ECOLE POLYTECHNIQUE DE BRUXELLES

**INFOH415 - Advanced Databases**

# Data warehousing with Google BigQuery and Snowflake

Min Zhang 000586970

Yutao Chen 000585954

Ziyong Zhang 000585736

Xianyun Zhuang 000586733

Professor: Zimányi, Esteban

Fall 2023

# Contents

# List of Figures

# 1 Introduction

## 1.1 Overview

In this project, we studied and compared the data warehouse technology in the two products of Google BigQuery and Snowflake, and more accurately, the cloud data warehouse in the two tools. The result comparison is given through a series of tests including the Load Test, Power Test, Throughput Test, and Maintenance Test.

## 1.2 Problem Statement

### 1.2.1 Introduction of Data warehouse

A data warehouse is a particular database targeted toward decision support. It takes data from various operational databases and other data sources and transforms it into new structures that fit better for the task of performing business analysis. Data warehouses are based on a multidimensional model, where data are represented as hypercubes, with dimensions corresponding to the various business perspectives and cube cells containing the measures to be analyzed[7, p.45].

### 1.2.2 Challenges in the Age of Big Data

In the Age of Big Data, the amount of data currently available from a wide variety of sources increasingly poses challenges to scientific and commercial applications. Datasets are continuously growing beyond the limits that traditional database and computing technologies can handle[7, p.562].

Big data greatly impacts on the design of data warehouses, which not only must ingest and store large data volumes coming at high speed, but also deliver query performance under these

conditions and support new kinds of applications. Also, new kinds of data warehouses are emerging, such as cloud data warehouses. Indeed, organizations are increasingly moving their data warehouses to the cloud to take advantage of its flexibility, although many questions remain open in this respect[7, p.563].

## 1.3 Cloud Data Warehouse VS Traditional Data Warehouse

We also investigated some of the key differences between traditional and cloud data warehouses

Traditional data warehouses are hosted on-premises, with data flowing in from relational databases, transactional systems, business applications, and other source systems. However, they are typically designed to capture a subset of data in batches and store it based on rigid schemas, making them unsuitable for spontaneous queries or real-time analysis. Companies also must purchase their hardware and software with an on-premises data warehouse, making it expensive to scale and maintain. In a traditional warehouse, storage is typically limited compared to computing, so data is transformed quickly and then discarded to keep storage space free.

Today's data analytics activities have transformed into the center of all core business activities, including revenue generation, cost containment, improving operations, and enhancing customer experiences. As data evolves and diversifies, organizations need more robust data warehouse solutions and advanced analytic tools for storing, managing, and analyzing large quantities of data across their organizations. As a result, many enterprises are turning to cloud-based data warehouse solutions.

A cloud data warehouse makes no trade-offs from a traditional data warehouse but extends capabilities and runs on a fully managed service in the cloud. Cloud data warehousing offers instant scalability to meet changing business requirements and powerful data processing to support complex analytical queries[1].

There are many popular cloud-based data warehouse platforms to choose from, including Amazon Redshift, Google BigQuery, Microsoft Azure, Snowflake, and others. In this project, we chose Google BigQuery and Snowflake as two technologies for our research.

# 2 Technology Fundamentals

## 2.1 Cloud Data Warehouses Introduction

### 2.1.1 Google BigQuery

Google BigQuery is a fully managed, serverless data warehouse solution offered by Google Cloud. It is designed to process and analyze vast amounts of data in real-time, providing businesses with powerful insights. One of its key features is its ability to scale horizontally, enabling users to handle massive datasets with ease. BigQuery uses a SQL-like syntax for querying data and supports seamless integration with other Google Cloud services. Its serverless architecture eliminates the need for infrastructure management, making it a popular choice for organizations looking for a flexible and cost-effective solution for their data analytics needs[2].

### 2.1.2 Snowflake

Snowflake is a cloud-based data warehousing platform known for its unique architecture that separates storage and compute resources. This separation allows for on-demand and independent scaling of each, providing unmatched flexibility and efficiency. Snowflake supports data processing across multiple clouds, making it a cross-cloud, multi-platform solution. Its features include instant and elastic scalability, automatic and intelligent performance optimization, and a built-in, secure data-sharing capability. Snowflake's data warehouse as a service approach simplifies data management and analytics, empowering organizations to harness the full potential of their data in a secure and collaborative environment[5].

## 2.2 Benchmark Introduction

In this report, the objective is to conduct a comparative analysis between two prominent cloud data warehouses: Google Bigtable and Snowflake. After careful consideration of existing benchmarks, we have selected TPC-H as the benchmark for our research comparison.

The TPC-H is a decision support benchmark. It consists of a suite of business oriented ad-hoc queries and concurrent data modifications[6].

It is a frequently used performance metric for data warehouse systems, focusing on decision support situations, and is intended to replicate complicated ad-hoc searches and concurrent data updates that are prevalent in data warehouse systems. Regarding the properties and features of the TPC-H benchmark in the realm of data warehousing:

### 2.2.1 Scale Factors

The benchmark provides different scale factors. Each scale factor represents a different volume of data, allowing users to test the system's performance under various data sizes. The scale factor is used to determine the size of the database and is defined in terms of gigabytes of data.

### 2.2.2 Queries

TPC-H includes a set of 22 standardized queries (Q1 to Q22) that are complex and representative of decision support queries typically found in data warehousing environments. The queries cover various aspects of reporting, including aggregations, filtering, sorting, and joining multiple tables.

### 2.2.3 Data Loading

TPC-H specifies a procedure for loading data into the data warehouse before running the benchmark. This process ensures a consistent and standardized approach to preparing the database for testing.

### 2.2.4 Performance Metrics

TPC-H reports several performance metrics, including the total execution time for all queries and throughput. Throughput is measured in queries per hour. These metrics provide insights into the efficiency and effectiveness of the data warehouse system in processing complex analytical queries.

# 3 Implementation Process

In this chapter, we are going to explain the process of setting up the two databases and preparation of the data, how the queries were modified, and the execution process of the tests. This implementation consists of the following step.

## 3.1 Databases characteristics

For cloud data warehouses Snowflake and Google BigQuery, we do not need to concern with the specific details of underlying hardware configurations and processors, as these services are provided on cloud platforms rather than running on the local computers.

Therefore, when comparing the two databases, we set up the slots or working units that they are using regarding the costs instead of the details of hardware configurations and processors. Based on the same amount of cost, we then compare the test performances.

The Snowflake version we use is 7.41.0. The environment is set up on the Google Cloud Platform in the Europe West 4 region, using the Enterprise edition. The size for the virtual warehouse in this project is categorized as "X-Small," denoting that the amount of computational resources allocated to it is 1 Unit of Work.

The version of Google BigQuery we are utilizing is 2.0.98, and the environment is set up on the Google Cloud Platform. Based on the pricing information provided by Snowflake [5] and Google [2], we have configured the slots to 100 BQ-slots. Under this configuration, the cost per hour is $4, which is approximately equivalent to that of our Snowflake Enterprise version, which is $3.9 per hour.

## 3.2 Generating and loading data

Data Generation for this project was carried out using *dbgen* of the TPC-H program. The data sets are generated in the local computer using the command regarding different scales. The original data sets are in '.tbl' format with '|' as the separator. To better process the data and load them into the tables, we converted them into '.csv' files using '|' as the separator and removed the row delimiter '|' for each row.

In terms of data uploading, we designed specific python tools for both of the data warehouses, which enabled batch loading for the data and load time recording. For the Snowflake, we created a internal stage for every schema. Then we uses *PUT* command to upload data files into the stages from the local environment. For every schema, we created the tables with the column names, after which we uses the *COPY INTO <table>* command to load the data into the tables from the stages. The loading time was recorded for every schema.

