other than RedHat, you will have to devise you own scripts for rotating the logs. Depending on which logs you have enabled, and how you want them located, the scripts could range from very simple to very complex. In general, the procedure is to copy the logfile(s) out of the way, and use `mysqladmin flush-logs`.

Unfortunately, none of these techniques work for the error log. Since it is written from the safe_mysql script, the flush logs command does not flush it. Also note that safe_mysqld will continue to append to it on successive restarts. You may want to modify the startup or shutdown scripts for your MySQL server to maintain the error log.

Database Backup

A good backup strategy is by far the most important thing you can develop as an administrator. In particular, you'll be really glad you have good backups if you ever have a system crash and need to restore your databases with as little data loss as possible. Also if you ever accidentally delete a table or destroy a database, those backups will come in very handy.

Every site is different, so it is very difficult to give specific recommendations on what you should. You need to think about your installation and your needs. In this section, we present some general backup principles that you can adopt, and we cover the technical details of performing the backups. You will have to turn this information into a coherent strategy for your installation.

In general there are a number of backups:

- Store your backups on a different device (either on another disk or perhaps a tape device) than the database, if possible. If your disk crashes, you'll be really happy to have the backups in a different place. If you are doing binary logging, store the binary logs with the backups.
- Make sure you have enough disk space for the backups to complete.
- Use binary logging, if appropriate, so you can restore your database with minimal loss of data. If you choose not to use binary logging, you will only be able to recover your database to the point of your last backup. Depending upon your application, a backup without binary logs might be useless.
- Keep an adequate number of archived backups
- Test your backups, before an emergency occurs

mysqldump

Mysqldump is the MySQL utility provided for dumping databases. It basically generates an SQL script containing the commands (CREATE TABLE, INSERT, etc.) necessary to rebuild the database from scratch. The main advantages of this approach over direct copy (mysqlhotcopy) is that output is in a portable ASCII format which can be used across hardware and operating system to rebuild a database. Also since the output is a SQL script, it is possible to recover individual tables.

To use mysqldump to backup your database,we recommend that you use the `–opt` option. This turns on `–quick, --add-drop-table, --add-locks , --extended-insert ,` and `--lock-tables`. This should give you the fastest possible dump of your database.

> Be aware that this locks all the tables, so your database will essentially be offline while you are doing this.

So your command will look something like this:

```
$ mysqldump --opt test > /usr/backups/testdb
```

If you are using binary logging, you will also want to specify `--flush-logs`, so the binary logs get checkpointed at the time of the backup.

```
$ mysqldump --flush-logs --opt test > /usr/backups/testdb
```

mysqldump has a number of other options that you can use to customize your backup. For a list of all the options available for mysqldump, type mysqldump –help, or refer to Chapter 20, MySQL Programs and Utilities.

mysqlhotcopy

mysqlhotcopy is a perl script that uses a combination of LOCK TABLES, FLUSH TABLES and cp to perform a fast backup of the database. It simply copies the raw database files database files to another location. Since it is only doing a file copy, it is much faster than mysqldump. But, since the copy is in native format, the backup is not portable to other hardware or operating systems. Also, mysqlhotcopy can only be run on the same host as the database, whereas mysqldump can be executed remotely.

To run, mysqlhotcopy, type

```
$ mysqlhotcopy test /usr/backups
```

This will create a new directory in the /usr/backups directory which has a copy of all the datafiles in your database.

If you are using binary logging, you will also want to specify `--flushlog`, so the binary logs get checkpointed at the time of the backup.

Database Recovery

Individual recovery scenarios vary widely, ranging from disk hardware failures to corrupted data files to accidentally dropped tables, and many points in between. In this section, we provide a general overview of recovery procedures.

In general, you need two things to perform a database recovery: your backup files and you're your binary logs. In general, performing a recovery consists of

- Restoring the database from the last backup
- Applying the binary logs to bring the system completely up to date

If you don't have binary logging enabled, the best you will be able to do is to restore the system to the last full backup.

Recovering from mysqldump files

Assume for this example, we are recovering a database named test.

Bring up the mysql server and reload the database using the mysqldump files.

```
$ cat test.dump | mysql
```

This will bring the database back to the state it was at the last backup.

