Metric Reporting

Database Management Systems Laboratory CS39202

16.04.2023

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

Metric Reporting

Objective:

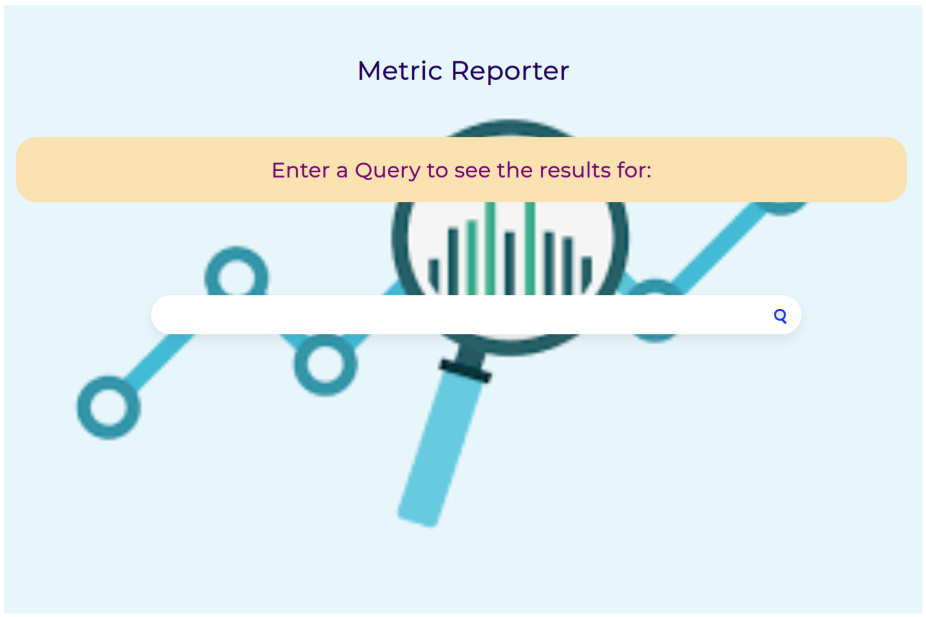
To build a wrapper/interface which collects query processing metrics like table statistics and CPU/memory usage in run time while executing a query. You may use the inbuilt commands of Postgres.

Overview:

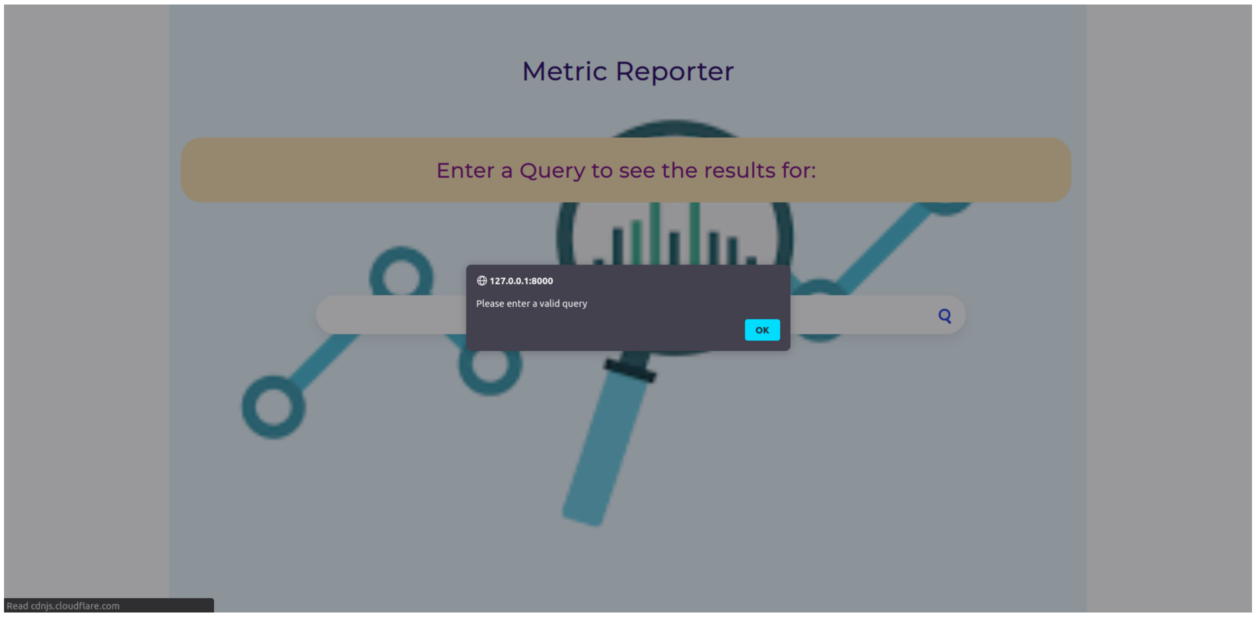
In the world of data processing, measuring the performance and efficiency of query processing is essential. With the ever-increasing amount of data and complexity of queries, it becomes crucial to measure and monitor the metrics of query processing in real time. Metric reporting provides a systematic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