For the BigQuery, firstly we uploaded the data files to the bucket within Google Could Storage. Then we created tables and loaded the data into the tables using the *bq* command in the BigQuery terminal. The loading time was also recorded for further analysis.

## 3.3 Adapting DDL to Data Warehouses

In order to create the corresponding tables and load the data from the data files correctly, adjustments were made to the table creation statements based on BigQuery features. For the Snowflake, no changes according to the Data Definition Language (DDL) should be made.

Two main changes were implemented for loading data into BigQuery:

1. Removal of the Primary Key:

TPC-H provided table creation statements for 8 tables, all of which include primary key constraints. Attempting to execute the original DDL statements as-is results in errors. According to the official Google BigQuery website, it does support the "primary key" clause during table creation.

Consequently, we have made modifications to the DDL statements by removing the "primary key" clause from the table creation statements.

2. Specify some data format as string:

Since the DATE type in BigQuery does not support the '%m/%d/%Y' type nor modifying the format using *FORMAT* command, we input the date variable in a specific format as sting when creating the table, after which we modified the SQL queries accordingly.

## 3.4 Adapting queries

22 queries are included in the TPC-H package. It is required to make some modifications for adapting those queries for the two data warehouses.

For BigQuery, two main modifications were made:

1. Specify the project id and data set name for every table.

2. Convert some specific date format: As mentioned before, some of the date type were loaded as string type. Therefore, we modified the queries accordingly to make sure that the filtering conditions for dates can be established. For example, for the attribute 'l_shipdate' which was input as string type, we used *DATE(PARSE_DATE('%m/%d/%Y', l_shipdate))* to ensure that the format conforms with the condition.

For Snowflake, there are two main functions being modified. The first one is the date adding function, we used the *DATEADD(datepart, number, date)* instead of the original one *date - interval ':1' datepart (number)*. The second modification is the *substring* function. The original query parameters are not applicable to Snowflake, and we adjusted them to become suitable for Snowflake.

## 3.5 Running queries

For running queries and executing tests, we developed Python scripts for BigQuery and Snowflake, respectively. The scripts enable us to access BigQuery and Snowflake platform using Python and execute all queries in a batch process. The individual execution times are recorded and output as csv files for further analysis.

## 3.6 Load test

Load test refers to the process of loading the generated data into the database system. Load time (TLOAD) specifically denotes the time taken to load the generated into the data database. The Load Test consists of two main steps: the data generation step and the data loading step.

First, as mentioned above, we used dsdgen tool to create data in '.tbl' files in loacal environment. After transforming them into the designated csv format, we used a Python script 'load_data.py' for Snowflake and a Shell script 'load_data_time.sh' for BigQuery to load the generated data into the databases while calculating the load time. The logic of both of the scripts is shown as below:

```
1 Load data algorithm:
2 open(load_time_file)
3 set timer start
```

```
load data into database
set timer end
calculate loading_time
write to load_time_file
close(load_time_file)
```

## 3.7 Power test

Power Test indicates the execution time of all statements in a query stream. The power test is conducted after the load test. Power Test Time (TPOWER) measures the time elapsed from the start to the end of a power test. We have developed scripts 'run_scripts.py' for BigQuery and Snowflake, respectively. The scripts executed all queries for 6 times in the specified directory and recorded the execution time of each query in a csv file.

The logic of the scripts is shown as below:

```
Power test algorithm:
open(Exec_time_file)
Extract the name of the query file
for run_number in range(1, 7):
    set timer start
    run the query
    set timer end
    calculate exec_time
    write to Exec_time_file
close(Exec_time_file)
```

## 3.8 Throughput test

Throughput test is executed under a multi-user scenario. The purpose is to measure the shortest time it takes for a system to process the queries when there are multiply users. The throughput test was performed after the power test. The concurrent users number was set as 4, 8 and 16 for both BigQuery and Snowflake. The implementation of the throughput test was also conducted by a python script named 'throughput.py', which could be found in our repository. The algorithm logic is as follows:

```
Throughput test algorithm:
open(Exec_time_file)
```

```
Extract the name of the query file
Create n concurrent users
for run_number in range(1, 7):
    set timer start
    for each concurrent users:
    run all the queries
set timer end
calculate exec_time
write to Exec_time_file
close(Exec_time_file)
```

## 3.9 Maintenance test

In general, the TPC-H test is divided into two parts:

1. Queries: Power Test, Throughput Test. This refers to power testing and throughput testing.
2. Modifications: RF1 and RF2. This refers to insertion and deletion update operations.

The definition of the update functions is as follows: A refresh stream is defined as the sequential execution of an integral number of pairs of refresh functions submitted from within a batch program.

Figure 3.1 represents the entire testing workflow. From the Figure, it is evident that within the test, we need to execute 22 decision support queries and two database refresh functions sequentially for the power test and then in parallel for the throughput test.

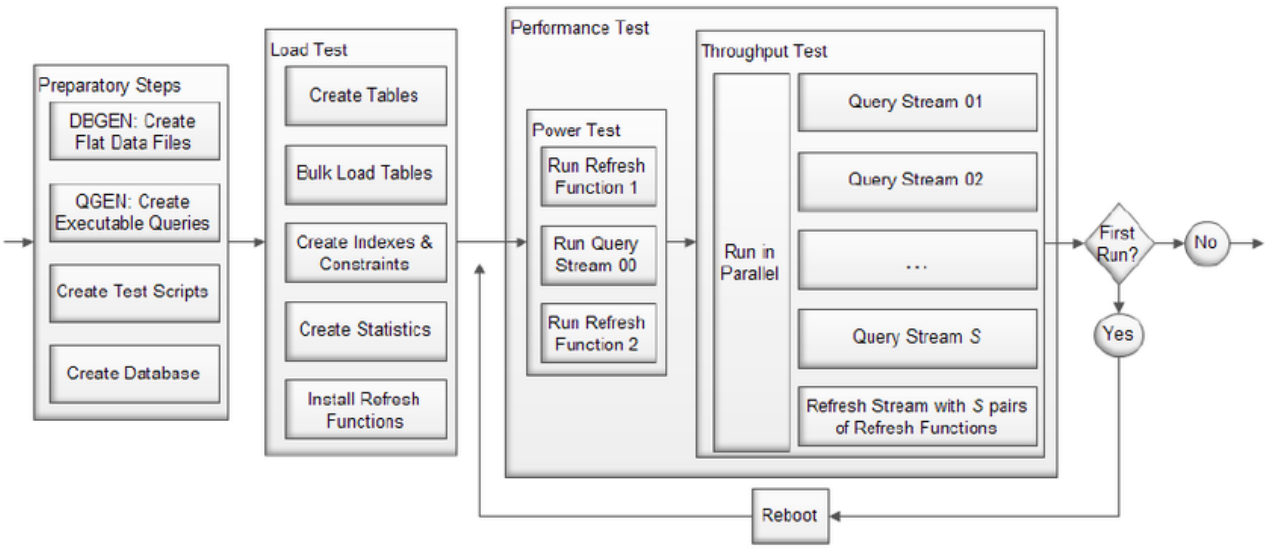


Figure 3.1: Testing Workflow

We've already understood the power test and the throughput test from the previous text. Regarding the refresh functionality, let's take a look at its component definitions:

1. Business principle: It outlines the business context where the refresh functionality can be applied.

2. Refresh function definition: It defines in code the operations the refresh function needs to perform.

3. Refresh data set: This defines the set of rows to insert or delete into the ORDERS and LINEITEM tables each time the refresh functionality is executed. This set of rows represents 0.1% of the initial overall content of these two tables.

### 3.9.1 RF1 (New Sales Refresh Function)

Regarding RF1, we can comprehend it from the following four aspects:

1. Objective

This update function adds new sales information to the database.

