Reading: Monitoring and Optimizing Your Databases in MySQL

Estimated time needed: 25 minutes

In this reading, you'll learn how you can monitor and optimize your queries with the help of the tool, phpMyAdmin.

Objectives

After completing this reading, you will be able to:

1. Identify the purpose of monitoring and optimizing your databases
2. Describe the steps to monitoring the health of your database with phpMyAdmin
3. Apply the best practices in optimizing your database

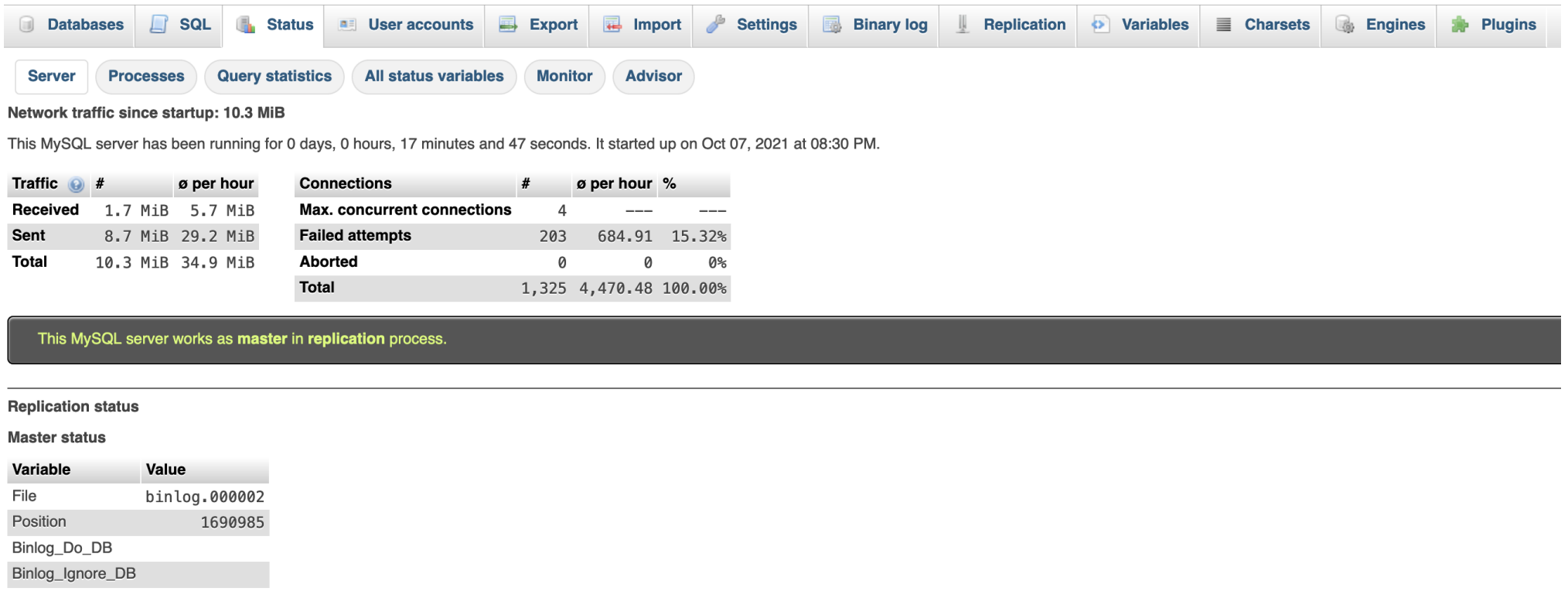
Software Used

In this reading, you will see usage of MySQL. MySQL is a Relational Database Management System (RDBMS) designed to efficiently store, manipulate, and retrieve data.

Monitor Your Database

Database monitoring refers to the tasks related to reviewing the operational status of your database. Monitoring is critical in maintaining the health and performance of your database and with proper monitoring, any problems that appear can be identified and resolved in a timely manner. This will keep your database running and avoid any outages that may affect yourself or your end users.

Tools such as phpMyAdmin, an open-source administration tool for MySQL and MariaDB, have built-in features that help with monitoring your database. Particularly in phpMyAdmin, you can gain a better understanding of how your server and its databases are functioning through the **Status** page.



Server Status

On the Server page, you're provided with more information about the server status, including how long you've been running the server for.