Screenshots:

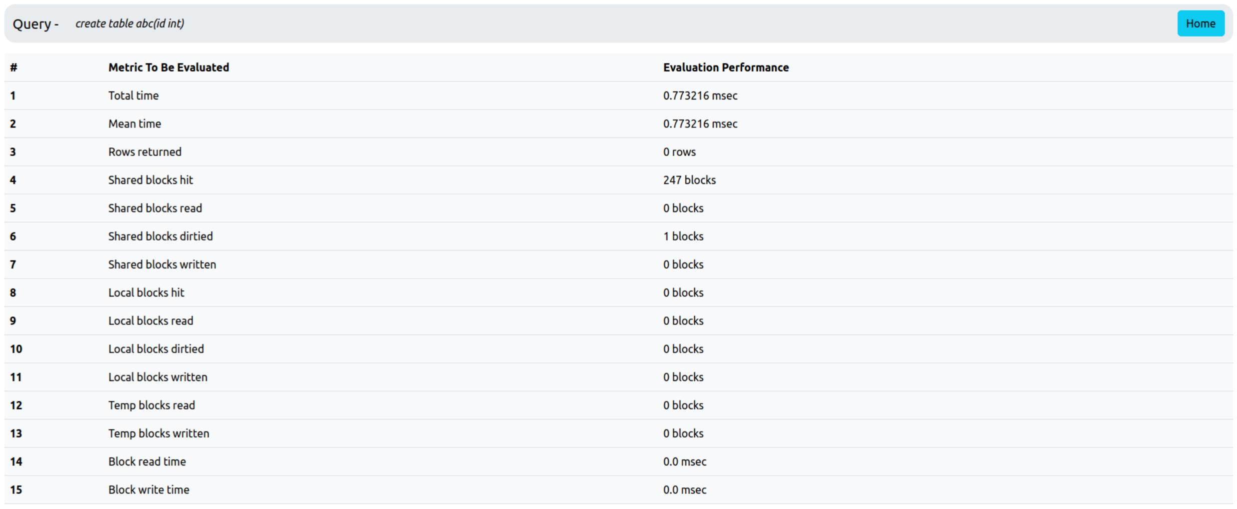


Home Page View


Metric Reporter
Enter a Query to see the results for:
127.0.0.1:8000
OK
Read cdnjs.cloudflare.com