2. Business Principle

The new sales update function adds new rows to the ORDERS and LINEITEM tables using a specified scaling factor and the data generation method used when populating the database.

3. Update Function Definition

```
LOOP (SF * 1500) TIMES
INSERT a new row into the ORDERS table
LOOP RANDOM(1, 7) TIMES
INSERT a new row into the LINEITEM table
END LOOP
```

4. Update Data Set

The rows to be inserted must be generated by DBGEN using the -u option, producing the same number of rows required for the multi-stream tests in the Throughput Test.

### 3.9.2 RF2 (Old Sales Refresh Function)

Regarding RF2, we can also comprehend it from the following four aspects: 1. Objective

This update function removes old sales information from the database.

2. Business Principle

The old sales update function deletes rows from the ORDERS and LINEITEM tables in the database to eliminate outdated information.

3. Update Function Definition

```
LOOP (SF * 1500) TIMES
DELETE FROM ORDERS WHERE O_ORDERKEY = [value]
DELETE FROM LINEITEM WHERE L_ORDERKEY = [value]
END LOOP
```

4. Refresh Data Set

DBGen must use the -U option to generate the rows to be inserted. This option will produce enough primary key sets required for the multi-stream tests in the Throughput Test, starting with deletion from the first row of the two target tables.

# 4 Results and Discussions

This section presents a comprehensive analysis of the test results obtained from the previous implementations, including the Load Test, Power Test, Throughput Test, and Maintenance Test. The performance and efficiency of BigQuery and Snowflake are evaluated both independently and in combination under various conditions.

## 4.1 Load Test Results

In this project, we executed the experiment on four different scales: 0.5 GB, 1 GB, 2 GB, and 3 GB. The load time only accounts for the time it takes to load data from the stage into the empty tables in the warehouses, excluding the process of uploading local files to the stages. As shown in Figure 6.9, the load time increases as the data scales enlarged in both Google BigQuery and Snowflake, which is intuitive. By comparison, the load time of Google BigQuery excesses greatly the one of Snowflake in each scale. However, the growth curve with scale of the load time of Google BigQuery is not as steep as that of Snowflake.

In future research, it could be explored whether the load time of BigQuery will be shorter than that of Snowflake in larger data sets.

## 4.2 Power Test Results

In different scales, the execution time of the 22 queries is measured.

The execution time of Google BigQuery is shown as Figure 4.2. Here, some anomalies were observed in the 0.5G scale. Notably, for query 10 and query 13, the execution times at the 0.5GB scale were significantly higher compared to other queries and scales. This phenomenon may be attributed to Google BigQuery's sub optimal performance in handling small data sets.

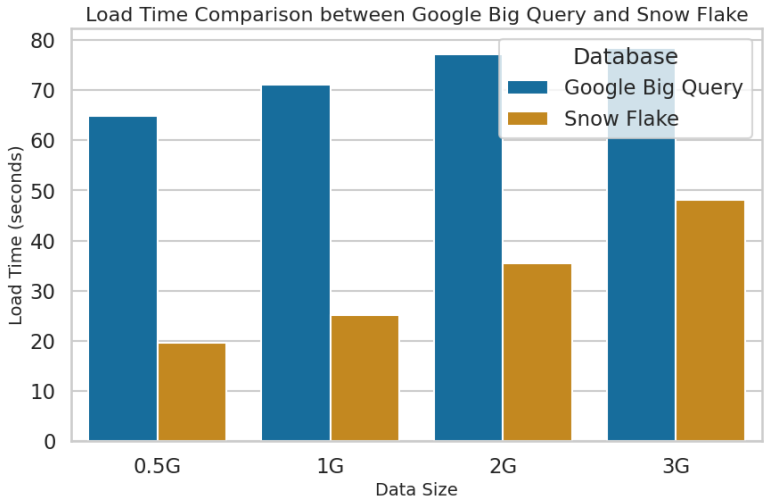


Figure 4.1: Big Query and Snow Flake Load time in 0.5 GB, 1 GB, 2 GB, and 3 GB


Figure 4.2: Big Query Power Test in 0.5 GB, 1 GB, 2 GB, and 3 GB

In the query execution time analysis of 22 queries on Snowflake shown as Figure 4.3, it is observed that the execution time for most queries tends to increase with the growth of the scale. In some instances, such as for query 8 and query 19 at certain scales, there is notable variance in the execution times. However, this variance can be considered within a reasonable range and may be influenced by factors such as the stability of the connection to the server.

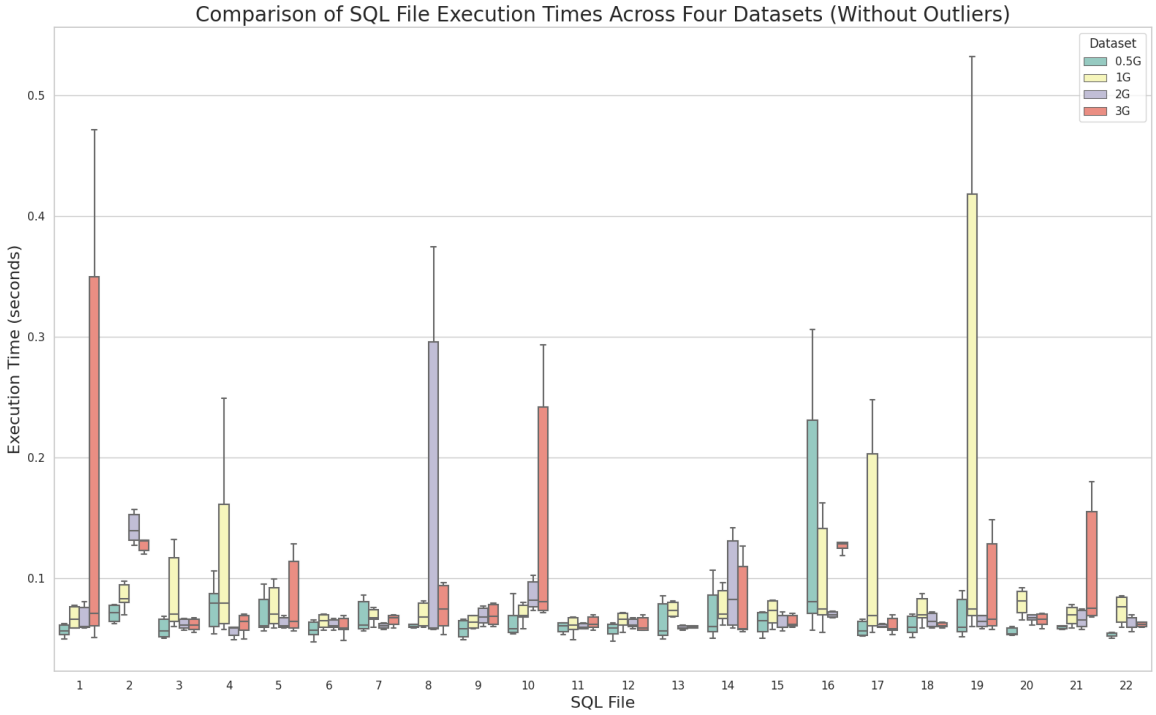


Figure 4.3: Snow Flake Power Test in 0.5 GB, 1 GB, 2 GB, and 3 GB

The average execution time of Google BigQuery for each query is notably longer than that of Snowflake under different scales. Figure 4.4 shows the comparison results under 3GB scale, which is similar in other two scales.

The comparison of the total execution time for Google BigQuery and Snowflake is shown as Figure 4.5. It is observed that at various scales, the total execution time of queries in Google BigQuery is several times longer than that in Snowflake. Notably, for Google BigQuery, there is a decrease in the total execution time at the largest 3G scale. This aligns with the speculation mentioned earlier that Google BigQuery might exhibit better performance with larger data sets.

