Maulik Bhatt

Percona Monitoring And Management Tool (For MySQL)

Contents

[Percona Monitoring And Management Tool (For MySQL) Documentation 4](#_Toc191894037)

[PMM Server and Client Setup Documentation 4](#_Toc191894038)

[Prerequisites 4](#_Toc191894039)

[Step 1: Set up PMM Server using Docker Desktop UI 4](#_Toc191894040)

[Step 2: Set up PMM Client on Ubuntu 4](#_Toc191894041)

[Step 3: Open Required Ports on Windows Defender Firewall 5](#_Toc191894042)

[Step 4: Connect PMM Client to PMM Server 5](#_Toc191894043)

[Step 5: Add MySQL Instance to PMM Monitoring 6](#_Toc191894044)

[Conclusion 6](#_Toc191894045)

[Comparison of MySQL Built-in Performance Monitoring Tools vs PMM Tool for MySQL 6](#_Toc191894046)

[MySQL Built-in Performance Monitoring Tools 6](#_Toc191894047)

[Overview 6](#_Toc191894048)

[Key Features 7](#_Toc191894049)

[Advantages 7](#_Toc191894050)

[Limitations 7](#_Toc191894051)

[Percona Monitoring and Management (PMM) Tool 7](#_Toc191894052)

[Overview 7](#_Toc191894053)

[Key Features 7](#_Toc191894054)

[Advantages 8](#_Toc191894055)

[Limitations 8](#_Toc191894056)

[Feature Comparison 8](#_Toc191894057)

[Conclusion 9](#_Toc191894058)

[Which is Better and Why? 9](#_Toc191894059)

[Conclusion 9](#_Toc191894060)

[MySQL High CPU Usage Investigation in PMM (Query Analytics - QAN) 10](#_Toc191894061)

[Step 1: Simulating High CPU Usage with Bulk Insert Query 10](#_Toc191894062)

[Step 2: Measuring CPU Usage from Prometheus Metrics 11](#_Toc191894063)

[Step 3: Checking PMM Query Analytics (QAN) 11](#_Toc191894064)

[🔹 Viewing the Query in QAN 11](#_Toc191894065)

[🔹 Query Details in PMM QAN 11](#_Toc191894066)

[Metrics Explanation 11](#_Toc191894067)

[Step 4: Analyzing Query Impact in PMM Dashboards 12](#_Toc191894068)

[🔹 Check Query Load in QAN 12](#_Toc191894069)

[🔹 Analyze MySQL Instance in PMM 13](#_Toc191894070)

[🔹 Check CPU Usage Impact 13](#_Toc191894071)

[Step 5: Identifying the Root Cause of High CPU Usage in PMM 13](#_Toc191894072)

[🔍 Step 5.1: Identify the Primary Cause Using PMM Dashboards 13](#_Toc191894073)

[📌 Step 5.2: Verify if the Identified Cause Matches the Original High CPU Spike 14](#_Toc191894074)

[Conclusion 14](#_Toc191894075)

[Query Analysis Documentation 14](#_Toc191894076)

[Query 14](#_Toc191894077)

[Query Analysis 15](#_Toc191894078)

[Query Time Distribution 15](#_Toc191894079)

[Metrics 15](#_Toc191894080)

[Explanation of Metrics 15](#_Toc191894081)

[EXPLAIN Table 15](#_Toc191894082)

[Explanation of EXPLAIN Table Fields 15](#_Toc191894083)

[Conclusion 16](#_Toc191894084)

[Optimizing MySQL Queries Using Percona Monitoring and Management (PMM) 16](#_Toc191894085)

[Overview 16](#_Toc191894086)

[1. Query Analytics (QAN) 16](#_Toc191894087)

[Accessing QAN in PMM: 16](#_Toc191894088)

[Key Metrics to Look For: 16](#_Toc191894089)

[Optimization Tips: 17](#_Toc191894090)

[2. Slow Query Log Analysis in PMM 17](#_Toc191894091)

[Viewing Slow Queries in PMM: 17](#_Toc191894092)

