DATABASE MANAGEMENT SYSTEMS PROJECT: METRIC REPORTING GROUP PHOENIX

Group Members:
Deepiha .S 20CS30015
Devendra Palod 20CS10024
Rohit Kumar Prajapati 20CS30041
Suhas A M 20CS10066
Tanmay Mohanty 20CS10089

1 Introduction

The goal of this project is to build a wrapper/interface which collects query processing metrics like table statistics, CPU/memory usage in runtime while executing a query using inbuilt commands of PostgreSQL. The interface is built using Streamlit, a popular library for building web applications in Python.

2 Code Overview

The code is organized into the following sections:

1. Importing necessary libraries
2. Configuring Streamlit application
3. Connecting to the PostgreSQL database
4. Building the user interface
5. Analyzing the input query and displaying the metrics

2.1 Importing necessary libraries

The following libraries are imported:

- streamlit: The main library for building the web application

- streamlit.components.v1: A module for working with Streamlit components
- PIL.Image: A module for working with images
- pandas: A library for data manipulation and analysis
- json: A library for working with JSON data
- psycopg2: A library for connecting to PostgreSQL databases
- psutil: A library for retrieving system and process metrics
- st_aggrid: A module for integrating the AG-Grid component with Streamlit

2.2 Configuring Streamlit application

The Streamlit application is configured using the set_page_config method, which sets the page title as "Metric Reporting". The default Streamlit menu and footer are hidden using CSS.

2.3 Connecting to the PostgreSQL database

A connection to the PostgreSQL database is established using the psycopg2.connect method, which takes the following parameters:

- database: The name of the database (in this case, "dbms")
- user: The username for connecting to the database (in this case, "testuser")
- password: The password for connecting to the database (in this case, "test")
- host: The hostname of the PostgreSQL server (in this case, "localhost")
- port: The port number on which the PostgreSQL server is running (in this case, "5432")

A cursor object is created for executing SQL commands using the connection.

2.4 Building the user interface

The user interface is built using Streamlit components. The following components are used to build the UI:

- title, header, subheader, and image: These components are used to display the title, header, subheader, and an image on the application.
- text_input: This component is used to accept the SQL query from the user.
- button: This component is used to trigger the analysis of the input query.

2.5 Analyzing the input query and displaying the metrics

When the user clicks the "Analyse" button, the following steps are executed:

1. The `EXPLAIN (ANALYZE, BUFFERS, VERBOSE, FORMAT JSON)` command is appended to the input SQL query to obtain the query plan and execution statistics.
2. The CPU and memory usage at the start of query execution are recorded using `psutil`.
3. The modified query is executed using the cursor object, and the result is fetched.
4. The CPU and memory usage at the end of query execution are recorded.
5. The output rows are processed, and the required metrics are extracted from the JSON data.
6. A throughput metric is calculated based on the actual rows, plan width, and execution time.
7. The metrics are displayed in a two-column layout using Streamlit components. The first column shows the common metrics, like CPU usage, execution time, planning time, and memory usage. The second column shows the plan table metrics using the AG-Grid component.

Common Metrics Displayed:

1. CPU Usage: This metric refers to the percentage of CPU resources used by the query during its execution. It's measured as a percentage of the total CPU resources available at the time of execution. In this code, it's calculated using the 'psutil.cpu_percent()' function before and after executing the query and taking the average.
2. Execution Time: This metric refers to the amount of time it takes for the query to complete its execution. It's measured in milliseconds and is the sum of the planning time and execution time of the query.
3. Planning Time: This metric refers to the time taken by the PostgreSQL query planner to generate the query execution plan. It's measured in milliseconds.
4. Memory Usage (RSS): This metric refers to the amount of physical memory used by the PostgreSQL process to execute the query. It's measured in bytes and is obtained using the 'psutil.Process().memory_info().rss' function.
5. Memory Usage (VMS): This metric refers to the total amount of virtual memory used by the PostgreSQL process to execute the query. It's measured in bytes and is obtained using the 'psutil.Process().memory_info().vms' function.

6. Throughput: This metric refers to the amount of data processed per second by the query. It's measured in MBps (megabytes per second) and is calculated using the formula: '(actual_rows * width * 1000) / (exe_time * 1024 * 1024)', where 'actual_rows' is the number of rows returned by the query, 'width' is the width of each row in bytes, and 'exe_time' is the execution time of the query in milliseconds.