## 4.3 Throughput results

Delving deeper into our investigation, we now focus on the throughput test results, a crucial metric providing insights into system performance across various user loads. Key to our analysis is the evaluation of the results under different scales.

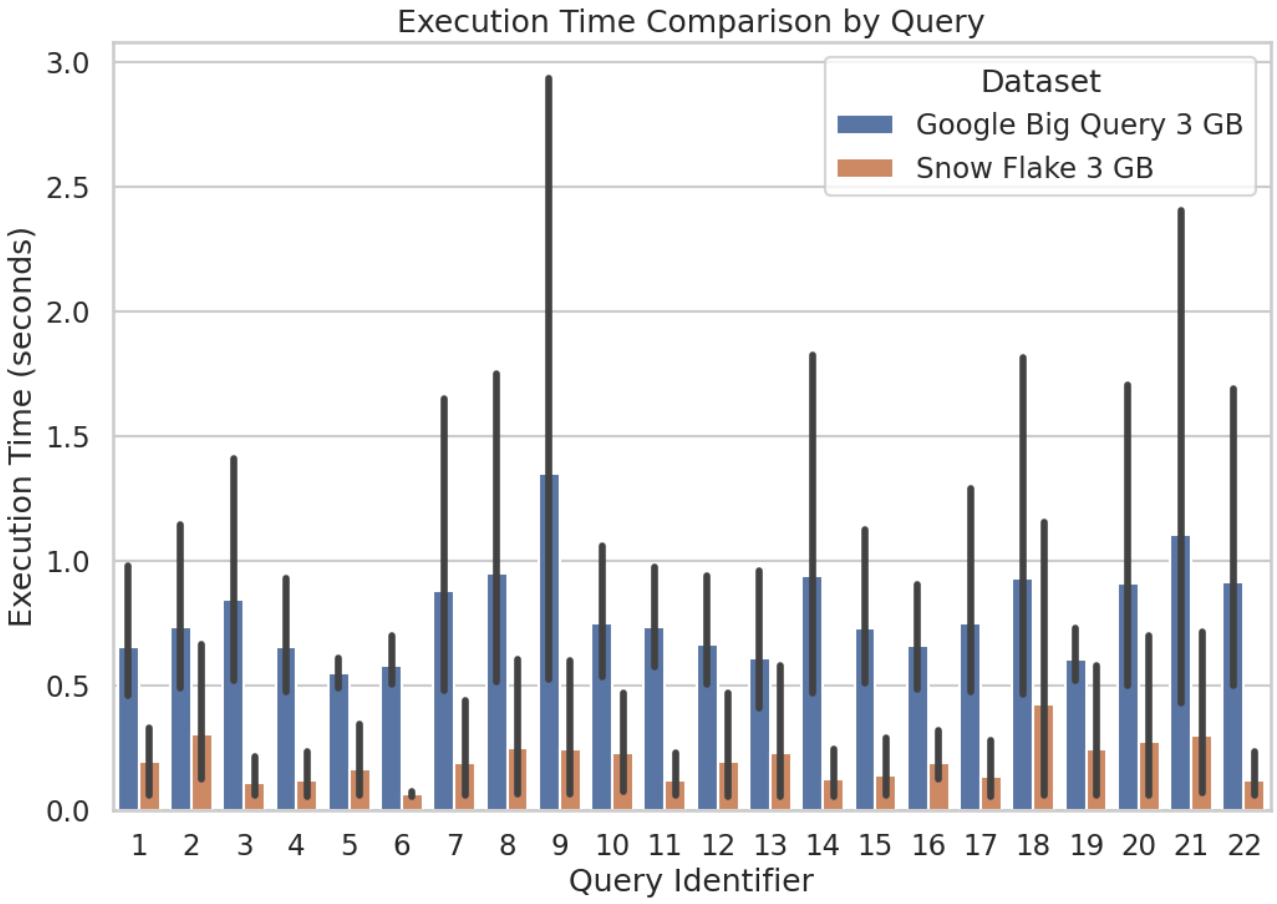


Figure 4.4: Comparison of Big Query and Snow Flake Power Test at 3 GB

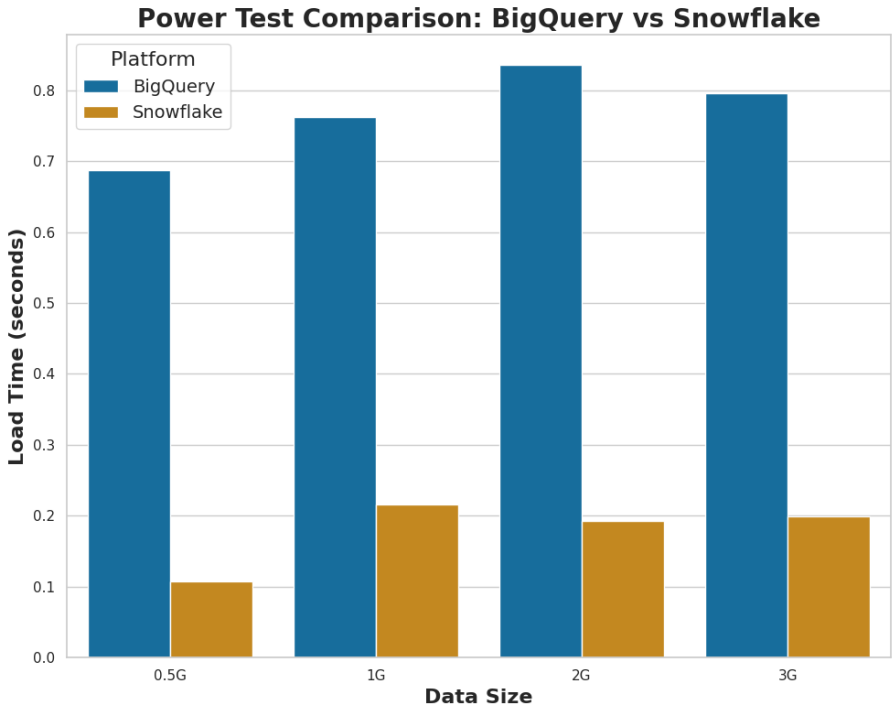


Figure 4.5: Big Query and Snow Flake Power Test Comparison in 0.5 GB, 1 GB, 2 GB, and 3 GB

We set the concurrent user counts to 4, 8, and 16, conducting tests on both Google BigQuery and Snowflake across four scales. Due to the similarity in patterns observed across different scales and concurrent user counts, we analyze results for selected conditions, with additional findings available in Appendix.

Figure 4.6 compares the average execution time per user between the scenario under 16 concurrent users and the one under only 1 users. Overall, there is not a significant difference between the two, indicating that Google BigQuery performs moderately when dealing with multiple concurrent users.