[Optimization Insights: 17](#_Toc191894093)

[3. Index Usage Monitoring in PMM 17](#_Toc191894094)

[Viewing Index Efficiency: 17](#_Toc191894095)

[4. Performance Insights Using PMM Dashboards 17](#_Toc191894096)

[Checking Database Performance Metrics: 17](#_Toc191894097)

[Using PMM for Query Optimization: 17](#_Toc191894098)

[Conclusion 18](#_Toc191894099)

[PMM Enterprise for MySQL Monitoring 18](#_Toc191894100)

[Enterprise vs. Free Version 18](#_Toc191894101)

[Enterprise Features of PMM for MySQL 18](#_Toc191894102)

[Is PMM Enterprise Worth It for MySQL Monitoring? 18](#_Toc191894103)

[Enterprise Dashboards and Metrics in PMM 18](#_Toc191894104)

[Advanced MySQL Dashboards 18](#_Toc191894105)

[Enterprise Query Analytics 19](#_Toc191894106)

[Additional Enhancements in PMM Enterprise 19](#_Toc191894107)

# Percona Monitoring And Management Tool (For MySQL) Documentation

# PMM Server and Client Setup Documentation

## Prerequisites

* Docker Desktop installed on Windows
* Ubuntu 20.04 system for PMM Client

## Step 1: Set up PMM Server using Docker Desktop UI

1. Open Docker Desktop on your Windows machine.
2. Ensure you are using Linux containers.
3. Pull the PMM Server image version 2.41.2:

* docker pull percona/pmm-server:2.41.2

1. Open the Docker Desktop UI and create a new container:
   * Click on “Containers/Apps” in the left sidebar.
   * Click on the “Run” button to create a new container.
   * In the “Image” field, enter percona/pmm-server:2.41.2.
   * In the “Container Name” field, enter PMM-Server.
   * Under “Optional Settings,” specify the ports:
     + Enter 0 to assign randomly generated host ports.
2. Click on the “Run” button to start the container.
3. Note the randomly assigned host ports for HTTP and HTTPS. For example, if the ports are assigned as 55000 for HTTP and 55001 for HTTPS, use these ports in the next steps.

## Step 2: Set up PMM Client on Ubuntu

1. On your Ubuntu machine, download the PMM Client package using wget:

* wget https://downloads.percona.com/downloads/pmm2/2.41.2/binary/debian/focal/x86\_64/pmm2-client\_2.41.2-6.focal\_amd64.deb?\_gl=1\*1gyi662\*\_gcl\_au\*MjY1MzA3MzkwLjE3Mzk1MDkzNDc.

1. Install the PMM Client package using dpkg:

* sudo dpkg -i pmm2-client\_2.41.2-6.focal\_amd64.deb

1. Verify the PMM Client installation by checking the version:

* pmm-admin --version

1. The output should display the PMM Client version 2.41.2, confirming the installation was successful.

* ProjectName: pmm-admin  
  Version: 2.41.2  
  PMMVersion: 2.41.2

## Step 3: Open Required Ports on Windows Defender Firewall

Before connecting the PMM Client to the PMM Server, ensure that the necessary ports are open on your Windows machine using Windows Defender Firewall.

1. Open the Windows Defender Firewall settings:
   * Click the Start button and type “Windows Defender Firewall”.
   * Select “Windows Defender Firewall with Advanced Security”.
2. Create a new inbound rule:
   * In the left sidebar, click “Inbound Rules”.
   * In the right sidebar, click “New Rule”.
3. Select the rule type:
   * Choose “Port” and click “Next”.
4. Specify the ports:
   * Select “TCP”.
   * In the “Specific local ports” field, enter the ports assigned to the PMM Server (e.g., 55000 and 55001).
   * Click “Next”.
5. Allow the connection:
   * Select “Allow the connection” and click “Next”.
6. Specify the profiles:
   * Check all profiles (Domain, Private, Public) and click “Next”.
7. Name the rule:
   * Enter a name for the rule (e.g., “PMM Server Ports”).
   * Click “Finish”.

