**MySQL Server logs**

* Error Log
* General
* Slow Log
* Binary Log
* Relay Log

The Error Log

mysql> SHOW VARIABLES LIKE 'log\_error';

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

| Variable\_name | Value |

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

| log\_error | /var/log/mysql/error.log |

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

1 row in set (0.00 sec)

The General Query Log

The general query log is another useful log because it (surprise, surprise!) keeps track

of every query sent to the server by a client. It also displays details about which clients

are connected to the server and what these clients are doing.

MYSQL> SHOW VARIABLES LIKE '%GENERAL%';

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

| VARIABLE\_NAME | VALUE |

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

| GENERAL\_LOG | OFF |

| GENERAL\_LOG\_FILE | /VAR/LIB/MYSQL/DBA-VIRTUAL-MACHINE.LOG |

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

2 ROWS IN SET (0.00 SEC)

The Slow Query Log

A related log is the slow query log, which lists all the queries that exceed a predefined

amount of time (specified by the long\_query\_time variable). Any query that takes longer

than this value is listed in this log. If you’re looking for a way to optimize performance,

this log is a good place to start.

Typically, you would look at the queries in this log as candidates for revision to lessen

the impact on your server’s performance. Remember, though, the length of time a query

takes can be the result of factors other than poorly written code. Queries that usually run

under the “long” threshold can appear in this log if the server is tied up elsewhere.

The slow query log is activated by using the --slow-query-log option at server

mysql> show variables like 'slow\_query%';

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

| Variable\_name | Value |

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

| slow\_query\_log | OFF |

| slow\_query\_log\_file | /var/lib/mysql/u16\_master-slow.log |

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

2 rows in set (0.01 sec)

The Binary Log

MySQL 3.23.14 and later also support logging of all the commands that make changes

to a table’s data. Commands such as INSERT, REPLACE, DELETE, GRANT, and REVOKE,

along with UPDATE, CREATE TABLE, and DROP TABLE are all in this category.

This information is stored in a binary log, which provides a more efficient storage format

for data and also records a larger amount of information.

A utility named mysqlbinlog converts the binary log back to text so you can read it. The binary log can be activated by using the --log-bin option when starting MySQL, as shown:

[root@host]# **/usr/local/mysql/bin/mysqld\_safe --log-bin**

The default filename for the log is hostname-bin, with the file extension containing a

number identifying the log in the sequence. You can specify a different location for the

binary log by passing it to the --log-bin option as an argument.

**Tip:-** Because the binary log is critical for crash recovery, it’s always a good idea to save it to

a different drive or device than the one which holds the MySQL database files.

**Note** Versions of MySQL prior to MySQL 5.0 used a more primitive version of the binary log,

the “update log,” which recorded all the queries that changed a table’s data. Statements such

as INSERT, REPLACE, DELETE, GRANT, and REVOKE, along with UPDATE, CREATE

TABLE, and DROP TABLE, were all recorded in this log. However, this update log is no

longer supported in MySQL 5.0 and later, and is instead replaced by the binary log.

Why Would I Need to Use the Binary Log?

Updates that are part of a transaction are not executed immediately; they are kept

in a cache until the transaction is committed. Once a COMMIT command is received

by the MySQL server, the entire transaction is first written to the binary log, and

then the changes are saved to the database. If a part of the transaction fails for

whatever reason, the whole transaction is rolled back and no changes are written

to the binary log. Also, if you’re setting up master and slave servers for replication,

you must enable the binary log.

PART II

To refresh the logs, use the FLUSH LOGS command. This command causes the

server to close and then reopen the log files. For the binary logs, this command closes

the current log and creates a new log with a new sequence number so the old one can

be archived, if desired. The value of log flushing becomes more evident when you consider issues of log Rotation.

**Rotating Logs:**

Setting Up Criteria For Log Rotation

As an example, let’s have some criteria for managing general MySQL query logs. We can come up with a suitable set of criteria for log management by asking the following questions:

Q: What is the maximum size that the log file can grow?

A: Let’s say it can grow up to 300 MB after which it needs to be rotated and compressed.

Q: What is the frequency that you want the log file to be rotated?

A: We can say that we want logs to be rotated on a daily basis.

Q: How many old log files you want to retain?

A: We would like to retain the last 30 log files.

Based on the above criteria, the overall disk space required for general query log management is about 1.2 GB. Assuming a 90% compression ratio – we will have 30 compressed log files of size 30 MB each and a live log file of about 300 MB.

Managing The Logs Using Linux logrotate Utility

logrotate is a Linux utility that helps with the efficient administration of log files and provides options for automatic rotation, compression, and removal of log files. The criteria established above can be configured for logrotate utility by creating a configuration file in the /etc/logrotate.d folder.

Let’s call this configuration file mysqlgeneral and the contents of the file will be:

/var/log/mysql/general.log{

compress

dateext

maxsize 300M

copytruncate

maxage 365

dateformat -%Y%m%d%s

daily

rotate 30

notifempty

}

With the above options for logrotate, the general query logs get rotated either on a daily basis or when the log file size exceeds 300 MB. The old logs are compressed and 30 such files will be preserved. Log rotation will be skipped if the log file is empty due to the setting ‘notifempty’.

The ‘copytruncate’ option is to ensure that current log file is never deleted during rotation and only its contents get truncated. This is important since some applications expect that the log file is always available and it’s not possible to delete the log without stopping the application first.

Now that the log rotation configuration is set for the general query log, the logrotate utility has to be run so that the above configuration is executed. This is typically done through a cron job. We can set this to be running every hour by placing the logrotate script in /etc/cron.hourly directory:

#!/bin/sh

/usr/sbin/logrotate /etc/logrotate.conf

EXITVALUE=$?

if [ $EXITVALUE != 0 ]; then

/usr/bin/logger -t logrotate "ALERT exited abnormally with [$EXITVALUE]"

fi

exit 0