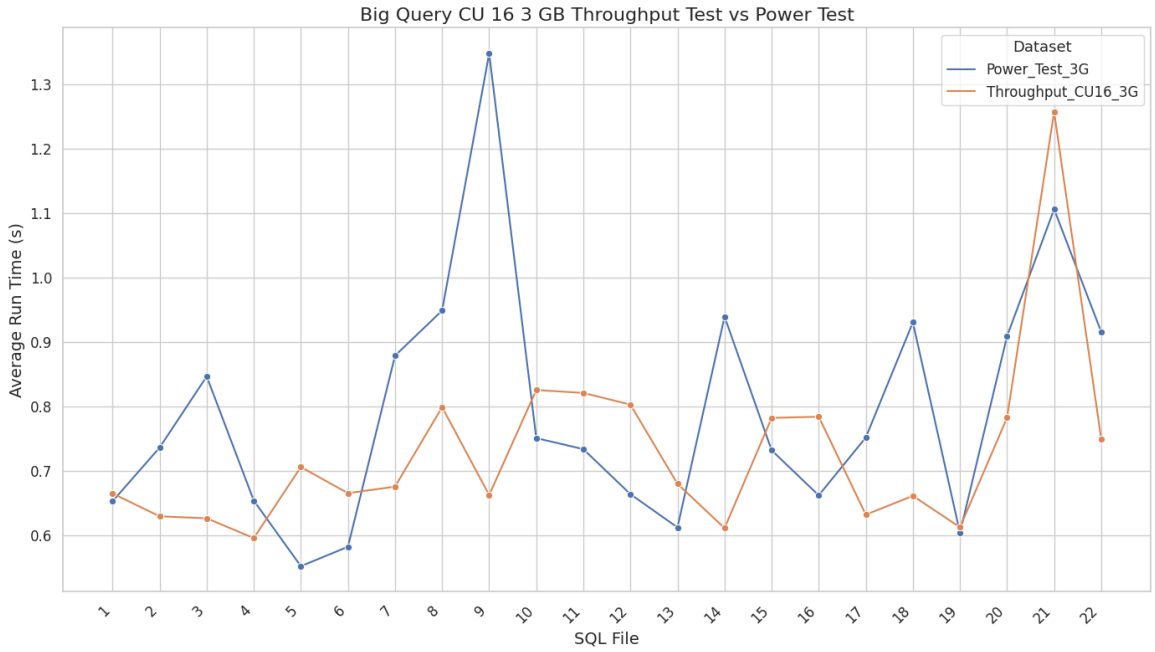


Figure 4.6: Big Query Throughput Test under 16 Concurrent Users within 3G

For Snowflake, the average execution time per user under 16 concurrent users and under 1 user is shown as Figure 4.7. The average execution time per user is significantly less than the latter one, indicating that Snowflake is better dealing with multiply concurrent users.

We could also compare the average execution time per user of Google BigQuery and Snowflake under different concurrent users. The patterns under various scenarios are similar, therefore, here we take the comparison under 3G with 16 concurrent users as an example. Shown as Figure 4.8, it is easily observed that the average execution time per user of BigQuery is several times that of Snowflake for every query.

To delve deeper into this phenomenon, we conducted further research. A previous study on TPC-DS, sponsored by Microsoft and conducted by GIGAOM, revealed instances where the execution time of Google BigQuery was ten times longer than that of Snowflake [3]. Another study on TPC-H indicated that the performance of both data warehouses is comparable under 5 concurrent

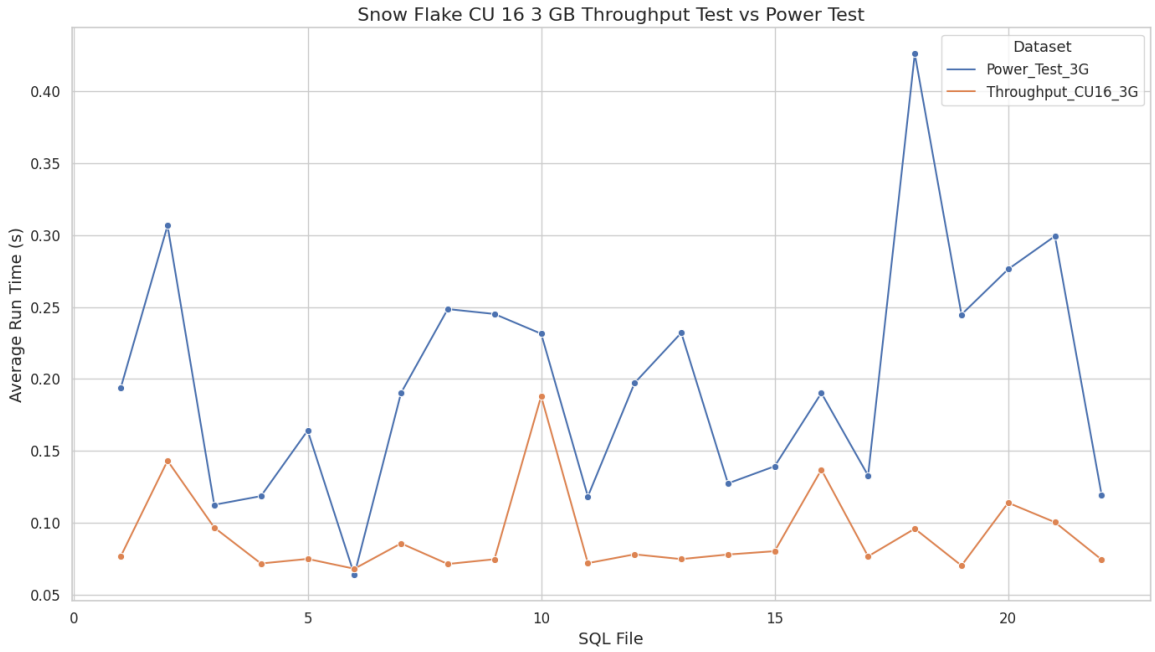


Figure 4.7: Snow Flake Throughput Test under 16 Concurrent Users within 3G

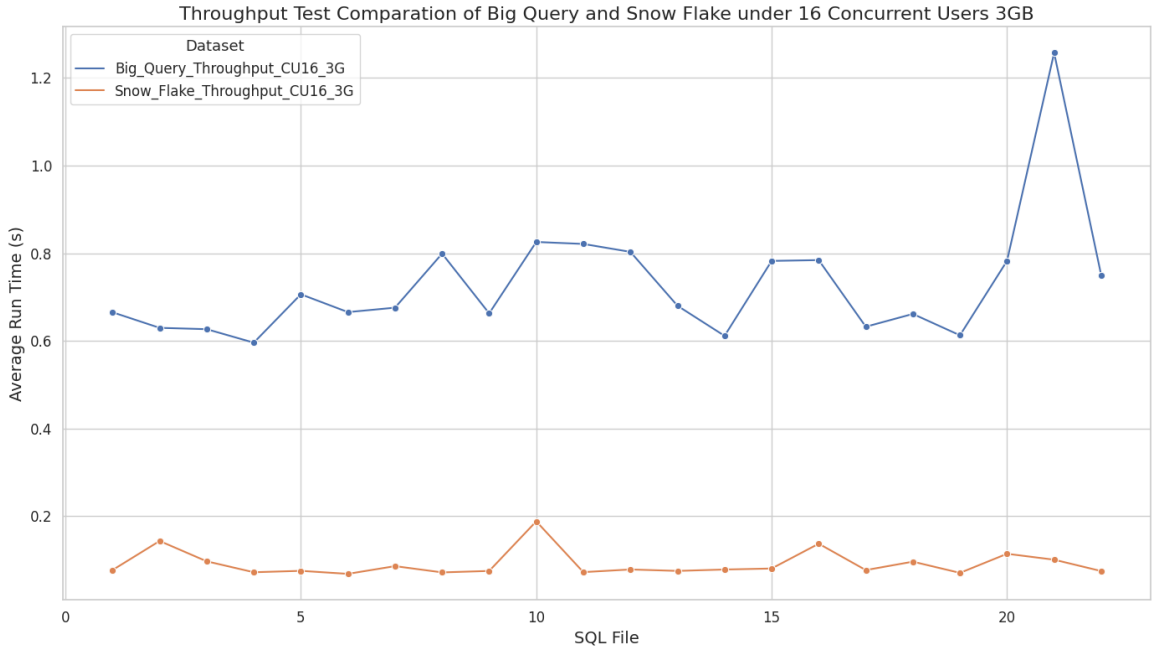


Figure 4.8: Comparison on Throughput Test under 16 Concurrent Users within 3G

users, but the cost of Google BigQuery is twice that of Snowflake [4]. Consequently, the finding of our project is aligned with these previous conclusions.

