ITPro Today

TOP TIPS

Ten Tricks for Optimizing MySQL

MySQL continues to gain popularity as the data platform for many critical business applications. As the most widely used open-source database, MySQL provides database services for many on-premises and cloud-based applications. MySQL performance is critical. Performance tuning MySQL depends on a number of factors, and businesses are always looking for ways to tweak the database to gain added performance. Some of the tips here are basic and some are more technical, but all need to be considered and implemented for optimal MySQL database performance.

1. Ensure Adequate Server Resources

Database servers like MySQL need four fundamental server resources: CPU, memory, storage, and network bandwidth. Of these, memory may be the most important factor for good database performance. MySQL will use memory to cache query results, reducing the need to use slower storage I/O. Use the Linux top command to get a quick check of your server's CPU and memory utilization.

2. Use SSD Storage

Taking advantage of the fastest storage systems possible is a quick way to increase your performance levels. If you are currently using HDDs for MySQL storage, moving to SSDs can significantly improve your database performance. SSDs are much faster than HDDs. They are also more reliable and use less power. A 7,200 RPM HDD can deliver about 150MB/s transfer throughput, while a SATA III SSD connection can provide about around 550MB/s read and 520MB/s write speeds.

3. Tune the Linux Swappiness Setting

If you're running MySQL on Linux, you can boost performance by optimizing Linux's Swappiness setting. The Swappiness value controls the ability of the kernel to swap out memory pages. On a database server with enough RAM, this value should be as low as possible. The default value is 60. A better choice is using a value of 1 which enables the minimum amount of swapping. To set the Linux Swappiness value, open the file /etc/sysctl.conf as an admin user and add the line: vm.swappiness = 1.

4. Optimize Your Indexes

Indexing mistakes are common culprits for slow MySQL performance. Without an index, MySQL must scan the whole table to locate the required rows, which is a slow and I/O intensive operation. It's important to remember the more indexes you have, the slower your database writes will be because of the need to update the associated indexes. Using the EXPLAIN statement, the log_queries_not_using_indexes configuration setting, or a database performance monitoring tool can help you find missing indexes.

5. Properly Size the Buffer Pool

The buffer pool enables frequently used data to be accessed directly from the in-memory buffer pool cache rather than always needing to go to disk. This enables MySQL to provide far faster response times for repeatedly queried data.. As a general rule of thumb, allocate about 80% of system memory to the buffer pool. For example, 6GB for 8GB RAM, 24GB for 32GB RAM, or 100GB for 128GB RAM. You can tune MySQL's buffer pool size by editing the value innodb_buffer_pool_size in the MySQL configuration file normally found at /etc/mysql/my.cnf.

6. Optimize the Redo Log File Size

Increasing the size of the redo log file from its default value of 48MB can result in improved performance for write operations. While there are more exact ways to calculate your log file size, quick rule of thumb is to make your log files the same size as the buffer pool. You can optimize the log file size by configuring the innodb_log_file_size setting in the MySQL configuration file at /etc/mysql/my.cnf.

7. Avoid the Use of SELECT *

Using SELECT * will retrieve all columns from the target table, resulting in unnecessary I/O, memory, and network bandwidth usage. These unnecessary columns cause additional load on the database, slowing down the specific query and the entire system as a whole. Instead of using SELECT *, specify the actual columns required in the SELECT clause. This will improve MySQL query performance and overall system utilization by only retrieving the required data.

8. Avoid LIKE Expressions With Leading Wildcards

However, MySQL cannot use indexes when there is a leading wildcard in a query. For instance, the following query will cause MySQL to perform a full table scan even if you have indexed the 'last_name' field of the students table: SELECT * FROM students WHERE last_name LIKE '%son'.

9. Take Advantage of EXPLAIN

EXPLAIN is a useful tool to help you understand and optimize your MySQL queries. Take advantage of EXPLAIN simply by appending the EXPLAIN expression to the beginning of your query. EXPLAIN will read and evaluate the query, then output a row of information about each table in the query. EXPLAIN will provide information about the query, including number of rows processed, possible indexes, and the index actually chosen. Use EXPLAIN like this: EXPLAIN SELECT last_name, first_name FROM students.

10. Monitor Your Workloads

One of the best ways to improve and maintain MySQL performance is to perform regular database monitoring. Database performance monitoring can reveal the slowest and most expensive queries you might want to tune as well as CPU, memory and I/O utilization, IO hotspots, and wait statistics. You can use regular monitoring to create performance baselines, which can help you see trends and performance over time and quickly find anomalies. It can also help you find out if your database tuning or configuration changes were effective.

Visit solarwinds.com to learn about our Database Performance Monitor solution and to request a 14-day free trial.