1. The **Traffic** table shows the incoming and outgoing traffic, where **Received** is the number of bytes received by the server (incoming), and **Sent** is then number of bytes sent out by the server (outgoing). These numbers can tell you how much traffic you're receiving and based on performance, whether you're receiving unusually high traffic.

2. The **Connections** table gives you information about the maximum number of connections you can have at the same time, the number of failed connection attempts, the number of aborted connection attempts, and the total number of connections (successful or not).

Network traffic since startup: 10.3 MiB

This MySQL server has been running for 0 days, 0 hours, 17 minutes and 47 seconds. It started up on Oct 07, 2021 at 08:30 PM.

1

Traffic	#	ø per hour
Received	1.7 MiB	5.7 MiB
Sent	8.7 MiB	29.2 MiB
Total	10.3 MiB	34.9 MiB

2

Connections	#	ø per hour	%
Max. concurrent connections	4	---	---
Failed attempts	203	684.91	15.32%
Aborted	0	0	0%
Total	1,325	4,470.48	100.00%

Processes

On the Processes page, you can see the operations currently being performed within the server.

There will always be at least two users: **event_scheduler** and **root**.

Server | Processes | Query statistics | All status variables | Monitor | Advisor

Filters

Show only active | Refresh

Processes	ID	User	Host	Database	Command	Time	Status	Progress	SQL query
Kill	5	event_scheduler	localhost	*None*	Daemon	2982	Waiting on empty queue	---	---
Kill	1975	root	172.25.0.3:40202	mysql	Query	0	init	---	SHOW PROCESSLIST
Kill	1976	root	172.25.0.3:40204	*None*	Sleep	0	---	---	---

Note: Enabling the auto refresh here might cause heavy traffic between the web server and the MySQL server.

Refresh rate: 5 seconds | Start auto refresh

This table shows the **ID** of the process; the **User** associated with it, the **Host** to which it's connected to; the default **Database** or None; the **Command** type, if it's being executed or if the session is idle (Sleep); the **Time** it has been in its **Status** for; and the **Status** which gives a description of what it's doing right now.

As can be seen in the previous table, you can replicate the displayed process list in the command-line interface with the following command:

```
SHOW PROCESSLIST
```

```
mysql> SHOW PROCESSLIST;
+------+-----------------+------------------+-------+---------+------+-------------------------+------------------+
| Id   | User            | Host             | db    | Command | Time | State                   | Info             |
+------+-----------------+------------------+-------+---------+------+-------------------------+------------------+
|    5 | event_scheduler | localhost        | NULL  | Daemon  | 5024 | Waiting on empty queue | NULL             |
| 2548 | root            | 172.25.0.1:48704 | world | Query   |    0 | init                    | SHOW PROCESSLIST |
+------+-----------------+------------------+-------+---------+------+-------------------------+------------------+
2 rows in set (0.00 sec)
```

Query Statistics

On the Query Statistic page, you'll see the types of queries run and how often they've been run. These include queries that you have run, in addition to queries run by phpMyAdmin in the background.

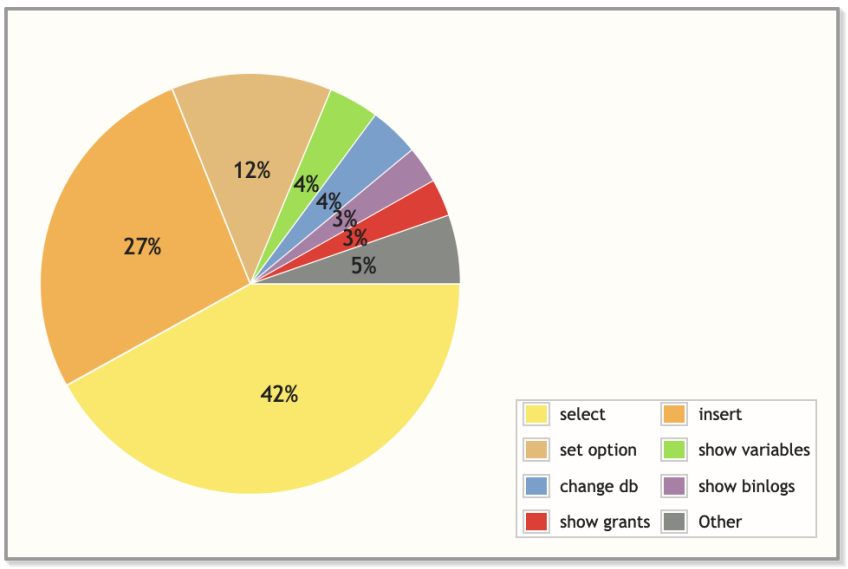
The page also offers a visualization of the breakdown of usage.