## 4.4 Maintenance results

### 4.4.1 Power test results

#### RF1 results

The execution time for RF1 increases proportionally with the database size in both BigQuery and Snowflake, as shown in the FigureFigure 4.9.

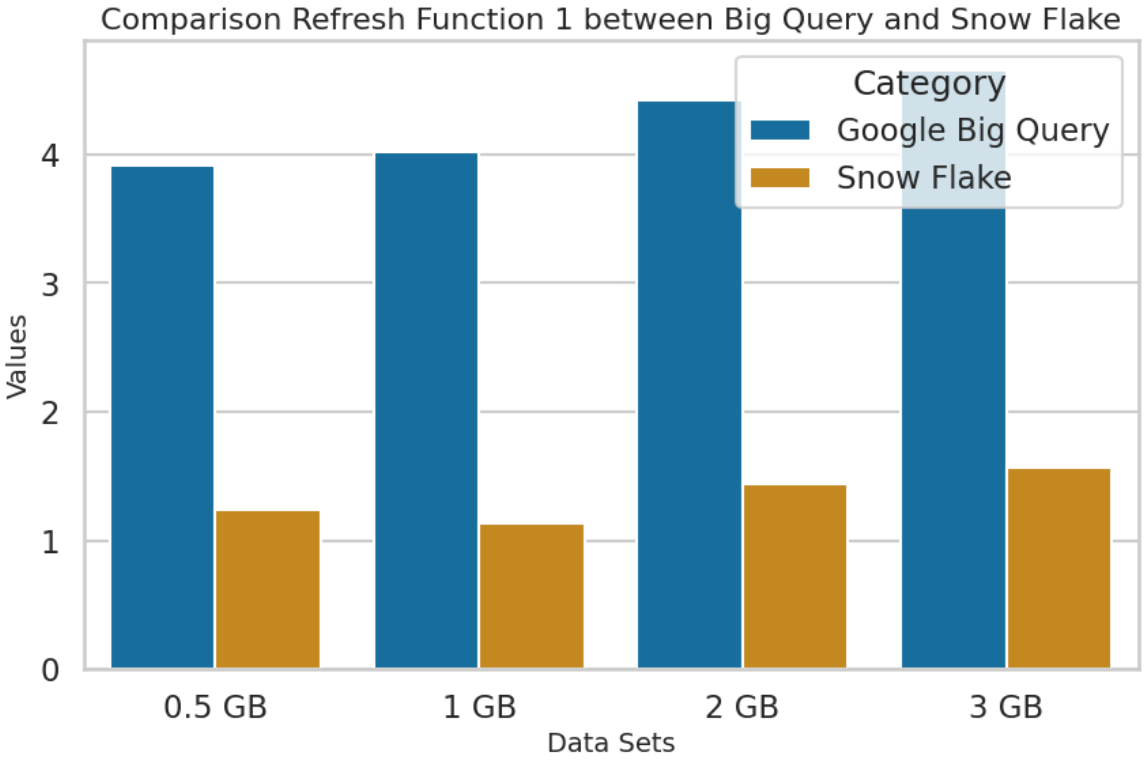


Figure 4.9: Comparison of RF1 in Big Query and Snow Flake

#### Query Stream results

FigureFigure 4.10, FigureFigure 4.11, FigureFigure 4.12, and FigureFigure 4.13 respectively display the execution time of the query stream on 0.5GB, 1GB, 2GB, and 3GB for Google BigQuery.

