solarwinds

BEST PRACTICES FOR EFFECTIVE MYSQL PERFORMANCE TUNING

By: Janis Griffin
Database Performance Evangelist/Senior DBA

SOLARWINDS
WHITEPAPER

BEST PRACTICES FOR EFFECTIVE MYSQL PERFORMANCE TUNING

INTRODUCTION

While many enterprises rely on MySQL because of its open source freedom, lower total cost of ownership, scalability, application development and other advantages, it has only recently been instrumented for performance tuning and optimization. When 25% of users will abandon a database-driven web application after only three seconds of delay[1], slow database performance is no longer something that can be easily ignored.

What kind of improvements can be made through tuning? In one instance, tuning a customer-facing website application for end user response time resulted in increasing product searches from 10,000 to 30,000 per day and dropping product search queries from 22 hours to 22 minutes.[2]

In this whitepaper, we discuss the challenges specific to MySQL tuning and provide practical guidance on how to achieve database-dependent application performance that can result in better enterprise productivity, agility and decision making, as well as improved user responsiveness. Drawing on our experience, we'll:

- Demonstrate how to effectively use the MySQL performance schema to pinpoint bottlenecks and get clues on the best tuning approach
- Show you techniques for quickly identifying inefficient operations through review of query execution plans
- Discuss monitoring techniques to make sure the query stays tuned

HOW DO WE KNOW WHAT TO TUNE?

The impetus for seeking better performance might come from users vexed by productivity bottlenecks or executives determined to drive efficiencies, but improvement starts with one common question: How do we know what to tune?

Database tuning is both an art and a science, a venture that requires technical expertise and business acumen. Knowing how MySQL will access data, being able to read and verify the MySQL execution plan, understanding SQL query design and knowing how to write good SQL statements is required, since modifying the query might be necessary.

It's also critical to determine how the application dovetails with business processes. Are the demands on the database made by any given user necessary? In one instance, a group within a company was unable to run the end-of-month reports they needed. Upon investigation, it was found that another group was dominating compute resources by running a series of reports that no one realized were obsolete. In this case, simply having insight into business processes solved the issue—no special technology required.

Unfortunately, tuning is almost never so simple. Prioritizing which statements to tune can be problematic and, because each statement is different, creative approaches are necessary; that's where the "art" comes into play. Throwing shiny new hardware like SSDs and memory may seem like an easy and increasingly affordable solution. However, if you don't know the root cause of the performance issue, you may be throwing money – and opportunity – away.

Those who manage a MySQL environment deal with a complex, dynamic environment, often bearing the responsibility of managing the hundreds of database instances that serve their enterprise. The continued migration of databases to the cloud potentially complicates the situation. Consequently, optimizing MySQL performance requires a proven process and tools will enable you to accurately assess baseline performance over time and accurately identify causes of slow performance.

TUNING TAKES A VILLAGE—WHO SHOULD BE INVOLVED?

Database managers and developers each bring different skills to the table, which makes a convincing argument that tuning calls for a team effort. Users need to be part of the roster as well, because they are the primary customer, and it's their experiences that usually first call attention to issues. Why are they the ones to point out slow application performance? It's because DBAs are frequently trying to "just keep the lights on" while administering a large number of database instances, and developers are keenly focused on functionality as well as meeting their deadlines. Further, DBAs probably don't know the code as well as developers, and developers might use SQL code generators to create the SQL statements, since they primarily code in PHP or Java.

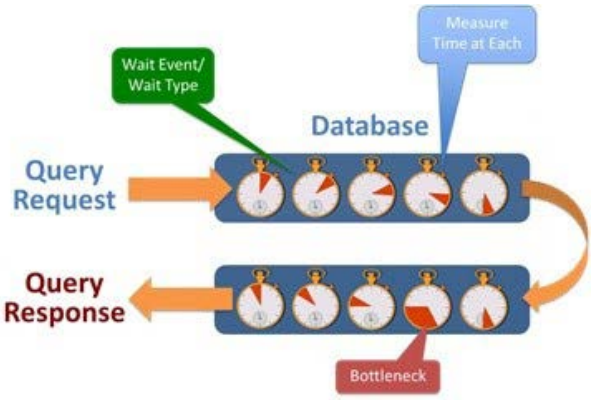
By defining tuning to be a team project, management can draw on the skills possessed by each person in the team and diffuse the finger-pointing that not having a clear picture of issues sometimes causes.

