WEBYOG®

MySQL Whitepaper

7 MOST IMPORTANT MYSQL TUNING CHECKS

BY PINAL DAVE

INTRODUCTION

MySQL is the most popular and powerful open-source database. As applications become more data-dependent, the database application resources are often under pressure to perform at their limits. Every Relational Database Management System (RDBMS) performs better when it is configured optimally, so it is important to consider the relational database platform and potential workload.

This white paper focuses on seven of the most important tuning configurations for MySQL. After reading this paper, DBAs and businesses will better understand how to drive efficiency and maximize their investment. Let's start by exploring three common configuration challenges for database administrators.

Another obstacle faced by DBAs and the businesses they work with is cost. An unlimited budget is not a reality for most companies, and the common approach is to do more with less. After all, driving operational efficiencies is simply practical business sense. Increasing the efficiency of your database performance costs significantly less than upgrading hardware or adding more memory.

The final challenge we will examine is the lack of a systematic process for writing efficient queries. All too often, queries run faster on the developer's machine than it does on the production machine. As a result, the DBA must spend time investigating hundreds of queries running simultaneously to find the culprit.

Many years ago, one of the biggest debates was about MySQL Engine selection. Thankfully, MySQL selected InnoDB as the default engine for MySQL version 5.5 which settled the debate. Additional storage engines are available such as the 'CSV' and the 'ARCHIVE' storage engines that ship with the server. 'CSV' should only be used in special cases. Recent InnoDB servers support compression, however, there is little need to use the 'ARCHIVE' engine today, as you can compress InnoDB tables that are rarely accessed in order to save disk space. Over the years, several third-party storage engines were developed, but most of them did not mature before being discontinued. There is an exception to this: the MyRocks engine based on the RocksDB database developed by Facebook. This engine optimized low storage consumption and was built for high-end hardware (such as arrays of fast SSDs and many CPU cores) which is obviously important to social network and similar applications.

Here we will concentrate on InnoDB as it is the best option for most users. There are many reasons the industry prefers InnoDB as the engine for MySQL. Some of the reasons are:

- InnoDB supports row-level locking, which is vital to performance. MyISAM supports only table-level locking, thus creating a bottleneck when updating the table frequently.
- InnoDB implements transactions, which are vital to mission-critical database applications that involve banking and e-commerce.
- InnoDB supports relationship constraints, such as foreign keys, which makes it more relational database friendly over MyISAM.
- InnoDB supports ACID (Atomicity, Consistency, Isolation, Durability) transactions, making it fully compliant to RDBMS rules. MyISAM is not ACID compliant.
- InnoDB manages indexes and base tables with the help of a storage manager internally with a memory buffer pool. This increases performance efficiency. In contrast, MyISAM uses disk files primarily for the base table.

MySQL has been focusing on InnoDB for many years and constantly pushing improvements in the engine. MyISAM, on the other hand, has had little development in the last several years.

In summary, when deciding which engine to use, InnoDB seems to be the best option if you want the latest features of RDBMS.

MYSQL CONFIGURATIONS

MySQL is an RDBMS that can be installed on multiple platforms, and each platform has its own advantages and limitations. However, MySQL as an RDBMS has some specific configurations which are always true irrespective of platform and version – let us discuss a few of these configurations.

The first step in MySQL configuration is to locate the file where MySQL configuration is stored.

If you are using Linux, the location of the configuration file is

```
'/etc/mysql/my.cnf'.
```

If you are using Windows, the location of the configuration file is

```
'C:\ProgramData\MySQL\MySQL Server x.x\my.ini'
```

(you need to replace x.x with your MySQL version).

If you cannot find where your MySQL configuration file my.ini is located, you can also refer to official documentation for guidance: https://dev.mysql.com/doc/refman/8.0/en/option-files.html

SLOW QUERY LOG

The very first thing to do is to configure Slow Query Log variables in the config file. As with any RDBMS, you should have insight inside the system by logging any slow-running queries.

To configure this setting, you need to define a threshold (in seconds). Once a query crosses the threshold, MySQL engine will record the details in the Slow Query Log.

```
Syntax:     slow_query_log = 1
long_query_time = 1
log-slow-queries=/var/log/mysql/slow-query.log
```

In the syntax above slow query logging is enabled, and the threshold of the slow query is set to 1 second. The default value for the long query time is 10 seconds, which is quite long for most applications.

By default, the Slow Query Log is not enabled in MySQL. Enabling it may have a negligible impact on the overall performance for an extremely busy system. You can offset this by tuning one of the slow queries in the log.

By default, MySQL Slow Query Log does not include the administrative statements. To include them we can additionally enable log_slow_admin_statements in the configuration file. In general, logging of administrative statements should be unnecessary, unless you are squeezing the last bit of efficiency out of your RDBMS.