Please enter a valid query

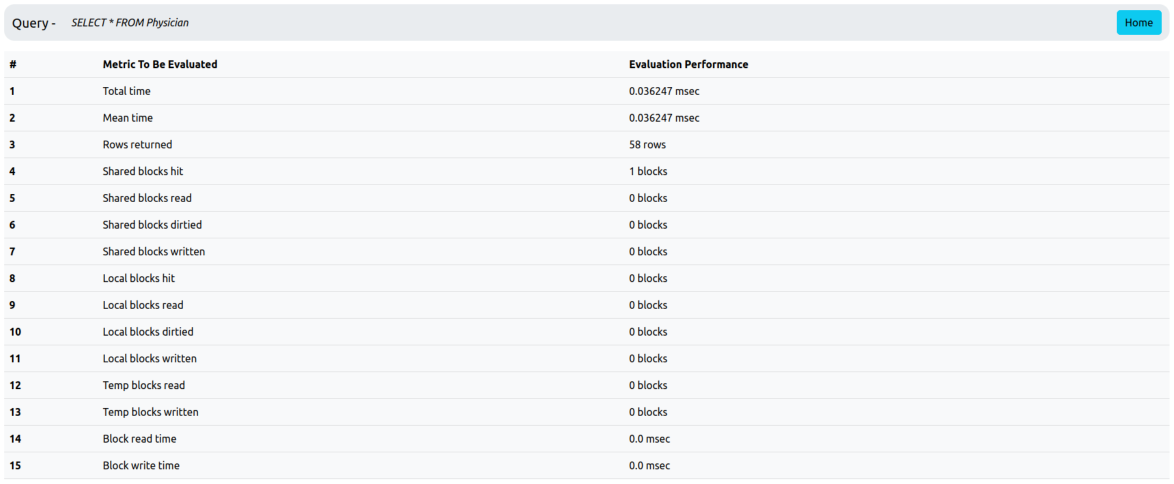
Query Error Message



Query - *create table abc(id int)*

#	Metric To Be Evaluated	Evaluation Performance
1	Total time	0.773216 msec
2	Mean time	0.773216 msec
3	Rows returned	0 rows
4	Shared blocks hit	247 blocks
5	Shared blocks read	0 blocks
6	Shared blocks dirtied	1 blocks
7	Shared blocks written	0 blocks
8	Local blocks hit	0 blocks
9	Local blocks read	0 blocks
10	Local blocks dirtied	0 blocks
11	Local blocks written	0 blocks
12	Temp blocks read	0 blocks
13	Temp blocks written	0 blocks
14	Block read time	0.0 msec
15	Block write time	0.0 msec

Metric Results of the Query - Create


Query - *SELECT * FROM Physician*

#	Metric To Be Evaluated	Evaluation Performance
1	Total time	0.036247 msec
2	Mean time	0.036247 msec
3	Rows returned	58 rows
4	Shared blocks hit	1 blocks
5	Shared blocks read	0 blocks
6	Shared blocks dirtied	0 blocks
7	Shared blocks written	0 blocks
8	Local blocks hit	0 blocks
9	Local blocks read	0 blocks
10	Local blocks dirtied	0 blocks
11	Local blocks written	0 blocks
12	Temp blocks read	0 blocks
13	Temp blocks written	0 blocks
14	Block read time	0.0 msec
15	Block write time	0.0 msec

Metric Results of the Query - Select

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/ Methodology:

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.

PostgreSQL automatically records a large number of statistics about its operation, but in the web application designed by our team we have included all indicators that will give the users information about the functionality and health of their database servers.

For accessing all the required indicators we have used, `pg_stat_statements`. It is an extension that has been around in PostgreSQL since version 8.4. It's evolved over the years (notably since 9.2) to become such a helpful tool.

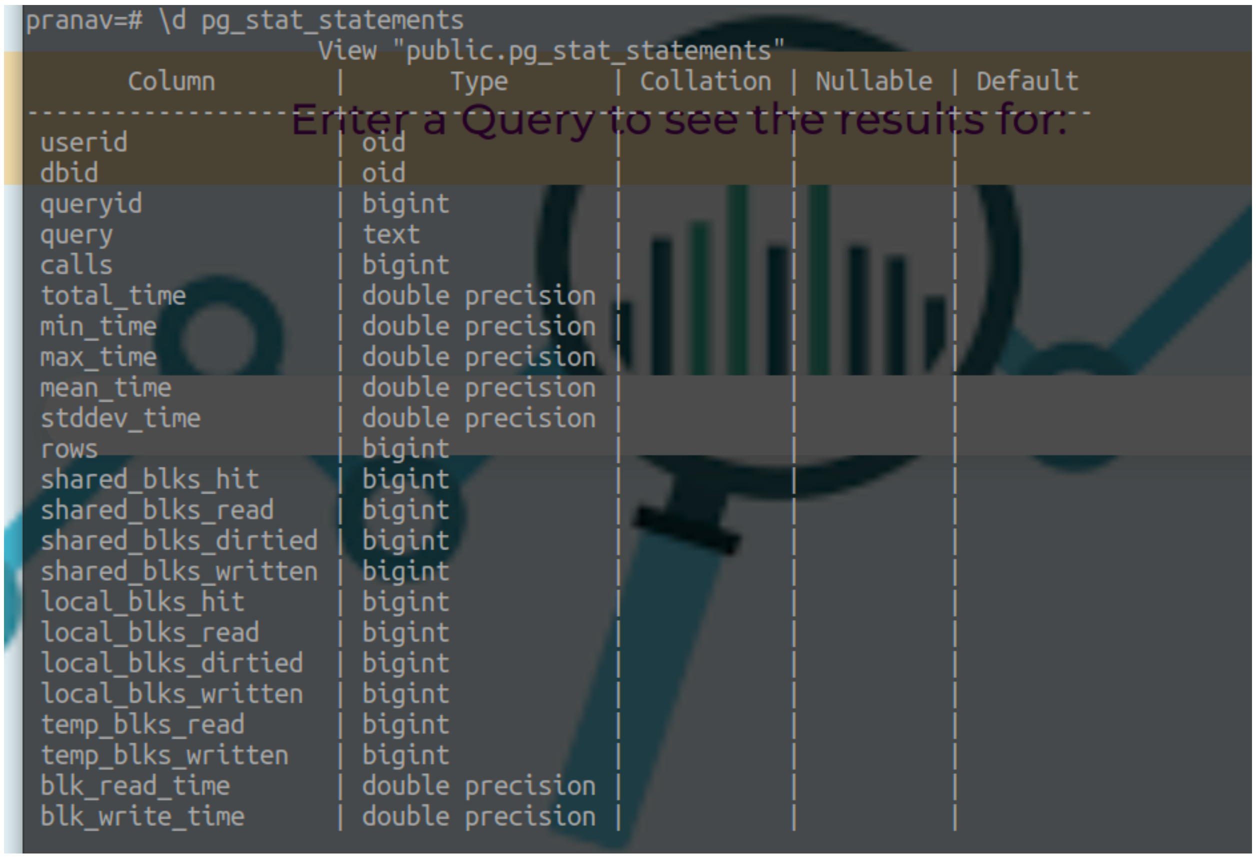
`pg_stat_statements` is included in the contrib module, so it ships with standard Postgres, but might not be automatically enabled.