PREPARING TO TUNE—WHAT SHOULD BE TUNED?

Whether an enterprise is looking at disappointing internal- or external-facing database dependent processes, finding the root cause commonly begins with assessing counters and system health metrics. But that course of action often doesn't provide complete information, can be misleading or focuses attention on a limited time span which doesn't correlate with end user experience.

Determining which SQL statements to focus on is the first step in the tuning project. User complaints are obvious road signs that point you to the queries or batch jobs they are running. Next, looking for the queries that are consuming resources, like I/O or CPU, can narrow the search. You might also look to server metrics such as memory utilization, buffer cache or table scans for clues.

While all of these investigations are likely to yield important clues, a more effective approach is response time analysis (RTA.) RTA is a proven way to monitor, manage and tune database performance that is based on the optimal end user outcome, whether that user is an application owner, batch process or consumer. RTA does for those who manage MySQL what application performance management does for IT—identifies and measures a process end-to-end, starting with a query request and ending with a query response, including the wait time spent at each discrete step in the process.



You can apply RTA to MySQL by analyzing the performance and information schemas to reveal performance issues which traditional tools (used to assess the health of servers and databases) don't identify. The performance_schema is a real-time in-memory database providing information about users, the queries that are running and resources used. Information_schema holds the metadata about all the objects in a MySQL instance and also contains INNODB storage engine's transactional information.

Step 1: Gather Metrics

Because RTA takes into account the end user experience, you can quickly identify what they are waiting on and know where to begin gathering baseline performance data. By focusing on query response time, you can rank the hundreds or potentially thousands of steps that an SQL query works through before delivering the end result set.

In addition to the total time waited on for a given query, MySQL gives you data on Instrumented Waits and Thread States, showing the time required for each step and the resource allotment, which are great clues on where to start tuning. With these initial measurements in hand, you can now move to examining the execution plan.

MySQL 5.6+

Enhancements to performance and information features in MySQL have made it more useful when tuning.

Performance_schema

- Greatly improved to give more information on users, the queries they are running and resources they are using
- Provides current and historical events at the Statement, Stages and Waits levels
- Consumers now on by default

Information_schema

- Holds the metadata for all the objects in an MySQL instance
- Contains INNODB storage engine's transactional information

The performance drain characteristic of past versions has been significantly reduced and many more thread states and instrumented waits are captured.

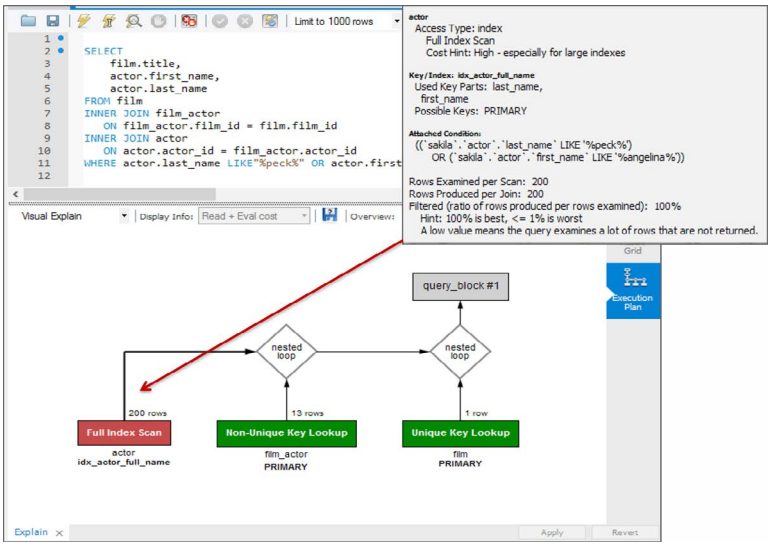
Step 2: Examine the Execution Plan

Experienced MySQL users know that MySQL will generate a tabular execution plan when you call EXPLAIN or Explain EXTENDED commands. As of 5.6.3, Optimizer Trace became available, which gives more information on how Optimizer arrived at the final plan.