Repeat the steps to create another inbound rule for the HTTPS port if needed.

## Step 4: Connect PMM Client to PMM Server

1. On your Ubuntu machine, configure the PMM Client to connect to the PMM Server:

* sudo pmm-admin config --server-insecure-tls --server-url=https://admin:admin@<PMM-Server-IP>:<PMM-Server-HTTPS-Port>
* Replace <PMM-Server-IP> with the IP address of your PMM Server and <PMM-Server-HTTPS-Port> with the HTTPS port assigned to the PMM Server (e.g., 55001).

1. Verify the connection to the PMM Server:

* pmm-admin status
* The output should confirm that the PMM Client is connected to the PMM Server.

## Step 5: Add MySQL Instance to PMM Monitoring

1. On your Ubuntu machine, add a MySQL instance to PMM monitoring:

* sudo pmm-admin add mysql --username=<MySQL-Username> --password=<MySQL-Password> --host=<MySQL-Host> --port=<MySQL-Port>
* Replace <MySQL-Username>, <MySQL-Password>, <MySQL-Host>, and <MySQL-Port> with your MySQL instance’s credentials and connection details.

1. Verify that the MySQL instance has been added:

* pmm-admin list
* The output should list the MySQL instance, confirming that it has been successfully added to PMM monitoring.

## Conclusion

You have successfully set up the PMM Server using Docker Desktop UI, the PMM Client on an Ubuntu system, opened the necessary ports on Windows Defender Firewall, connected the PMM Client to the PMM Server, and added a MySQL instance to PMM monitoring.

# Comparison of MySQL Built-in Performance Monitoring Tools vs PMM Tool for MySQL

## MySQL Built-in Performance Monitoring Tools

### Overview

MySQL provides several built-in performance monitoring tools that are available out-of-the-box. These tools include:

1. **Performance Schema:** A feature for monitoring MySQL server performance. It provides a way to inspect the internal execution of the server at runtime.
2. **Information Schema:** A collection of views that provide information about the database schema, configuration, and status.
3. **SHOW STATUS and SHOW VARIABLES Commands:** Commands to retrieve server status and configuration variables.
4. **EXPLAIN Command:** Provides information about how MySQL executes a query, which helps in query optimization.
5. **Slow Query Log:** Logs queries that take longer than a specified time to execute.

### Key Features

* **Performance Schema:**
  + Monitors various aspects of the MySQL server, including query execution, memory usage, mutexes, and I/O operations.
  + Provides detailed and granular performance data.
  + Can be configured to collect specific types of data.
* **Information Schema:**
  + Provides metadata about database objects such as tables, columns, indexes, and more.
  + Includes views for monitoring server status and performance.
* **SHOW STATUS and SHOW VARIABLES:**
  + SHOW STATUS returns server status information such as the number of connections, queries, and uptime.
  + SHOW VARIABLES provides configuration variables and their current values.
* **EXPLAIN Command:**
  + Analyzes and displays the execution plan of a query.
  + Helps identify performance bottlenecks in query execution.
* **Slow Query Log:**
  + Logs queries that exceed a specified execution time.
  + Useful for identifying and optimizing slow-running queries.

### Advantages

* No additional software installation required.
* Integrated with MySQL server.
* Provides detailed and low-level performance data.
* Configurable and flexible.

### Limitations

* Can be complex to configure and use effectively.
* Performance Schema can introduce overhead on the server.
* Limited in terms of visualization and user-friendly interfaces.
* Requires manual analysis and interpretation of data.

## Percona Monitoring and Management (PMM) Tool

### Overview

Percona Monitoring and Management (PMM) is an open-source monitoring and management tool specifically designed for MySQL and MongoDB databases. PMM provides comprehensive performance monitoring, query analytics, and database management capabilities.