To enable to pg_stat_statements following procedure is used:

- In the database which we want to access pg_stat_statements, following command is run: CREATE EXTENSION pg_stat_statements;

Although pg_stat_statements tracks across all databases on the same server by default, you do need to activate the extension in each database where you wish to use it.

From the view provided by pg_stat_statements that contains aggregated query statistics, appropriate statistics are used to report the health of the database server.



```
pranav=# \d pg_stat_statements
                    View "public.pg_stat_statements"
       Column        |       Type       | Collation | Nullable | Default
---------------------+------------------+-----------+----------+---------
 userid              | oid              |           |          |
 dbid                | oid              |           |          |
 queryid             | bigint           |           |          |
 query               | text             |           |          |
 calls               | bigint           |           |          |
 total_time          | double precision |           |          |
 min_time            | double precision |           |          |
 max_time            | double precision |           |          |
 mean_time           | double precision |           |          |
 stddev_time         | double precision |           |          |
 rows                | bigint           |           |          |
 shared_blks_hit     | bigint           |           |          |
 shared_blks_read    | bigint           |           |          |
 shared_blks_dirtied | bigint           |           |          |
 shared_blks_written | bigint           |           |          |
 local_blks_hit      | bigint           |           |          |
 local_blks_read     | bigint           |           |          |
 local_blks_dirtied  | bigint           |           |          |
 local_blks_written  | bigint           |           |          |
 temp_blks_read      | bigint           |           |          |
 temp_blks_written   | bigint           |           |          |
 blk_read_time       | double precision |           |          |
 blk_write_time      | double precision |           |          |
```

- `total_exec_time` and `mean_exec_time` are in milliseconds, and calls is the number of times the query has been run. `min_exec_time`, `max_exec_time`, and `stddev_exec_time` are available as well. Before running any query pg_stat_statements is need to be reset using the following command: `SELECT pg_stat_statements_reset();`
- To check how fast the timers in our system are, we used the `pg_test_timing utility`. The overhead of collecting all this timing data is low for most hardware.
- We used the block read and write time (`blk_read_time` and `blk_write_time`) statistics, after turning on the `track_io_timing` parameter in `postgresql.conf`. Since timing operations might be extremely sluggish on some computers, it is disabled by default, however this wasn't the case on our machine.

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.

References:

- PostgreSQL: Documentation: 15: F.32. pg_stat_statements
- Query Optimization in Postgres with pg_stat_statements (crunchydata.com)
- Psycopg – PostgreSQL database adapter for Python — Psycopg 2.9.6 documentation
- https://docs.djangoproject.com/en/4.2/