Server | Processes | Query statistics | All status variables | Monitor | Advisor

Questions since startup: 19,323

- ø per hour: 16,500
- ø per minute: 275
- ø per second: 5

Statements	#	ø per hour	%
select	8,110	6,925	41.97
insert	5,208	4,447.1	26.95
set option	2,388	2,039.1	12.36
show variables	748	638.7	3.87
change db	741	632.7	3.83
show binlogs	551	470.5	2.85
show grants	551	470.5	2.85
show status	290	247.6	1.50
show master status	170	145.2	0.88
show slave status	170	145.2	0.88
show replica status	170	145.2	0.88
show processlist	137	117	0.71
create table	37	31.6	0.19
show tables	25	21.3	0.13
show fields	9	7.7	0.05
show warnings	5	4.3	0.03
show keys	4	3.4	0.02
drop table	3	2.6	0.02
create db	2	1.7	0.01
show databases	2	1.7	0.01
show create table	1	0.9	0.01
flush	1	0.9	0.01



As we can expect, `SELECT` is the command that has been used the most in the previous above. This result makes sense because we are often selecting specific columns and rows from our database to view.

All Status Variables

On the All Status Variables page, you'll see the variables listed, their value and their description. If the value of your variable is red, then that means it may need to be investigated and adjusted, if necessary.

For example, in this case **Aborted connects**, which is the number of failed attempts to connect to the server, is red. The number is rather high, meaning it should be looked into. A high number of aborted connects could mean that the program was not closed properly, the session timed out, or an error occurred in the middle of a data transfer.

Server | Processes | Query statistics | All status variables | Monitor | Advisor

Filters

Containing the word: | Show only alert values | Filter by category… | Show unformatted values | Refresh

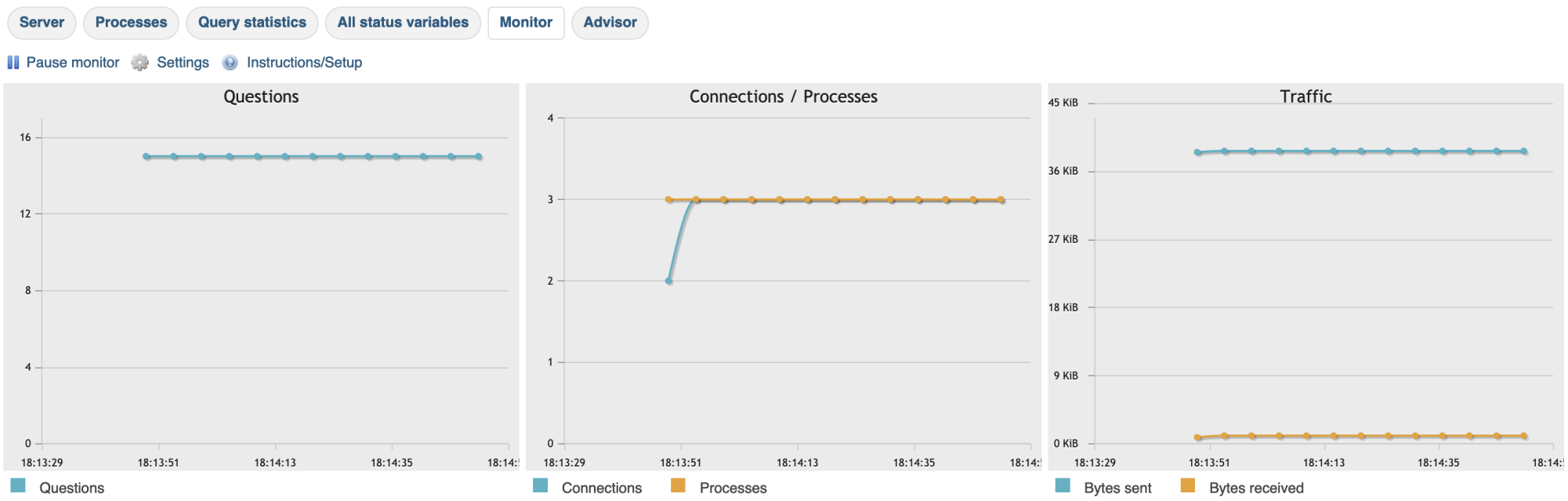
Variable	Value	Description
Aborted clients	0	The number of connections that were aborted because the client died without closing the connection properly.
Aborted connects	836	The number of failed attempts to connect to the MySQL server.
Acl cache items count	0	
Binlog cache disk use	0	The number of transactions that used the temporary binary log cache but that exceeded the value of binlog_cache_size and used a temporary file to store statements from the transaction.
Binlog cache use	5.2 k	The number of transactions that used the temporary binary log cache.
Binlog stmt cache disk use	0	
Binlog stmt cache use	3	
Bytes received	2.1 M	
Bytes sent	16.7 M	
Caching sha2 password rsa public key	-----BEGIN PUBLIC KEY----- MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAvLtO80HCKyKMgbiyK+xY fExTd0LxpwUMMJsFClbIR1o6kTu7rd1ZZ0g14JsO2wuDOg6/KBZrsRqP9UU5r9ix 6mIePlSkGh7k3oOFXV7c1Y54WVdZ1OuqfOnMol+S/ht50VgZk9g6lt7I/glQsHbm Se32B6j1q25k/6XUSYTPGX1kJtXAwSuHU3BU/Oz4mRCa5aPMRFclWwnSejqJ4/8+ AAB09KxS7GD3vRltQWdMv/538jdRSSGfgVNibt1s5QA1pw9ehfNAG3BQKckB+edY sufiYgde4t3kdJksQrzUmK7B7hlNlTN5oMOLn8nlz5Hc3TW/5VR2Ep06tnoQ+u2b DQIDAQAB -----END PUBLIC KEY-----	
Com admin commands	0	
Com assign to keycache	0	
Com alter db	0	
Com alter event	0	