Apply the binary logs to bring the system up to date. Use the `-one-database` mysql option to filter out SQL commands that apply to other databases. You only want to apply the binary logs that were created since your last backup. For each binary log file, type

```
$ mysqlbinlog host-bin.xxx | mysql --one-database=testdb
```

> Sometimes you will need to massage the output from the mysqlbinlog program before sending it through mysql. If you are recovering from a mistaken drop table statement, for example, you will need to remove this from the output of mysqlbinlog, otherwise you'll drop the table again!

Recovering from mysqlhotcopy files

Reload the database by copying the database files from the backup location to the mysql data location. Make sure the mysql server is down when you do this. Assume for this example, the database is backed up in /var/backup/test and the mysql data location is /usr/local/mysql/data.

```
$ cp -r /var/backup/test /usr/local/mysql/data
```

This will bring the database back to the state it was at the last backup.

Now, bring the mysql server up and apply the binary logs to bring the system up to date. Refer to the previous section for an example of how to do this.

Table Maintenance and Crash Recovery

Database tables can get out of whack when a write to the data file is not complete for some reason. This can happen due to a variety of reasons, such as a power failure or a non-graceful shutdown of the MySQL server.

MySQL provides two mechanisms for detecting for and repair table errors: `myisamchk/isamchk` and `mysqlcheck`.. It is a wise practice to perform these checks regularly. Early detection may increase your chances of successfully recovering from errors.

Mysqlcheck is new with version 3.23.38 of MySQL. The main difference between myisamchk/isamchk and mysqlcheck is that mysqlcheck allows you to check or repair tables while the server is running. Myisamchk/isamchk require that the server not be running.

Checking a table

If you suspect errors on a table, the first thing you'll want to do is use one of the utilities to check it out.

Myisamchk and isamchk are quite similar. They provide the exact same functions. The only difference is that myisamchk is used on MyISAM tables, and isamchk is used on ISAM tables.

Mysqlcheck can only be used with MyISAM tables.

You can tell what kind of table you are dealing with by looking at the extension of the data file. An extension of ".MYI" tells you it is MyISAM table and. ".ISM" indicates an ISAM table. So, myisamchk is only used with .MYI files, and isamchk with .ISM files. Simple enough?

Lets assume we have a database called test with two tables, `table1` which is an ISAM table, and `table2` which is a MyISAM table.

The first step is to check your table, using the appropriate utility. If you are using myisamchk or isamchk, make sure you MySQL server is not running to prevent the server from writing to the file while you are reading it.

```
$ myisamchk table2.MYI
Data records:       0   Deleted blocks:       0
- check file-size
- check key delete-chain
- check record delete-chain
- check index reference
```



```
$ isamchk table1.ISM
Checking ISAM file: table1.ISM
Data records:       0   Deleted blocks:       0
- check file-size
- check delete-chain
- check index reference
$ mysqlcheck table2.MYI
test.table2                                        OK
```

The default method is usually adequate for detecting errors. However, if no errors are reported but you still suspect damage, you can perform and extended check using the `--extend-check` option with `myisamchk/isamchk` or the `--extend option` with `mysqlcheck`. This will take a long time, but is very thorough. If the extended check doesn't report any errors, you are in good shape.

Repairing a table

If the check reported errors on a table, you can try to repair them.

If you are using myisamchk or isamchk, make sure your MySQL server is not running when you attempt the repair. Also, it is a good idea to back up the data files before attempting a repair operation, in case something goes haywire.

With myisamchk/isamchk, you want to first try the `--recover` option.

```
$ isamchk --recover table1.ISM
$ myisamchk -recover table2.MYI
```

If this fails for some reason, they you can try `--safe-recover`, a slower recovery method which can fix a some errors `--recover` cannot..

```
$ isamchk --safe-recover table1.ISM
$ myisamchk --safe-recover table2.MYI
```

With `mysqlcheck`,your only recovery option is `--repair.`

```
$ mysqlcheck --repair test table2
```

If these operations fail, your only remaining option is restore the table from your back ups and binary logs. See the section on Database Backup and Recovery for more information about this.

Scheduled Table Checking

We recommend that you take steps to perform regularly schedule table checks on your database files. This can be done by wrapping `isamchk/myisamchk/mysqlcheck` commands into a script that is executed periodically from `cron` or some other scheduling software.

You also may want to modify your system boot procedure to check tables at boot time. This is especially useful if the system is rebooting after a system crash.

