

Identifying Slow Queries

Slow queries can degrade the performance of your MySQL-based application, leading to lag, timeouts, or higher resource consumption. On Elestio, whether you're accessing MySQL via terminal, inside a Docker Compose container, or using MySQL CLI tools, there are structured ways to inspect and optimize query performance. This guide covers how to analyze slow queries, interpret execution plans, and apply performance improvements using techniques like EXPLAIN, slow query logs, and schema analysis.

Analyzing Slow Queries from the Terminal

When connected to a MySQL server from a terminal, you can use native SQL statements and built-in features to analyze the performance of specific queries. This is ideal for diagnosing issues in staging or production without needing container access.

Connect to your MySQL instance via terminal

To begin, log in to your MySQL server using the MySQL client:

```
mysql -u <username> -h <host> -p
```

You'll be prompted for the password. Once inside, you can start analyzing queries.

Use EXPLAIN to view the execution plan

The EXPLAIN keyword shows how MySQL plans to execute a query. It breaks down how tables are accessed and joined, whether indexes are used, and how many rows are expected to be scanned.

```
EXPLAIN SELECT * FROM orders WHERE customer_id = 42;
```

Review the type, key, rows, and Extra columns in the output. Look out for full table scans (type = ALL), which often signal that an index may be missing.

Check current running queries

To view which queries are actively running and their duration, use:

```
SHOW FULL PROCESSLIST;
```

This can help identify long-running or stuck queries in real time.

Analyzing Inside Docker Compose

If your MySQL service is running inside a Docker Compose setup (as Elestio uses), it may not be directly exposed on your host. In this case, analysis must be done from within the container.

Access the MySQL container

Open a shell inside your MySQL container using Docker Compose:

```
docker-compose exec mysql bash
```

This gives you a command-line shell inside the container.

Connect to MySQL from inside the container

Once inside the container, use the environment-defined credentials to access the database:

```
mysql -u $MYSQL_USER -p$MYSQL_PASSWORD $MYSQL_DATABASE
```

This gives you the same SQL interface as from the host terminal, enabling you to use EXPLAIN, SHOW PROCESSLIST, and performance schema tools.

Enable and view the slow query log

MySQL can log slow queries to a file. This must be enabled in your container's `my.cnf` configuration file:

```
[mysqld]
slow_query_log = 1
slow_query_log_file = /var/log/mysql/slow.log
long_query_time = 1
```

After applying these settings, restart the container. Slow queries taking longer than `long_query_time` (in seconds) will be logged.

You can then inspect the log file:

```
cat /var/log/mysql/slow.log
```

Using Performance Schema

MySQL's performance schema and built-in commands help track query statistics over time. This is useful when diagnosing repeat offenders or inefficient patterns.

Enable the performance schema (if not already)

Ensure performance_schema is enabled in your MySQL configuration:

```
[mysqld]
performance_schema=ON
```

Restart the container after updating the config.

Identify top queries using statement summaries

This SQL query shows which SQL statements have the longest average execution times:

```
SELECT digest_text, count_star, avg_timer_wait/1000000000000 AS avg_time_sec
FROM performance_schema.events_statements_summary_by_digest
ORDER BY avg_timer_wait DESC
LIMIT 10;
```

This helps you find the most resource-intensive queries over time.

Understanding the MySQL Execution Plan

Reading the output of EXPLAIN is essential to understand how MySQL processes your query and whether it is using indexes efficiently.

Key output fields to interpret:

- **type**: The join type. Prefer ref, range, or const over ALL (which indicates a full table scan).
- **key**: The index used for the query. A NULL value may indicate a missing index.
- **rows**: Estimated number of rows MySQL will scan. Lower is better.
- **Extra**: Look for warnings like Using temporary or Using filesort, which may suggest suboptimal queries.

Use `EXPLAIN ANALYZE` (available in MySQL 8.0+) to see actual vs. estimated performance:

```
EXPLAIN ANALYZE SELECT * FROM orders WHERE customer_id = 42;
```

Optimizing Queries for Better Performance

Once you've identified inefficient queries, optimization involves rewriting queries, adding indexes, or adjusting the schema.

Common techniques:

- **Add indexes** to columns frequently used in WHERE, JOIN, and ORDER BY.
- **Avoid SELECT*** : Only fetch columns you need to reduce I/O.
- **Use LIMIT** when fetching preview data or paginated results.
- **Re-write joins or subqueries** to reduce temporary tables and filesort operations.
- **Update statistics** with ANALYZE TABLE:

```
ANALYZE TABLE orders;
```

Revision #2

Created 30 April 2025 08:35:19 by kaiwalya

Updated 30 April 2025 08:59:03 by kaiwalya