Before we finish discussing the Slow Query Log, it is important to discuss one additional setting: log_queries_not_using_indexes. If there are any queries that are not using indexes, they will be logged by enabling this setting in the configuration file.

```
Syntax:     log_slow_admin_statements = 1
log_queries_not_using_indexes = 1
```

VARIABLE MAX_CONNECTION

This variable max_connection defines the maximum number of connections allowed to MySQL. The default value for this variable is 151. However, on very busy systems this may not be enough. If your system is very busy and users are complaining that they have to wait a long time or they are receiving 'MySQL Error, Too many connections...', this setting might be responsible for it.

```
Syntax: max_connection = 300
```

So, is the best possible value for your variable?' If you have enough CPU memory and disk speed, you can easily increase this variable to a higher value. You should start increasing the number by 100 at a time and see if it creates an issue with overall steadiness of the application. If you ever have to set this number higher than 1,000, look into your Linux system and set up the open_files_ limit to a higher number to avoid any issues with the OS.

VARIABLE INNODB_IO_CAPACITY

This variable represents the upper limit of the number of I/O operations performed per second by InnoDB engine. This is one of the most debatable variables in the industry. It can be a struggle to optimize this variable as well.

The default value of this variable is 200, and if you have a very moderate workload or lower-end SSD this number is usually good enough. This number should never be more than the amount of the I/O operations your system and storage can handle. Many people confuse this number with the RPM of the drives. Here is some guidance to optimize my client's system regarding this variable.

Every storage system manufacturer provides the input/output operations per second (IOPS) throughput for their system in ideal conditions. In reality, the IOPS throughput for the storage system can be highly variable, depending on system load.. Here is the Wikipedia page regarding the IOPS that may help: https://en.wikipedia.org/wiki/IOPS.

If you have ultra-fast SSDs and set this value very low, you may negatively impact the performance of your MySQL. Additionally, setting a very high value of this variable will unnecessarily flood your storage system and results in negative performance. Ideally, the MySQL engine should be using the IO system to its full capacity, and not faster or slower than the IO system can handle requests.

```
Syntax: innodb_io_capacity = 1000
```

It is a good idea to set the variables between the range of 200 and the maximum IOPS suggested by the vendor. If you are not sure what value works best for you keep increasing the value of this variable by 100 every week and measure the overall performance until you find the right value for your system.

VARIABLE INNODB_BUFFER_POOL_SIZE

This variable represents the memory buffer pool size. MySQL engine stores various data in this buffer, which helps performance during subsequent requests. Requests for data can be cached in

memory, avoiding disk I/O operations for subsequent references. This is one of the most critical settings often ignored by DBAs. It is recommended that MySQL be installed as a standalone application on your server and provide maximum possible available memory to the engine.

If you have more 4GB memory on your OS, it is recommended that you save some available memory for your OS (around 2GB) and assign the rest of the memory to the MySQL. Below are some recommended metrics:

Available Memory (GB)	Memory for OS (GB)	Memory for MySQL (GB)
4GB	2 GB	2 GB
8 GB	2 GB	6 GB
12 GB	2 GB	10 GB
16 GB	4 GB	12 GB
32 GB	4 GB	28 GB
64 GB	4 GB	60 GB

```
Syntax: innodb_buffer_pool_size = 12GB
```

Keep in mind that your buffer pool size does not have to be bigger than the total size of all your databases.

VARIABLE INNODB_LOG_FILE_SIZE

This variable represents the size of the committed log files in MySQL. The default value of this file is 48MB is small considering the size of most databases. It is critical to set this file size right after the MySQL installation, as it has a huge impact on how MySQL performs.

MySQL can use multiple files to write committed transaction data from the log buffer area (innodb_log_files_in_group default is 2). This variable represents the size of each of the log files. The recommendations are to have the combined size of the log files large enough so MySQL can handle the necessary workload without any performance hindrance.

The smaller the value of the log file, the more checkpoint flush activities are required in the buffer pool area where the data has to be written from the buffer pool to disk. If you have a larger log file, the frequency of the flush activities will be greatly reduced, boosting the overall performance.

A best practice is to set this variable so that in an hour there is no more than one checkpoint flush event. You can check the log flush event in MySQL by running this command: SHOW ENGINE INNODB STATUS.

If you are not sure what value you should set for this variable, you can set the value to 1GB as a starting point:

```
Syntax: innodb_log_file_size = 1GB
```

MySQL official guidance suggests setting the total size of the log files as large as the buffer pool or even larger. The biggest fear many DBAs have is that very large log files may make crash recovery longer. However, MySQL 5.5 and later versions of MySQL have made it possible to use large log files with fast start-up after crash recovery.