For more details about each variable and its meaning, you can read the Server Status Variables Documentation by MySQL.

Monitor

With the Monitor page, you can optimize your server configuration and identify issues such as intensive queries based on the provided real-time graphs.

By default, you may only see three charts: **Questions**, **Connections / Processes** and **Traffic**.

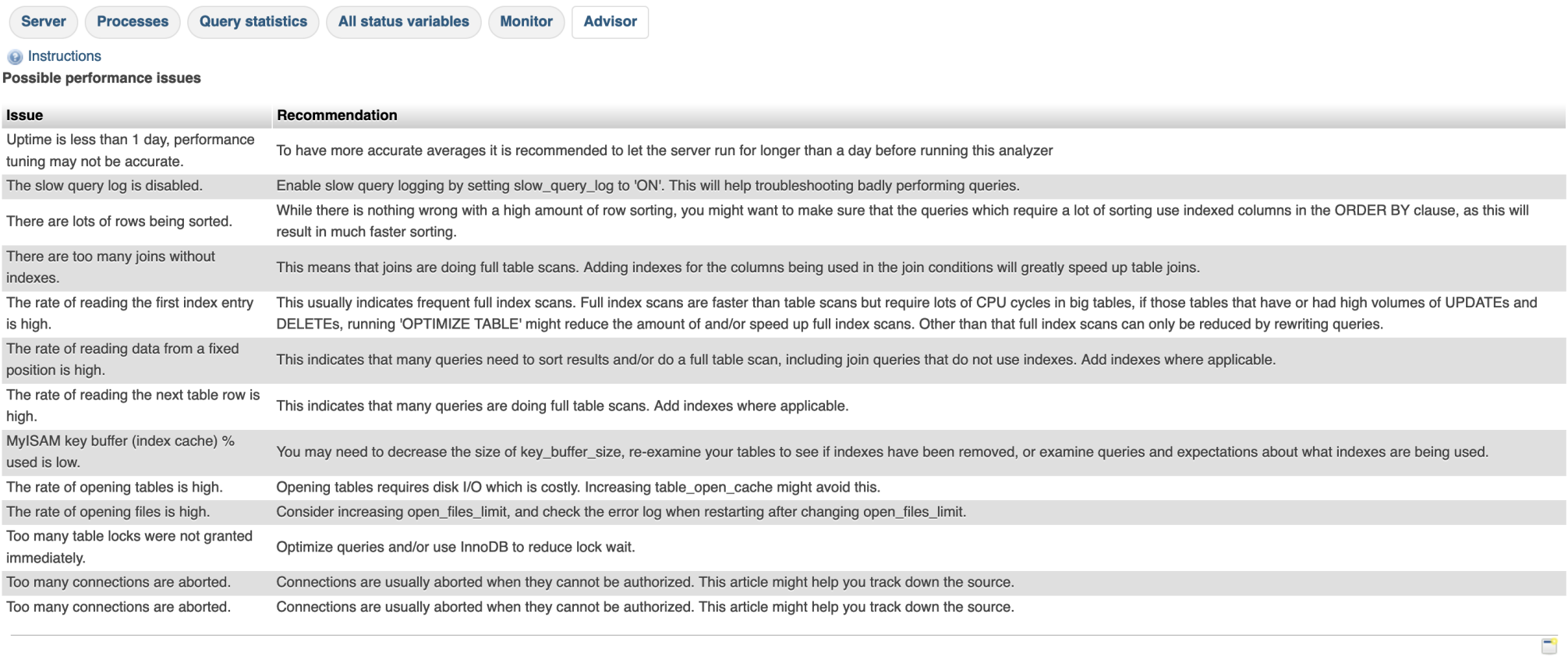


Questions is the number of queries that are being run at a given time, including any background queries by phpMyAdmin. Similar to what is displayed on the **Server** page, the **Connections / Processes** and **Traffic** provide a visual of the current connections, processes and incoming and outgoing bytes received and sent.

These real-time charts can help you spot any irregularities in your databases, such as a sudden spike in querying, connections or traffic. Additional charts can also be added, such as a chart that displays the system's CPU usage or system's memory. These will be further explored in the lab.

Advisor

phpMyAdmin also offers an Advisor system that provides recommendations based on analysis of server variables to help monitor and optimize your database.



Issue	Recommendation
Uptime is less than 1 day, performance tuning may not be accurate.	To have more accurate averages it is recommended to let the server run for longer than a day before running this analyzer
The slow query log is disabled.	Enable slow query logging by setting slow_query_log to 'ON'. This will help troubleshooting badly performing queries.
There are lots of rows being sorted.	While there is nothing wrong with a high amount of row sorting, you might want to make sure that the queries which require a lot of sorting use indexed columns in the ORDER BY clause, as this will result in much faster sorting.
There are too many joins without indexes.	This means that joins are doing full table scans. Adding indexes for the columns being used in the join conditions will greatly speed up table joins.
The rate of reading the first index entry is high.	This usually indicates frequent full index scans. Full index scans are faster than table scans but require lots of CPU cycles in big tables, if those tables that have or had high volumes of UPDATEs and DELETEs, running 'OPTIMIZE TABLE' might reduce the amount of and/or speed up full index scans. Other than that full index scans can only be reduced by rewriting queries.