### Key Features

* **Query Analytics:**
  + Provides detailed query performance metrics and visualizations.
  + Helps identify slow queries, analyze query execution plans, and optimize query performance.
* **System and Database Metrics:**
  + Collects and displays metrics for system performance (CPU, memory, disk I/O) and database performance (queries, connections, replication).
  + Provides real-time and historical performance data.
* **Dashboards and Visualizations:**
  + Pre-built Grafana dashboards for visualizing performance metrics.
  + Customizable dashboards to suit specific monitoring needs.
* **Alerting and Notifications:**
  + Configurable alerts for performance issues and threshold breaches.
  + Integration with notification channels such as email, Slack, and PagerDuty.
* **Integration with Prometheus:**
  + Uses Prometheus for metric collection and storage.
  + Leverages Prometheus’ powerful querying and alerting capabilities.

### Advantages

* Comprehensive and user-friendly interface.
* Provides deep insights into query performance and system metrics.
* Pre-built dashboards and visualizations for easy monitoring.
* Integrated alerting and notification system.
* Open-source and actively maintained by Percona.

### Limitations

* Requires additional software installation and configuration.
* May introduce some overhead on the monitored systems.
* Learning curve for setting up and customizing the tool.

## Feature Comparison

| Feature | MySQL Built-in Tools | Percona Monitoring and Management (PMM) |
| --- | --- | --- |
| **Ease of Use** | Requires manual configuration and analysis | User-friendly interface with pre-built dashboards |
| **Visualization** | Limited visualization capabilities | Advanced visualizations and customizable dashboards |
| **Query Analytics** | Manual analysis using EXPLAIN and slow query log | Detailed query analytics with visualizations |
| **System Metrics** | Basic system metrics via SHOW STATUS | Comprehensive system and database metrics |
| **Alerting and Notifications** | No built-in alerting | Integrated alerting and notifications system |
| **Integration** | Limited to MySQL server environment | Integration with Prometheus and other tools |
| **Additional Software** | No additional software required | Requires installation and configuration of PMM |

## Conclusion

### Which is Better and Why?

* **Ease of Use:**
  + PMM is generally easier to use and more user-friendly due to its graphical interface and pre-built dashboards.
  + MySQL built-in tools require manual configuration and analysis, which can be complex.
* **Visualization and Dashboards:**
  + PMM provides rich visualizations and customizable dashboards, making it easier to monitor and analyze performance data.
  + MySQL built-in tools lack advanced visualization capabilities.
* **Query Analytics:**
  + PMM offers detailed query analytics, helping identify and optimize slow queries.
  + MySQL built-in tools provide query analysis through EXPLAIN and slow query log, but require manual interpretation.
* **Alerting and Notifications:**
  + PMM includes an integrated alerting system with support for various notification channels.
  + MySQL built-in tools do not have built-in alerting capabilities.
* **Integration and Extensibility:**
  + PMM integrates with Prometheus and other monitoring tools, providing a scalable and extensible monitoring solution.
  + MySQL built-in tools are limited to the MySQL server environment.

### Conclusion

While MySQL built-in performance monitoring tools offer detailed and granular performance data, they can be complex to configure and use effectively. PMM, on the other hand, provides a comprehensive, user-friendly, and feature-rich monitoring solution for MySQL databases. PMM is generally better suited for monitoring and managing MySQL performance due to its ease of use, advanced visualizations, query analytics, and integrated alerting capabilities.

In conclusion, if you are looking for a robust and easy-to-use performance monitoring solution with advanced features and visualizations, PMM is the better choice. However, if you prefer a lightweight and integrated approach without additional software installation, MySQL built-in tools may be sufficient for your needs.

# **MySQL High CPU Usage Investigation in PMM (Query Analytics - QAN)**

## **Step 1: Simulating High CPU Usage with Bulk Insert Query**

To analyze high CPU usage in PMM, we executed the following bulk **INSERT query**:

INSERT INTO test\_cpu (id, name)  
SELECT  
 t.n,  
 CONCAT('CPU\_', t.n)  
FROM  
 (  
 SELECT  
 (  
 a.N + (10 \* b.N) + (100 \* c.N) + (1000 \* d.N) + (10000 \* e.N)  
 ) AS n  
 FROM  
 (  
 SELECT 0 AS N UNION ALL SELECT 1 UNION ALL SELECT 2 UNION ALL SELECT 3 UNION ALL SELECT 4 UNION ALL SELECT 5 UNION ALL SELECT 6 UNION ALL SELECT 7 UNION ALL SELECT 8 UNION ALL SELECT 9  
 ) a,  
 (  
 SELECT 0 AS N UNION ALL SELECT 1 UNION ALL SELECT 2 UNION ALL SELECT 3 UNION ALL SELECT 4 UNION ALL SELECT 5 UNION ALL SELECT 6 UNION ALL SELECT 7 UNION ALL SELECT 8 UNION ALL SELECT 9  
 ) b,  
 (  
 SELECT 0 AS N UNION ALL SELECT 1 UNION ALL SELECT 2 UNION ALL SELECT 3 UNION ALL SELECT 4 UNION ALL SELECT 5 UNION ALL SELECT 6 UNION ALL SELECT 7 UNION ALL SELECT 8 UNION ALL SELECT 9  
 ) c,  
 (  
 SELECT 0 AS N UNION ALL SELECT 1 UNION ALL SELECT 2 UNION ALL SELECT 3 UNION ALL SELECT 4 UNION ALL SELECT 5 UNION ALL SELECT 6 UNION ALL SELECT 7 UNION ALL SELECT 8 UNION ALL SELECT 9  
 ) d,  
 (  
 SELECT 0 AS N UNION ALL SELECT 1 UNION ALL SELECT 2 UNION ALL SELECT 3 UNION ALL SELECT 4 UNION ALL SELECT 5 UNION ALL SELECT 6 UNION ALL SELECT 7 UNION ALL SELECT 8 UNION ALL SELECT 9  
 ) e  
 ) t;

This query inserted **100,000 records**, creating a **high CPU load** in MySQL.

## **Step 2: Measuring CPU Usage from Prometheus Metrics**

To check the **CPU usage caused by this query**, we use the following **Prometheus query**:

100 \* rate(process\_cpu\_seconds\_total{service\_name="mysql"}[5m]) / count(node\_cpu\_seconds\_total{mode="user"})

This query calculates **MySQL’s CPU consumption** as a percentage of total user CPU time over the last **5 minutes**.

## **Step 3: Checking PMM Query Analytics (QAN)**

### **🔹 Viewing the Query in QAN**

1. Navigate to **PMM → Query Analytics (QAN)**.
2. Set the **time range** to match the period when the **CPU usage spiked**.
3. Filter queries by **highest CPU Time**.
4. Locate the **INSERT INTO test\_cpu** query.

### **🔹 Query Details in PMM QAN**

| **Metric** | **Value** |
| --- | --- |
| **Query Count** | 1 execution (low QPS) |
| **Total Query Time** | 1.63 sec (97.51% of total load) |
| **Avg Query Time** | 1.63 sec per execution |
| **Lock Time** | 288.00 µs (19.73% of total) |
| **Rows Examined** | 50 rows per execution |
| **Full Join** | 4 per query |
| **Full Scan** | 1 per query |
| **No index used** | 1 |
| **Rows Affected** | 100,000 rows per execution |
| **Tmp Tables** | 5 per query |

### Metrics Explanation

#### **Query Count**

* **Rate/Second**: The number of queries executed per second.
* **Sum**: Total number of queries executed.
* **Per Query Stats**: The average number of queries executed per query.

#### **Query Time**

* **Rate/Second**: The amount of time spent executing queries per second.
* **Sum**: Total time taken by all queries.
* **Per Query Stats**: The average time taken per query.

#### **Lock Time**

* **Rate/Second**: The amount of time queries were locked per second.
* **Sum**: Total lock time for all queries.
* **Per Query Stats**: The average lock time per query.

#### **Rows Examined**

* **Rate/Second**: The number of rows examined per second.
* **Sum**: Total number of rows examined by all queries.
* **Per Query Stats**: The average number of rows examined per query.

#### **Full Join**

* **Rate/Second**: The number of full joins executed per second.
* **Sum**: Total number of full joins executed by all queries.
* **Per Query Stats**: The average number of full joins per query.