VARIABLE INNODB_FLUSH_METHOD

The variable innodb_flush_method represents the method used to flush data to InnoDB data files and log files, which can have a very significant performance effect on I/O throughput. The default value for this variable changes based on the operating system of the MySQL. For Windows as a base OS, the default value is unbuffered and for Linux based OS the default value is fsync.

Here is a brief table that explains this variable and its possible values:

OS	Value	Data Flush Method
Windows	fsync	asynchronous I/O and non-buffered I/O
Windows	normal	asynchronous I/O and buffered I/O
Unix	fsync	Fsync for data and log
Unix	O_DSYNC	O_DSYNC for log and fsync for data
Unix	littlesync	unsupported / internal use
Unix	nosync	unsupported / internal use
Unix	O_DIRECT	O_Direct to open data files and fsync to flush data and log files
Unix	O_DIRECT_NO_FSYNC	Uses O_Direct for I/O flush and skips fsync to flush files

It is almost impossible to decide which options are best for you without benchmarking and proper testing of performance for your system:

```
Syntax: innodb_flush_method = O_DIRECT
```

If you are using Windows, it is suggested that you use fsync, which is the default option. On the system with hardware RAID controller, O_DIRECT method is preferred to avoid double buffering between the InnoDB buffer pool and OS Cache. When implemented, you should see significant improvements in performance with O_DIRECT. The O_DSYNC method can be helpful if your system is extremely read-heavy.

VARIABLE INNODB_DEDICATED_ SERVER (MYSQL 8.0.3 ONWARDS)

MySQL version 8.0.3 has introduced a new configuration settings variable, innodb_dedicated_ server. When this variable is set to true, InnoDB automatically configures three of the last discussed important variables according to the amount of memory detected on the server:

- innodb_buffer_pool_size
- innodb_log_file_size
- innodb_flush_method

```
Syntax: innodb_dedicated_server = 1
```

This feature is very helpful if you are using a dedicated server where your MySQL server has access to all of the available system resources. By enabling one variable, MySQL will take care of the rest. However, if you are using a shared system resource, you should avoid this feature.

VARIABLE INNODB_FLUSH_LOG_AT_TRX_COMMIT

This variable represents MySQL's behavior with ACID compliance and its impact on performance. ACID is a set of database transaction properties intended to guarantee validity of data and the system even in the event of a disaster. The default value for this variable is 1 which stands for full ACID compliance. However, by changing to a different value you can gain increased performance by relaxing the ACID compliance. Here are the different options available for this variable.

Value 1 (default value)

This value represents Full ACID compliance. Logs are written and flushed to the disk after every single transaction commit, which guarantees the availability of the data in case of disaster.

Value 0

The logs are written and flushed to the disk after every second (not guaranteed). This means that if there are any disasters, the system may lose any data which is not yet committed to the disk.

Value 2

The logs are written after every transaction, but they are flushed to the disk after every second (not guaranteed). This option is right in the middle of Value 1 and Value 2. This means that if there are any disasters, the system may lose any data which is not yet been flushed to the disk.

```
Syntax: innodb_flush_log_at_trx_commit = 0
```

Be cautious about changing this variable. If you are comfortable losing the data up to 1 second and if they are not doing any financial transactions, it may makes sense to change this setting to any other option and gain additional performance benefits.

CONCLUSION

MySQL is a very well-developed RDBMS, and there are several variables, options, and configurations one can change to get improved performance. In this whitepaper we have discussed seven of the most important configuration checks for DBAs. They are great starting points for any MySQL deployment. It is very common to get a performance boost from MySQL deployment after making these important changes.

Once the system has stabilized after making all the initial changes, it is a good idea to focus on additional configuration changes as needed by your application. Once the MySQL system starts giving robust performance improvements from the configuration changes, DBAs can start focusing on the next level of improvement by doing index maintenance and monitoring the system health via performance schema.

Here is a quick syntax summary of all the changes that we have discussed in this whitepaper for reference.

```
Syntax:
slow_query_log = 1
long_query_time = 1
log-slow-queries=/var/log/mysql/slow-query.log
log_slow_admin_statements = 1
log_queries_not_using_indexes = 1
max_connection = 300
innodb_io_capacity = 1000
innodb_buffer_pool_size = 12GB
innodb_log_file_size = 1GB
innodb_flush_method = O_DIRECT
innodb_dedicated_server = 1
innodb_flush_log_at_trx_commit = 0
```

ABOUT THE AUTHOR

Pinal Dave is a SQL Server Performance Tuning Expert (https://blog.sqlauthority.com) and an independent consultant. He has authored twelve SQL Server database books, twenty-five Pluralsight courses and has written over 5,000 articles on the database technology on his blog at https://blog.sqlauthority.com. Along with 17+ years of hands-on experience, he has a Master of Science degree and a number of database certifications.