MySQL Workbench gives you both a graphic representation and tabular view into the execution plan. It also displays more granular details about each step when you mouse over an icon. In the example shown here, you can see a "high cost warning" that Optimizer recognizes in this plan – one reason that the query needs tuning.

Using EXPLAIN will give a tabular view of the select type, the table names, possible keys it looks at, and the key that it finally chose. It will also show the key length, foreign key reference and the number of rows that it plans on reading. Finally, the "extra" columns will give you more information about how it's going to filter and access the data.

Reading execution plans is a critical part of tuning. For example, the possible_keys field can often help in optimizing queries, since if the column is NULL, it indicates no relevant indexes could be found.[3]



It's important to note that MySQL Optimizer can transform the SQL queries it receives. 'EXPLAIN EXTENDED' shows you how the MySQL optimizer rewrites your query when you use 'SHOW WARNINGS' after the EXPLAIN EXTENDED clause. The EXTENDED output will display additional information, including the rewritten or transformed query. This can be very useful if you have data type mismatches (i.e. implicit conversions) or have 'SELECT *' in the query, as you can quickly see what transformations need to take place.

Optimizer Tracing returns all the access paths and statistics the Optimizer considered before arriving at the final execution plan, and is especially revealing when you are trying to find out why MySQL Optimizer is "stuck" on a certain plan. The trace is stored in the INFORMATION_SCHEMA. OPTIMIZER_TRACE table (and the default size is 16K, which is often too small to see a large query).

To turn on optimizer tracing, you 'SET OPTIMIZER_TRACE="enabled=on"' and then run your query. To view the trace, simply select it out of the OPTIMIZER_TRACE table in the INFORMATION_SCHEMA. Keep in mind that Optimizer Tracing is only a snapshot in time and doesn't show historical trends, which can mislead you to assume that you have pinpointed an issue which is not truly the root cause.

Step 3: Review Table and Index Information

After gathering baseline metrics and reviewing the execution plan, the next step is to review the table and index information. We suggest breaking this into three steps.

1. Confirm whether you are looking at a table or a view. Determine where the table resides, review the keys and constraints to see how the tables are related. Use 'MYSQLSHOW --STATUS {database name}' to obtain table sizes.

2. Examine columns in the WHERE clause – Use 'SHOW INDEXES FROM {TABLE NAME}' to check out the indexes and their cardinality and length.

3. Know your existing indexes – Establish the number of columns the indexes contain; MySQL can only use the left-leading column in a multi-column or composite index.

Step 4: Measure the Results, Repeat

Once you've successfully tuned the query, you need to monitor the improvement to prove that tuning made a difference. Take new baseline measurements and compare them to the initial reading. Tuning can be an iterative process as there is always room for improvement as applications change, data grows and workloads change.

CONTINUOUS MONITORING: THE NEW NORMAL

Once you've successfully resolved a critical incident or singular performance issue, the final important step to take is to adopt a continuous monitoring approach. Continuous performance monitoring tools mark the next evolution of tools available for DBAs implementing RTA. These tools work in collaboration with MySQL to simplify performance management by consolidating performance information in one place, displayed on a dashboard that shows where to fix issues and where to proactively tune.

It isn't humanly possible to monitor the hundreds if not thousands of databases a team of DBAs manage, and so an obvious benefit of a continuous performance management tool is that it can cope with the volume. Other advantages emerge as well. With insight into normal performance and trends over time, capacity planning and predictive analytics become more accurate. One can also assess the impact of transitioning database services to the cloud and track application performance in a shared environment where one database competes for resources with another, and then validate whether that move to a hosted storage and computing platform returns the positive ROI.

Thread State Example

The "sending data" state is MySQL performing large amounts of disk access (reads). Here's a tip that can help when tuning MySQL queries:

Compare **Rows Examined** to **Rows Affected or Sent** statistics. If **Rows Examined** is more than 10 times larger than **Rows Affected or Sent**, consider the following:

- Use summary tables where possible to limit the number of rows processed.
- Rewrite complicated queries to assist in processing fewer rows.
- Evaluate WHERE clauses to ensure you process only rows that are required.