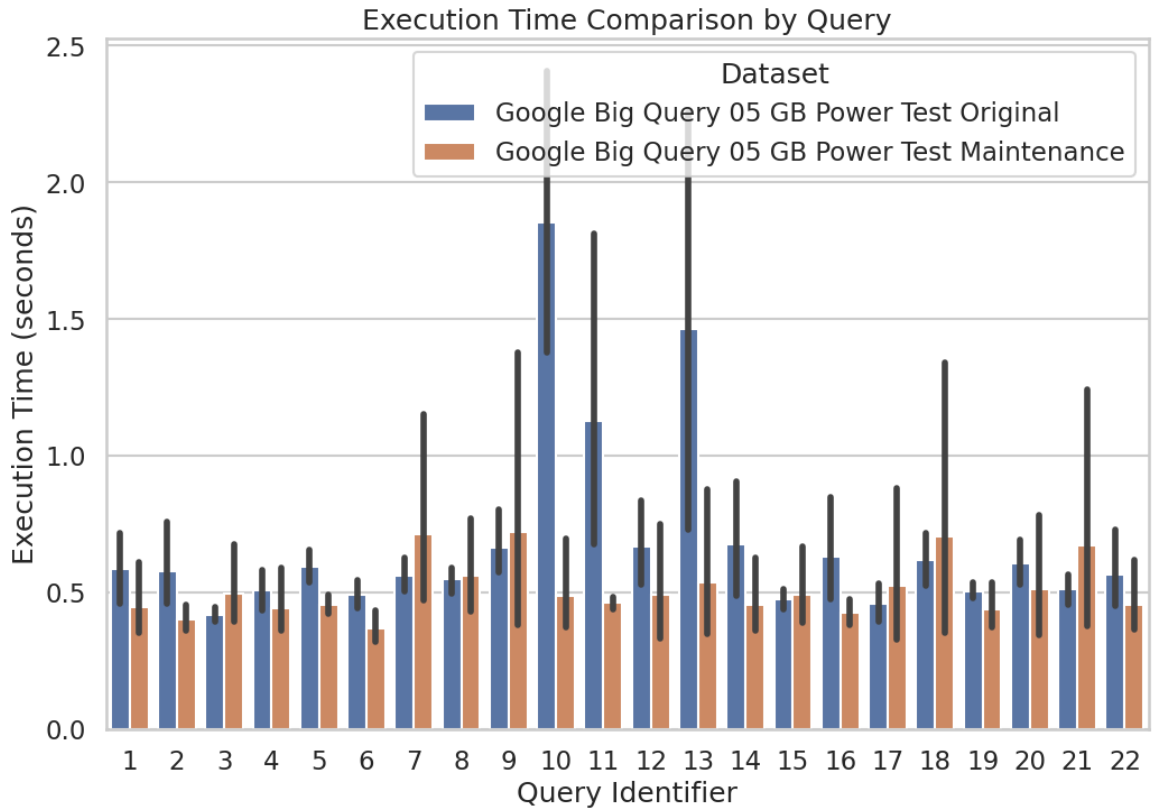


Figure 4.10: Comparison Power Test in 0.5 GB

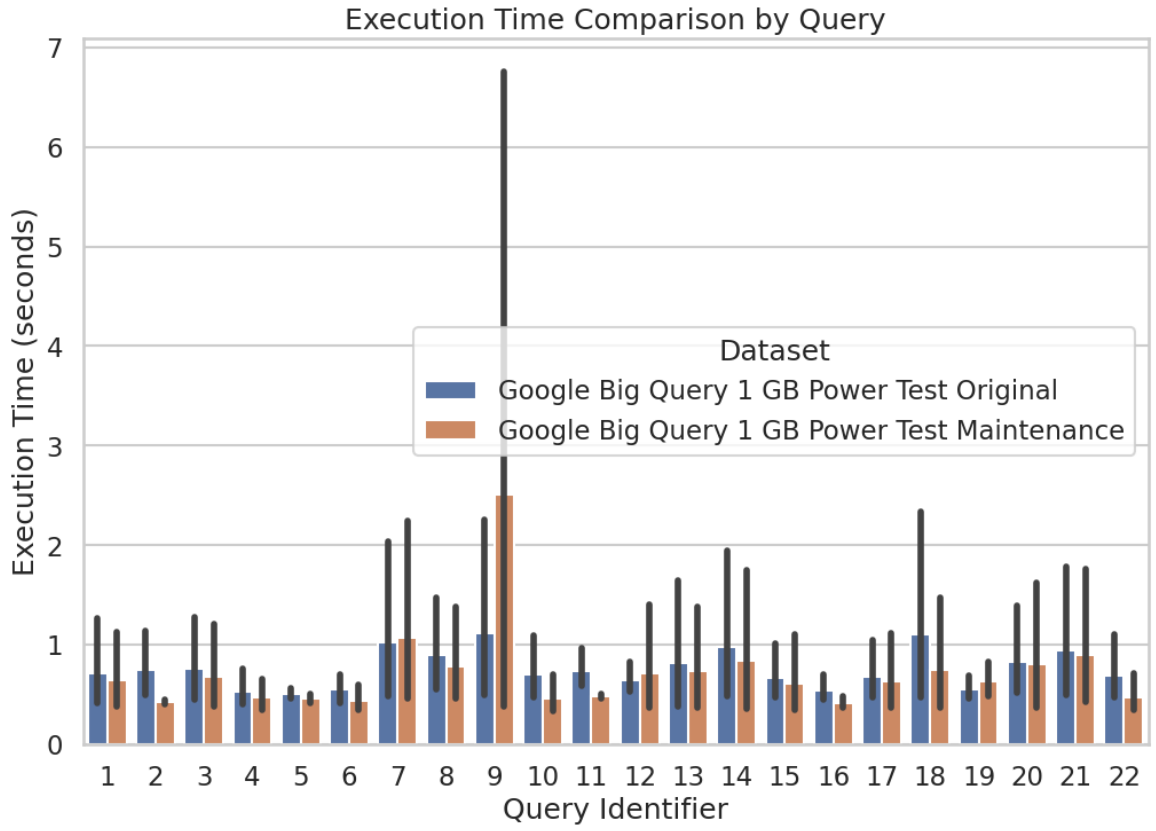


Figure 4.11: Comparison Power Test in 1 GB

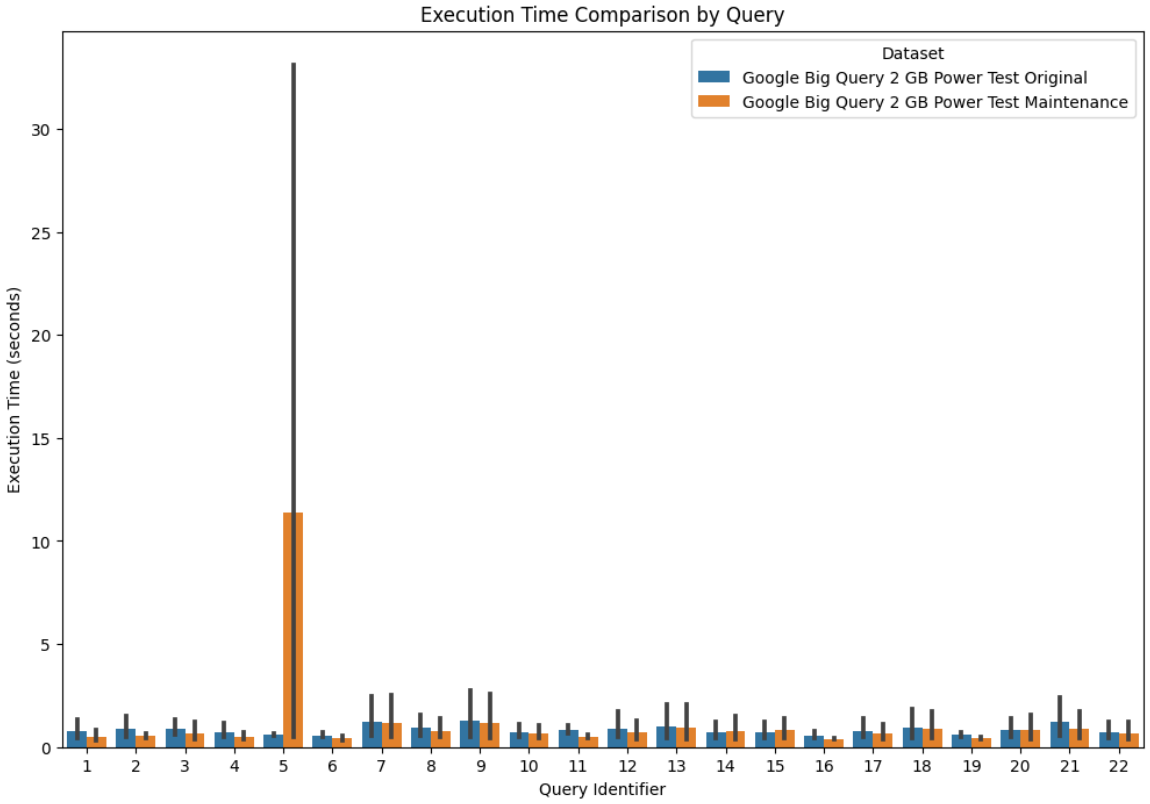


Figure 4.12: Comparison Power Test in 2 GB

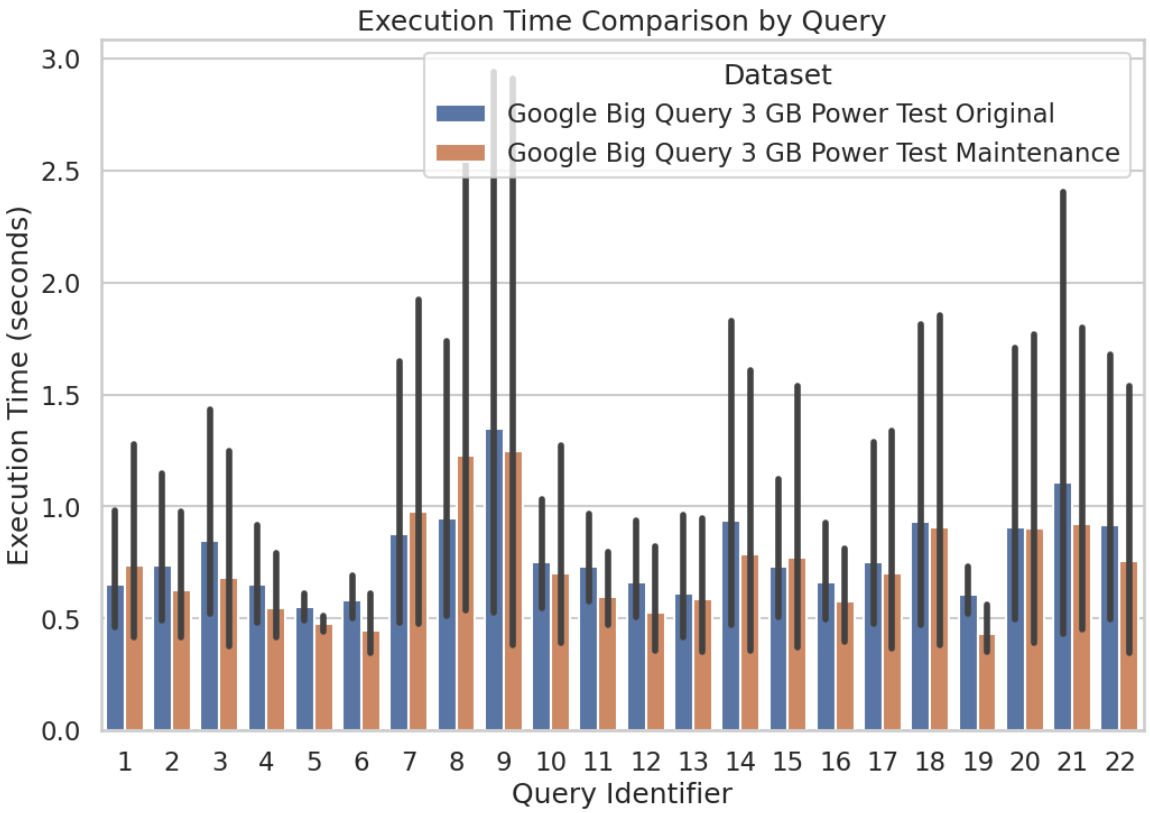


Figure 4.13: Comparison Power Test in 3 GB

We can clearly observe that in the 2GB database, query 5 exhibits an unusually large execution

time. As the execution time is the average time after running the query 6 times, after removing the outlier, we obtained FigureFigure 4.14.