The rate of reading data from a fixed position is high.	This indicates that many queries need to sort results and/or do a full table scan, including join queries that do not use indexes. Add indexes where applicable.
The rate of reading the next table row is high.	This indicates that many queries are doing full table scans. Add indexes where applicable.
MyISAM key buffer (index cache) % used is low.	You may need to decrease the size of key_buffer_size, re-examine your tables to see if indexes have been removed, or examine queries and expectations about what indexes are being used.
The rate of opening tables is high.	Opening tables requires disk I/O which is costly. Increasing table_open_cache might avoid this.
The rate of opening files is high.	Consider increasing open_files_limit, and check the error log when restarting after changing open_files_limit.
Too many table locks were not granted immediately.	Optimize queries and/or use InnoDB to reduce lock wait.
Too many connections are aborted.	Connections are usually aborted when they cannot be authorized. This article might help you track down the source.
Too many connections are aborted.	Connections are usually aborted when they cannot be authorized. This article might help you track down the source.

Please note, the system's recommendations are based on general rules and simple calculations that may not apply to your database, so it's important to understand what you are changing with the MySQL documentation and how you can undo that change if there was no improvement from it. It's generally best practice to tune your system by changing one setting at a time.

Now that you know how to monitor the health of your database, let's take a look at how we can optimize it!

Optimize Your Database

Database optimization refers to maximizing the speed and efficiency of retrieving data from your database, improving the experience for both yourself and your learners. Database optimization can also help with improving the performance of slow queries.

It's best to keep your data types simple, using the smallest or a specialized data type when possible. Smaller data types tend to be faster because they require less space, with specialized data types such as date and time (**DATETIME**) optimized to save disk space and memory. To understand the amount of storage a data type can take, you can read the MySQL Data Types Documentation.

In addition, it's helpful to avoid **NULL** values in columns when possible. Non-null values make operations faster as they can use indexes better and eliminate the need to check if each value is **NULL**.