#### **Full Scan**

* **Rate/Second**: The number of full scans executed per second.
* **Sum**: Total number of full scans executed by all queries.
* **Per Query Stats**: The average number of full scans per query.

#### **No Index Used**

* **Rate/Second**: The number of queries executed without using an index per second.
* **Sum**: Total number of queries executed without using an index.
* **Per Query Stats**: The average number of queries executed without using an index per query.

#### **Rows Affected**

* **Rate/Second**: The number of rows affected by queries per second.
* **Sum**: Total number of rows affected by all queries.
* **Per Query Stats**: The average number of rows affected per query.

#### **Tmp Tables**

* **Rate/Second**: The number of temporary tables created per second.
* **Sum**: Total number of temporary tables created by all queries.
* **Per Query Stats**: The average number of temporary tables created per query.

## **Step 4: Analyzing Query Impact in PMM Dashboards**

### **🔹 Check Query Load in QAN**

1. Navigate to **PMM → Query Analytics (QAN)**.
2. Sort queries by **CPU Time** and **Execution Time**.
3. Click on the **INSERT INTO test\_cpu** query to check **Query Execution Stats**.

### **🔹 Analyze MySQL Instance in PMM**

1. Navigate to **PMM → MySQL Instance Summary**.
2. Check the following key metrics:
   * **Threads Running** → Spikes indicate contention.
   * **Temporary Tables Created** → High numbers suggest query inefficiencies.
   * **InnoDB Buffer Pool Usage** → Monitors memory impact.

### **🔹 Check CPU Usage Impact**

1. Navigate to **PMM → Node Summary**.
2. Look for:
   * **CPU Usage (%)** – Is MySQL consuming most of it?
   * **I/O Wait (%)** – High values indicate storage bottlenecks.
   * **Memory Usage** – Ensures MySQL isn’t swapping.

## **Step 5: Identifying the Root Cause of High CPU Usage in PMM**

Now that we have analyzed MySQL’s performance metrics in PMM QAN, MySQL Instance Summary, and Node Summary, let’s correlate this data to determine the exact cause of high CPU usage.

### **🔍 Step 5.1: Identify the Primary Cause Using PMM Dashboards**

#### **1️⃣ Check PMM → MySQL Instance Summary**

* Look at **Threads Running** → Are too many active queries causing CPU contention?
* Check **Temporary Tables Created** → Are excessive temp tables slowing execution?
* Analyze **Handler Read Metrics** → High **Handler\_read\_rnd\_next** may indicate full table scans.

#### **2️⃣ Check PMM → MySQL InnoDB Metrics**

* Look at **Buffer Pool Reads & Writes** → Is the buffer pool overloaded?
* Check **Row Operations** → High row operations may indicate inefficient queries.
* Analyze **InnoDB Log Waits** → High values suggest transaction bottlenecks.

#### **3️⃣ Check PMM → Node Summary**

* Look at **CPU Usage by Process** → Is MySQL the top CPU consumer?
* Check **I/O Wait Percentage** → High I/O wait suggests storage bottlenecks.
* Analyze **Load Average** → If consistently high, MySQL is overloading the system.

### **📌 Step 5.2: Verify if the Identified Cause Matches the Original High CPU Spike**

* Cross-check findings with **PMM QAN, MySQL Instance Summary, and Node Summary**.
* If MySQL’s CPU consumption aligns with the **INSERT query execution time**, we have confirmed our high CPU usage cause.
* If multiple queries contribute, analyze them separately in **QAN**.

## **Conclusion**

Based on the metrics, the **INSERT INTO test\_cpu** query significantly impacted the CPU usage:

* **High Query Time**: The query took 1.63 seconds on average, accounting for 97.51% of the total load.
* **Rows Affected**: The query affected 100,000 rows, indicating a high volume of data processing.
* **Full Join and Full Scan**: The query involved multiple full joins and full scans, which are CPU-intensive operations.
* **Temporary Tables**: The query created 5 temporary tables, suggesting inefficiencies.

