# Metric Reporting

Database Management Systems Laboratory CS39202

26.03.2023

|  |  |
|---|---|
| Utsav Vinay Mehta | 20CS10069 |
| Sidharth Vishwakarma | 20CS10082 |
| Khush Bajaj | 20CS30027 |
| Kulkarni Pranav Suryakant | 20CS30029 |
| Swapnil Yasasvi | 20CS30054 |

# Metric Reporting

## Introduction:

In the world of data processing, measuring the performance and efficiency of query processing is an essential aspect. With the ever-increasing amount of data and complexity of queries, it becomes crucial to measure and monitor the metrics of query processing in real-time. Metric reporting provides a systematic approach to measure and report the performance of query processing, which can help to optimize the queries and database performance.

In this report, we will discuss the concept of metric reporting in query processing, along with a detailed explanation of building a wrapper/interface that collects query processing metrics using the inbuilt commands of Postgres.

## What is Metric Reporting?

Metric reporting refers to the process of collecting, analyzing, and reporting performance metrics of a system. In the context of query processing, metric reporting provides a systematic approach to measure and report the performance of queries. The primary goal of metric reporting is to optimize the queries and database performance by identifying the bottlenecks and areas of improvement.

## Metric Reporting in Postgres:

Postgres is an open-source relational database management system (RDBMS) known for its robustness, reliability, and extensibility. Postgres provides various inbuilt commands to collect the metrics of query processing, which can be used to build a wrapper/interface that collects query processing metrics in real-time.

### The following are the essential metrics that can be collected using Postgres inbuilt commands:

1. Table statistics: Postgres provides the pg_stat_all_tables view, which contains statistics about tables, such as the number of sequential and index scans, tuples fetched, and dead tuples. These statistics can help to identify the most accessed tables, the frequency of table scans, and the amount of data read.
2. CPU/Memory usage: Postgres provides the pg_stat_activity view, which contains information about the current running queries, including the CPU and memory usage of each query. This information can help to identify the queries that consume the most CPU and memory resources.
3. Query Execution Plan: Postgres provides the EXPLAIN command, which can be used to obtain the query execution plan. The execution plan provides information about the query's execution steps, such as the order of table scans, the use of indexes, and the use of sorting and grouping. The execution plan can help to identify the most time-consuming steps in the query processing.

## Building a Wrapper/Interface for Metric Reporting:

### To build a wrapper/interface for metric reporting, we can use the following steps:

1. Connect to the Postgres database using a suitable programming language such as Python.
2. Execute the query and collect the query ID.
3. Use the pg_stat_activity view to collect the CPU and memory usage of the query by filtering the query ID.
4. Use the pg_stat_all_tables view to collect the statistics about the accessed tables by filtering the query ID.
5. Use the EXPLAIN command to obtain the query execution plan.
6. Store the collected metrics in a suitable data structure, such as a dictionary or a JSON object.
7. Display the collected metrics in a suitable format, such as a table or a graph.

## Deliverables -

1. Code for the wrapper/interface: This includes the code written in a suitable programming language such as Python to connect to the Postgres database, execute the query, collect the query processing metrics using inbuilt commands, and store them in a suitable data structure.
2. Documentation: This includes documentation describing the purpose and functionality of the wrapper/interface, installation instructions, and any dependencies required.
3. Test cases: This includes test cases to ensure that the wrapper/interface is working as intended, and the collected metrics are accurate.
4. Sample outputs: This includes sample outputs in a suitable format, such as tables or graphs, to demonstrate how the collected metrics can be visualized and analyzed.
5. User guide: This includes a user guide on how to use the wrapper/interface, including instructions on how to configure and run it, how to interpret the collected metrics, and how to optimize query processing using the metrics.

## Conclusion:

Metric reporting provides a systematic approach to measure and report the performance of query processing, which can help to optimize the queries and database performance. Postgres provides various inbuilt commands to collect the metrics of query processing, which can be used to build a wrapper/interface that collects query processing metrics in real-time. The wrapper/interface can be built using a suitable programming language such as Python, and the collected metrics can be displayed in a suitable format to facilitate analysis and optimization.