If the number of rows retrieved hasn't changed but the time spent in Sending Data has increased, this could indicate slow I/O performance.

This also can indicate slow network speeds between the MySQL server and the application.

CONCLUSION

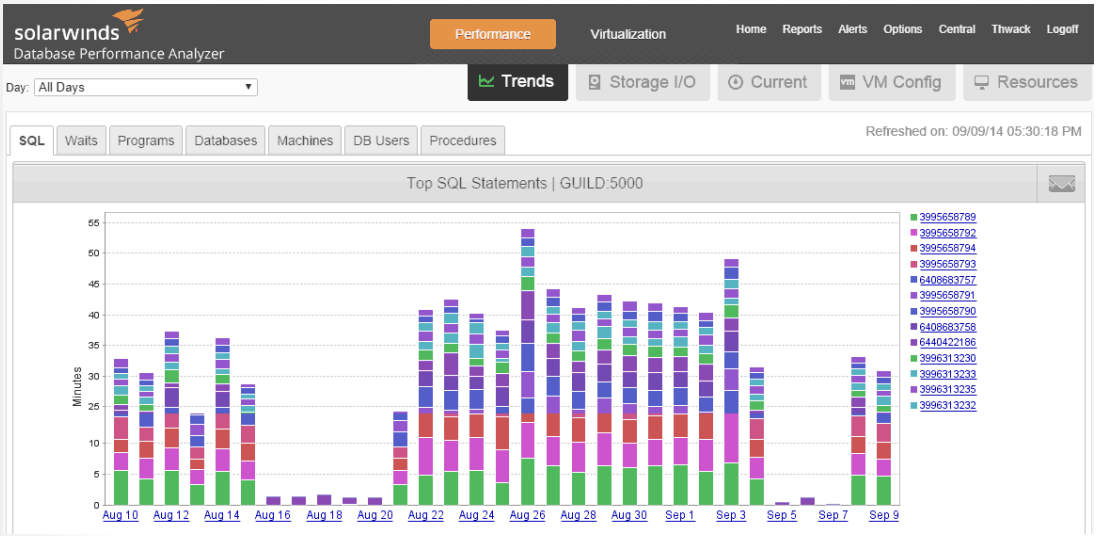
A growing number of IT teams are moving from established performance monitoring practices to RTA, recognizing that taking an end user perspective, where the time elapsed using an application, is the true measure of productivity and success of the database service. MySQL has evolved to support that paradigm, and along with third-party tools gives DBAs further information to flesh out the story partially told by database health metrics and network performance measurements.

Using RTA, you can identify bottlenecks, pinpoint their root causes and prioritize your actions based on the impact to end users. RTA enables IT management to direct issues to the appropriate resources, whether that's the DBA, a developer or network administrator. In short, RTA helps you provide better service, resolve trouble tickets faster by knowing precisely where to focus MySQL tuning efforts and free up time to focus your efforts where they have the most impact.

Computing power has soared. Data volumes have grown astronomically, and enterprise dependence on application performance has become truly business critical. With so much at stake and so many responsibilities, DBAs need to find and fix problems the first time, every time.

HOW CAN DATABASE PERFORMANCE ANALYZER HELP?

Database Performance Analyzer (DPA) from SolarWinds provides the fastest way to identify and resolve database performance issues. DPA is part of the SolarWinds family of powerful and affordable IT solutions that eliminate the complexity in IT management software. DPA's unique Multi-dimensional Database Performance Analysis enables you to quickly get to the root of database problems that impact application performance with continuous monitoring of MySQL, SQL Server, Oracle, SAP ASE and DB2 databases on physical, Cloud-based and VMware servers.



For additional information, please contact SolarWinds at 866.530.8100 or e-mail sales@solarwinds.com.

LEARN MORE DOWNLOAD FREE TRIAL

References

1. Aberdeen Group, "Reaching the Top of the Web Performance Mountain" (May 2013)

2. The Total Impact of SolarWinds Database Performance Analyzer, Forrester, Michelle Bishop, Principal Consultant, 2013

3. www.sitepoint.com/using-explain-to-write-better-mysql-queries/