Overall, the query is confirmed to be a major contributor to high CPU usage.

# Query Analysis Documentation

**Date:** 2025-02-27  
**Analyst:** MaulikBhatt7

## Query

SELECT  
 ProductName,  
 SaleDate,  
 Quantity,  
 Price,  
 Discount  
FROM  
 ProductSales  
WHERE  
 SaleDate BETWEEN '2024-01-01'  
 AND '2024-12-31' -- Indexed  
 AND CustomerID = 101 -- Indexed  
 AND Quantity > 10 -- Not Indexed  
 AND Price > 50.00 -- Not Indexed  
 AND Discount < 5.00  
LIMIT  
 0, 50000

## Query Analysis

### Query Time Distribution

* **Lock Time:** 33.98%
* **Other:** 66.02%

### Metrics

| Metric | Rate/Second | Sum | Per Query Stats |
| --- | --- | --- | --- |
| **Query Count** | <0.01 QPS | 2.00 | 12.5% of total, 1.00 |
| **Query Time** | <0.01 load | 1.09 ms | 1.08% of total, 545.90 µs |
| **Lock Time** | <0.01 | 371.00 µs | 7.11% of total, 33.98% of query time, 185.50 µs |
| **Rows Sent** | <0.01 | 2.00 | 4.88% of total, 1.00 |
| **Rows Examined** | <0.01 | 2.00 | 0.85% of total, 1.00 per row sent, 1.00 |

### Explanation of Metrics

* **Query Count:** The number of times the query was executed.
* **Query Time:** The total time taken to execute the query.
* **Lock Time:** The time during which the query was waiting for locks.
* **Rows Sent:** The number of rows returned by the query.
* **Rows Examined:** The number of rows examined to return the result set.

### EXPLAIN Table

The EXPLAIN statement provides information about how MySQL executes a query. Below is the EXPLAIN output for the given query:

| id | select\_type | table | partitions | type | possible\_keys | key | key\_len | ref | rows | filtered | Extra |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | SIMPLE | ProductSales | NULL | ref | idx\_SaleDate,idx\_CustomerID | idx\_CustomerID | 4 | const | 2 | 3.70 | Using where |

### Explanation of EXPLAIN Table Fields

* **id:** The SELECT identifier. This is useful for identifying which SELECT statement corresponds to which row in the output.
* **select\_type:** The type of SELECT query. In this case, SIMPLE indicates a simple SELECT query without any subqueries or unions.
* **table:** The table to which the row of output refers.
* **partitions:** The partitions that are accessed by the query. NULL indicates that partitions are not used.
* **type:** The join type. ref indicates that a non-unique index or a unique index (with a constant) is used.
* **possible\_keys:** The indexes that could be used to find the rows in the table.
* **key:** The key actually used to find the rows.
* **key\_len:** The length of the key used.
* **ref:** The columns or constants compared to the key.
* **rows:** The number of rows MySQL believes it must examine to execute the query.
* **filtered:** An estimate of the percentage of rows that will be returned by the query.
* **Extra:** Additional information about how MySQL will execute the query. Using where indicates that a WHERE clause is being used to filter rows.

## Conclusion

This query retrieves product sales data for the year 2024 for a specific customer (CustomerID = 101) with additional conditions on quantity, price, and discount. The analysis indicates that the query is efficient in terms of rows examined and rows sent, but lock time constitutes a significant portion of the query time, which might need optimization.

# Optimizing MySQL Queries Using Percona Monitoring and Management (PMM)

## Overview

Percona Monitoring and Management (PMM) is a powerful tool for monitoring and optimizing MySQL queries. It provides insights into query performance, slow queries, execution plans, and system resource usage.

## 1. Query Analytics (QAN)

### Accessing QAN in PMM:

1. Open the PMM Web UI.
2. Navigate to **Query Analytics** (QAN).
3. Select your MySQL database instance.
4. Analyze the query list based on execution time, load, and frequency.