Other Metrics Displayed:

- Node Type: Indicates the type of the node in the query plan. For example, it could be a Seq Scan node, an Index Scan node, a Hash Join node, or a Sort node.
- Parallel Aware: Indicates whether the node is capable of being executed in parallel or not.
- Async Capable: Indicates whether the node is capable of asynchronous execution or not.
- Relation Name: Indicates the name of the table or index being scanned.
- Schema: Indicates the schema name of the table or index being scanned.
- Alias: Indicates the alias name of the table or index being scanned.
- Startup Cost: The cost to start up the operation, which includes things like disk seeks or hash table initialization.
- Total Cost: The estimated total cost of running the operation.
- Plan Rows: The estimated number of rows returned by the plan node.
- Plan Width: The estimated average width of each row in bytes.
- Actual Startup Time: The actual time taken to start up the operation, including any initialization costs.
- Actual Total Time: The actual total time taken to execute the operation, including startup costs, processing costs, and any cleanup costs.
- Actual Rows: The actual number of rows returned by the plan node.
- Actual Loops: The number of times the node was executed.
- Output: Indicates the output columns of the node.
- Shared Hit Blocks: The number of shared buffer hits, i.e., how many times the block was found in the shared buffer cache.
- Shared Read Blocks: The number of shared buffer reads, i.e., how many times the block had to be read from disk.

- Shared Dirtied Blocks: The number of shared buffers that were dirtied by the node.
- Shared Written Blocks: The number of shared buffers that were written by the node.
- Local Hit Blocks: The number of local buffer hits, i.e., how many times the block was found in the local buffer cache.
- Local Read Blocks: The number of local buffer reads, i.e., how many times the block had to be read from disk by the local process.
- Local Dirtied Blocks: The number of local buffers that were dirtied by the node.
- Local Written Blocks: The number of local buffers that were written by the node.
- Temp Read Blocks: The number of temporary file blocks read by the node.
- Temp Written Blocks: The number of temporary file blocks written by the node.

3 Web Interface

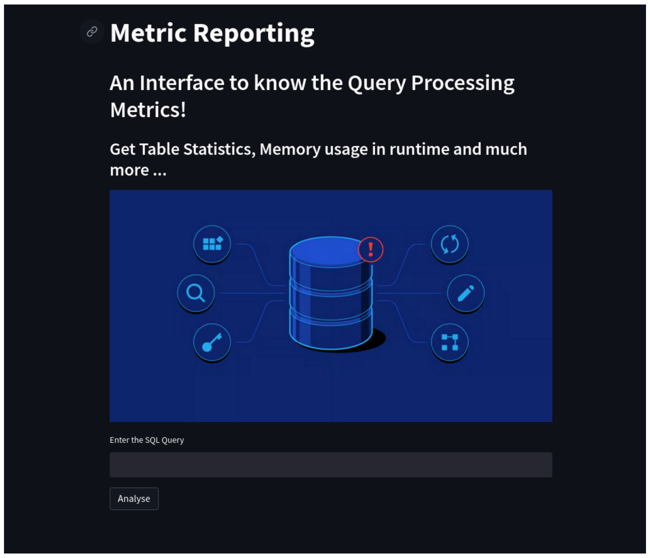
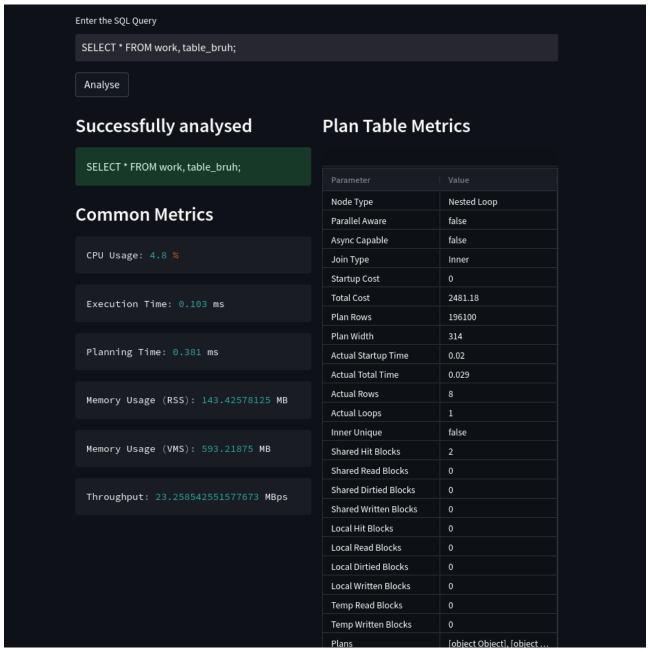

Metric Reporting
An Interface to know the Query Processing Metrics!
Get Table Statistics, Memory usage in runtime and much more ...
Enter the SQL Query
Analyse

Figure 1: This is a screenshot of the interface



SELECT * FROM work, table_bruh;

Successfully analysed

SELECT * FROM work, table_bruh;

