# Detect and terminate long-running queries

Long-running queries in MySQL can degrade database performance by consuming system resources like CPU, memory, and disk I/O for extended periods. In production environments such as Elestio, it’s essential to monitor and manage these queries effectively to maintain responsiveness and avoid service disruptions. This guide explains how to detect, analyze, and safely terminate long-running queries in MySQL using terminal tools, Docker Compose setups, and built-in logging features. It also includes preventive strategies for avoiding such queries in the future.

## Monitoring Long-Running Queries

When connected to your MySQL instance via the terminal using the MySQL CLI, you can inspect active sessions and identify queries that have been running for an excessive duration. This is useful for spotting inefficient or blocked operations.

To check all current sessions and running queries, execute:

```
SHOW FULL PROCESSLIST;
```

This will return all client connections along with their process ID (Id), command type (Command), execution time in seconds (Time), and the actual SQL query (Info). The Time column indicates how long each query has been executing, allowing you to prioritize the longest-running ones.

If you want to isolate queries that have been running for more than a certain duration, such as 60 seconds, you can filter using the `information_schema.processlist` view:

```
SELECT * FROM information_schema.processlist
WHERE COMMAND != 'Sleep' AND TIME > 60;
```

This command excludes idle connections and focuses only on active queries that may need attention.

## Terminating Long-Running Queries Safely

Once you've identified a query that's taking too long, MySQL allows you to stop it using its process ID (Id). This can be done either by cancelling just the query or by killing the entire connection.

To stop the query and leave the connection active, run:

```
KILL QUERY <Id>;
```

This interrupts the execution of the current query but keeps the client connection open.

If the connection is completely stuck or no longer needed, you can terminate it entirely:

```
KILL CONNECTION <Id>;
```

Use this approach with caution, especially in shared environments, as it may interrupt ongoing operations or cause errors for connected applications.

# Managing Long-Running Queries

If MySQL is running in a Docker Compose setup on Elestio, you'll first need to access the container to inspect queries. You can do so by opening a shell inside the MySQL container:

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

Once inside the container, connect to the MySQL service using the credentials defined in your environment:

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

After connecting, you can run the same SHOW FULL PROCESSLIST and KILL commands to identify and handle long-running queries directly from inside the container environment. The logic and process are identical; the only difference is that you're executing these operations within the container shell.

# Using Slow Query Logs

MySQL supports slow query logging, which records statements that exceed a specified execution time. This is useful for long-term analysis and identifying recurring performance issues.

To enable this feature, update your MySQL configuration file (e.g., my.cnf or mysqld.cnf) with the following lines:

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

This setup logs any query taking longer than one second. Once configured, restart the MySQL service to apply the changes.

You can then inspect the log with:

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

To summarize patterns in slow queries, use the mysqldumpslow tool:

```
mysqldumpslow /var/log/mysql/slow.log
```

This helps you identify repetitive or particularly expensive SQL statements based on execution time and frequency.

# Analyzing Expensive Queries Over Time

To gain visibility into queries that are consistently slow over time, enable MySQL's performance_schema. This built-in feature aggregates statistics about SQL statement execution, allowing you to pinpoint inefficiencies.

Make sure performance schema is enabled in your config:

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

Once it's active, use this query to analyze the most time-consuming query patterns:

```
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 highlights SQL statements that are not just slow once, but frequently expensive, helping you focus on queries with the biggest overall impact.

# Best Practices to Prevent Long-Running Queries

It’s better to prevent long-running queries than to reactively terminate them. A few strategic adjustments in your query design and database configuration can significantly improve performance.

- **Index critical columns** used in WHERE, JOIN, and ORDER BY clauses to speed up lookups and sorting.
- **Avoid SELECT *** in queries — fetch only the necessary columns to reduce result size and memory usage.
- **Use EXPLAIN** to analyze how a query will be executed and whether indexes are being used

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

- **Limit result sets** in user-facing tools or admin dashboards using LIMIT clauses to avoid returning large datasets unnecessarily.
- **Set execution time limits** at the session level

```
SET SESSION MAX_EXECUTION_TIME = 2000; -- in milliseconds
```

- **Implement timeouts in applications** and ORMs to prevent client-side hanging when the database becomes slow.
- **Monitor actively** using slow query logs, processlist views, and the performance schema. Consider integrating this into your monitoring stack to set up alerts for unusually long or frequent queries.

---

Revision #1
Created 2025-04-30 08:54:52 UTC by kaiwalya
Updated 2025-04-30 08:58:24 UTC by kaiwalya