### Key Metrics to Look For:

* **Query Time:** High execution time indicates slow queries.
* **Rows Examined vs. Rows Sent:** A high difference may indicate missing indexes.
* **Query Load:** Identifies queries consuming the most resources.
* **Lock Time:** Indicates contention issues.

### Optimization Tips:

* Use **indexes** to reduce table scans.
* Avoid **SELECT** \* in queries.
* Optimize **JOINs** by indexing foreign keys.
* Use **EXPLAIN ANALYZE** to understand query execution.

## 2. Slow Query Log Analysis in PMM

### Viewing Slow Queries in PMM:

1. Open PMM and go to **Query Analytics**.
2. Apply a filter for **Slow Queries**.
3. Sort queries by execution time to find the slowest ones.
4. Click on a query to view its execution details.

### Optimization Insights:

* Check **execution plans** provided by PMM.
* Identify **frequently running slow queries**.
* Look for queries causing high resource utilization.

## 3. Index Usage Monitoring in PMM

### Viewing Index Efficiency:

1. Navigate to **Query Analytics**.
2. Select a specific query and review its execution details.
3. Identify if **indexes are used** or if a full table scan occurs.
4. Adjust indexing strategies accordingly.

## 4. Performance Insights Using PMM Dashboards

### Checking Database Performance Metrics:

1. Go to **PMM Dashboard**.
2. Select **MySQL Summary** or **MySQL InnoDB Metrics**.
3. Review key performance indicators like:
   * Query execution time.
   * Query load distribution.
   * Lock wait times.

### Using PMM for Query Optimization:

* Identify queries with **high CPU or I/O usage**.
* Analyze **query execution patterns**.
* Optimize long-running queries using insights from **PMM Query Analytics**.

## Conclusion

By leveraging PMM’s Query Analytics and dashboards, you can identify and optimize inefficient queries to improve database performance. Regularly monitoring key query metrics ensures your MySQL database runs efficiently.

## PMM Enterprise for MySQL Monitoring

### Enterprise vs. Free Version

For MySQL monitoring, the **PMM Enterprise** version mainly provides additional enterprise-grade features, but the core MySQL monitoring capabilities are already available in the **free version**.

### Enterprise Features of PMM for MySQL

Compared to the free version, PMM Enterprise offers:

✅ **Advanced Security** – LDAP, Single Sign-On (SSO), Role-Based Access Control (RBAC). ✅ **Technical Support** – Direct access to Percona’s expert support team. ✅ **Enterprise Dashboards & Metrics** – More in-depth MySQL performance insights. ✅ **Custom Alerts & Reports** – Improved alerting and reporting features. ✅ **Automated Query Tuning** – Helps in optimizing MySQL queries automatically. ✅ **Extended Retention Period** – Longer historical data retention for deeper analysis. ✅ **Integration with Enterprise Tools** – Better compatibility with other monitoring systems.

### Is PMM Enterprise Worth It for MySQL Monitoring?

* If you’re running a **small to mid-sized MySQL setup**, the **free version is sufficient**.
* If you need **enterprise-grade support, security, and deeper insights**, then the **paid version** might be beneficial.

## Enterprise Dashboards and Metrics in PMM

PMM Enterprise offers a range of dashboards and metrics to monitor MySQL performance, enhancing the capabilities of the free version.

### Advanced MySQL Dashboards

These dashboards provide deeper visibility into various aspects of MySQL performance, including: - **Performance Schema** - **Waits Analysis** - **Replication Statistics** - **Table and User Statistics** - **InnoDB Metrics**

### Enterprise Query Analytics

This feature allows for in-depth analysis of query performance, helping identify slow-performing queries and anomalies that might affect applications.

## Additional Enhancements in PMM Enterprise

In addition to core monitoring, PMM Enterprise provides:

* **Advanced Security Features** – Including Single Sign-On (SSO) and LDAP integration.
* **Custom Plugins and Integrations** – Allowing for tailored monitoring solutions.
* **Automated Issue Detection** – With proactive notifications to address potential problems before they impact performance.

These enterprise features provide a more comprehensive monitoring experience, especially beneficial for **larger or more complex MySQL deployments**.