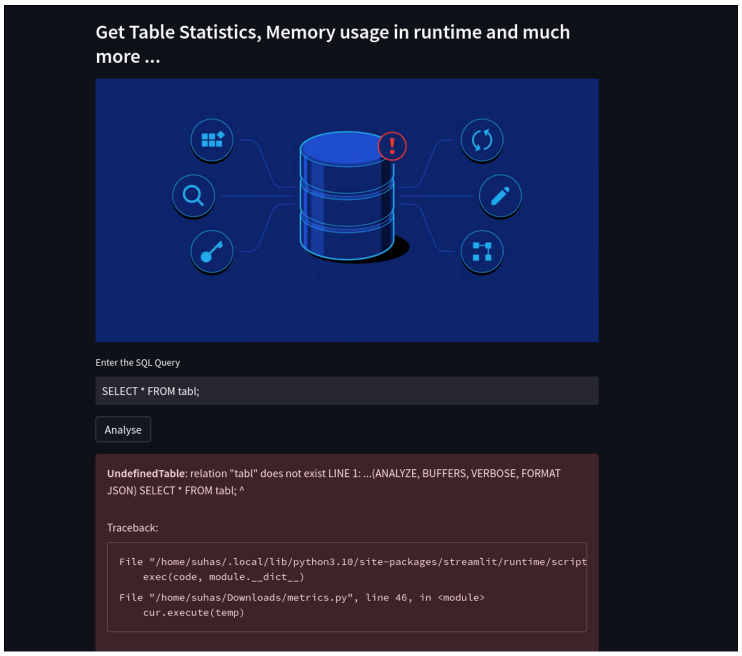
Common Metrics

```
CPU Usage: 4.8 %
Execution Time: 0.103 ms
Planning Time: 0.381 ms
Memory Usage (RSS): 143.42578125 MB
Memory Usage (VMS): 593.21875 MB
Throughput: 23.258542551577673 MBps
```

Plan Table Metrics

Parameter	Value
Node Type	Nested Loop
Parallel Aware	false
Async Capable	false
Join Type	Inner
Startup Cost	0
Total Cost	2481.18
Plan Rows	196100
Plan Width	314
Actual Startup Time	0.02
Actual Total Time	0.029
Actual Rows	8
Actual Loops	1
Inner Unique	false
Shared Hit Blocks	2
Shared Read Blocks	0
Shared Dirtied Blocks	0
Shared Written Blocks	0
Local Hit Blocks	0
Local Read Blocks	0
Local Dirtied Blocks	0
Local Written Blocks	0
Temp Read Blocks	0
Temp Written Blocks	0
Plans	[object Object], [object ...

Figure 2: This is a screenshot of the Metrics being displayed for a query


Get Table Statistics, Memory usage in runtime and much more ...
Enter the SQL Query
Analyse

```
SELECT * FROM tabl;
```

UndefinedTable: relation "tabl" does not exist LINE 1: ...(ANALYZE, BUFFERS, VERBOSE, FORMAT JSON) SELECT * FROM tabl; ^

Traceback:

```
File "/home/suhas/.local/lib/python3.10/site-packages/streamlit/runtime/script
    exec(code, module.__dict__)
File "/home/suhas/Downloads/metrics.py", line 46, in <module>
    cur.execute(temp)
```

Figure 3: Error message for an incorrect query

4 Conclusion

This project demonstrates the use of Streamlit and PostgreSQL to build a web application for collecting and displaying query processing metrics. The application provides valuable insights into the performance of SQL queries, allowing users to optimize and fine-tune their queries for better performance.

5 References

References

[1] Cybertec. (n.d.). How to interpret PostgreSQL EXPLAIN ANALYZE output? CyberTec PostgreSQL. Retrieved April 16, 2023, from https://www.cybertec-postgresql.com/en/how-to-interpret-postgresql-explain-analyze-output/

[2] Hinton, D. (2022, January 27). Re: Is there any issue in adding the primary key to all tables in a PostgreSQL schema? [Online forum post]. Retrieved April 16, 2023, from https://www.postgresql.org/message-id/CAGoODpfvGPJYVvMNiGLnGdWRpPUKq3ncVScpa-v15-266ci_XQ@mail.gmail.com

[3] Matsuda, H. (2021, May 4). Reading an EXPLAIN ANALYZE query plan. thoughtbot. Retrieved April 16, 2023, from https://thoughtbot.com/blog/reading-an-explain-analyze-query-plan

[4] The PostgreSQL Global Development Group. (n.d.). Using EXPLAIN. PostgreSQL 14. Retrieved April 16, 2023, from https://www.postgresql.org/docs/current/using-explain.html