Let's take a look at the various data types and how we can best use them.

Numeric Data Types

In MySQL, two of the most commonly used data types are: integer types and fixed-point types.

Integer Types

With Integer Types, we see that MySQL supports **TINYINT**, **SMALLINT**, **MEDIUMINT**, **INT** and **BIGINT**. Depending on your use, you should choose the smallest possible data type.

From the Integer Types Documentation page by MySQL, we notice a table similar to the following:

Type	Storage (Bytes)	Minimum Value Signed	Maximum Value Signed	Minimum Value Unsigned	Maximum Value Unsigned
TINYINT	1	-128	127	0	255
SMALLINT	2	-32,768	32,767	0	65,535
MEDIUMINT	3	-8,388,608	8,388,607	0	16,777,215
INT	4	-2,147,483,648	2,147,483,647	0	4,294,967,295
BIGINT	8	-2^{63}	$2^{63}-1$	0	$2^{64}-1$

Based on the table presented in the MySQL documentation, we can see that there are six columns **Type**, **Storage (Bytes)**, **Minimum and Maximum Value Signed** and **Minimum and Maximum Value Unsigned**. Unsigned values mean that the values are always non-negative values whereas signed values can be negative, as can be seen in the maximum and minimum values of each type.

This table is telling us how much storage each type uses. For example, **TINYINT** uses the least amount of storage and **BIGINT** uses the most. When looking at their maximum and minimum values, this makes sense because the range of unsigned values for **TINYINT** is 0 to 255, whereas for **BIGINT** it is 0 to a very large number. As such, it'll take up more space.

If you know that your values will only go up to 255, such as if you're logging information of sports teams and there's only 200 in the league, then it would be better to use **TINYINT** over **BIGINT** or **INT**.

Fixed-Point Types

Fixed-point types represent non-integer numbers. In MySQL, the two data types are **DECIMAL** and **NUMERIC**, both of which preserve the precision of a number. For example, if you wanted to store monetary data, this data type would come in handy.

String Data Types

The two most popular types of strings are **CHAR** and **VARCHAR**, and you can optimize these by limiting the number of characters in a column.

VARCHAR columns have variable-length strings, meaning in terms of storage, it holds the characters you assign it to. It will also take up an additional two bytes for storage purposes. **CHAR** columns had fixed length, meaning regardless of the actual characters in the column, it will always use the same amount of space.

For example, **CHAR(3)** will always use 3 bytes, even if those bytes are whitespace. The amount of storage **VARCHAR(3)** takes, however, will depend on the contents of that column. Therefore, **CHAR** is a good option if you know that every entry in the column will remain fixed, otherwise, **VARCHAR** may be the better option.

Generally, if you have a value that can be represented by either a string or a number, you should choose the numeric data type. Numeric values take less storage than strings, therefore optimizing your database.

Date and Time Data Types

MySQL also has special data types for date and time. These types include **DATE**, **TIME**, **DATETIME**, **TIMESTAMP** and **YEAR**.

DATE values are displayed as `YYYY-MM-DD`, **TIME** values are displayed as `hh:mm:ss`, **DATETIME** and **TIMESTAMP** values are displayed as a combination of both as `YYYY-MM-DD hh:mm:ss`, and **YEAR** values are displayed as `YYYY`.

It's best practice to use the appropriate date or time data types rather than, for example, strings since these data types are optimized to hold those values, therefore saving storage and performing better.

A key takeaway when it comes to optimizing data types is understanding your data. Not only by column name, but the values that are included in the columns. By understanding them, you'll be able to choose the best data type for that column.

Optimize Your Queries

In addition to reviewing your database and data types within it, optimizing your queries will also be helpful in optimizing the overall performance of your database. Part of optimizing your queries entails using the `EXPLAIN` statement to see your query's execution plan and reviewing your tables' indexes to ensure that they are used appropriately.

For a refresher on how you can optimize your queries, take a look at the Improving Performance of Slow Queries in MySQL Reading!

Next Steps

Now that you're more familiar with how you can monitor and optimize your query, let's take a look at how we can do this in practice with phpMyAdmin in the Skills Network Labs environment!

Author(s)

Kathy An

Other Contributor(s)

Changelog

Date	Version	Changed by	Change Description
2021-10-12	1.0	Kathy An	Created initial version
2021-10-13	1.1	Steve Hord	Copy edits