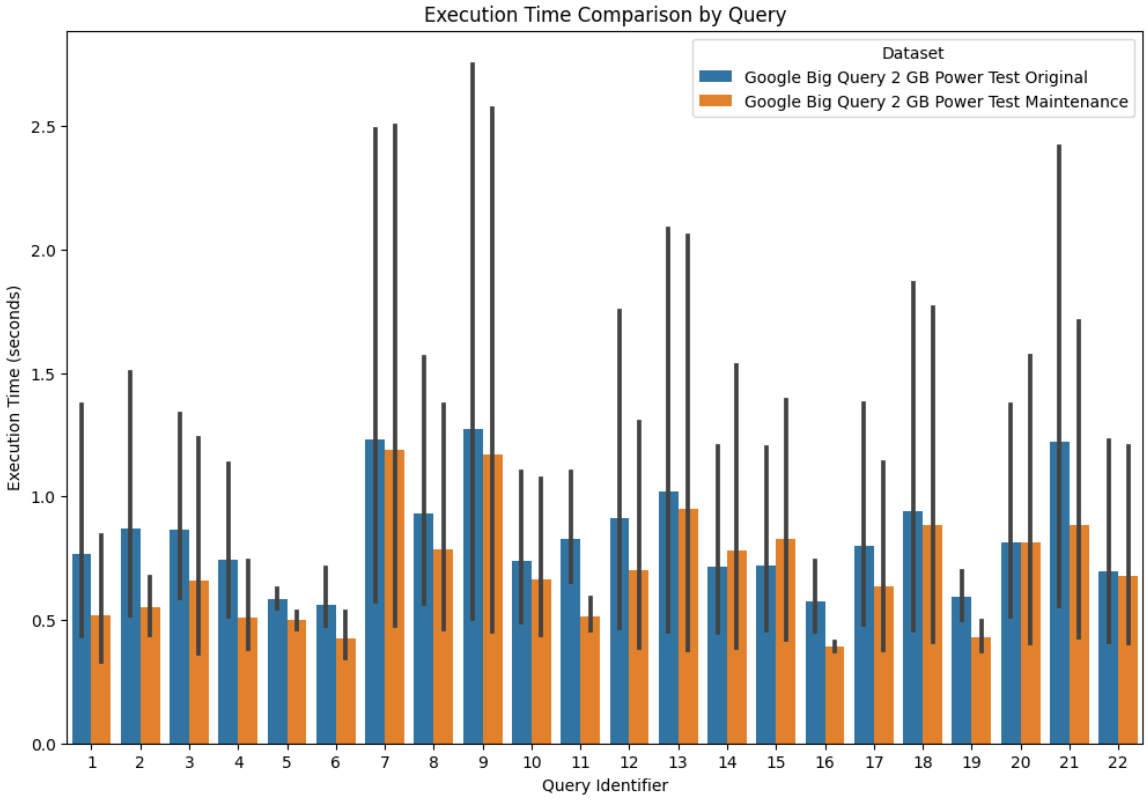


Figure 4.14: Comparison Power Test in 2 GB deleting outlier

Let's observe Figure 4.10, Figure 4.11, Figure 4.14, and Figure 4.13. After executing the RF1 function, which involves inserting some data, the runtime of certain queries tends to decrease. For instance, across all scale factor datasets, the runtime of query 10 decreased after inserting data. The reason some queries have shorter execution times after data insertion might be due to the relatively small size of the inserted data, which may not significantly impact the execution time compared to the original data. We will verify this hypothesis in the future after inserting large volumes of data.

Figure 4.15, Figure 4.16, Figure 4.17, and Figure 4.18 respectively display the execution time of the query stream on 0.5GB, 1GB, 2GB and 3GB for Snow Flake. We can observe a similar pattern in datasets of 0.5GB, 1GB,2GB, and 3GB. Which means after executing the RF1 function, which involves inserting some data, the runtime of certain queries tends to decrease.

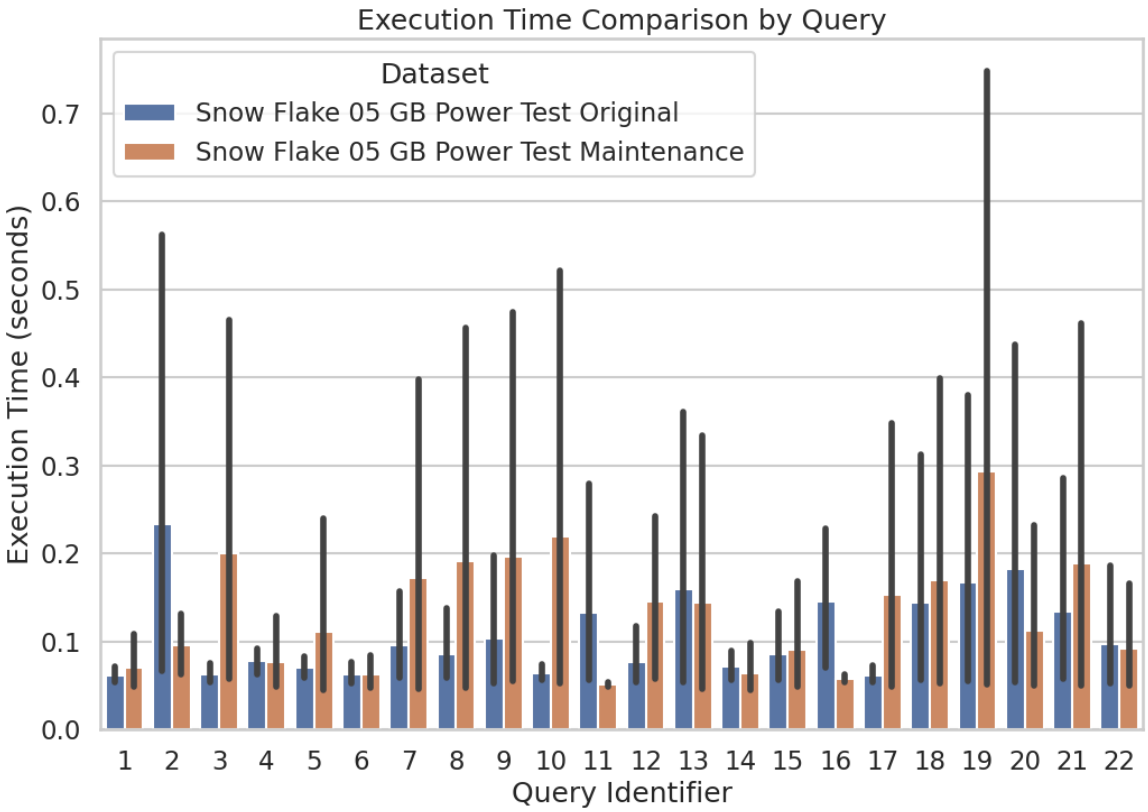


Figure 4.15: Comparison Power Test in 0.5 GB

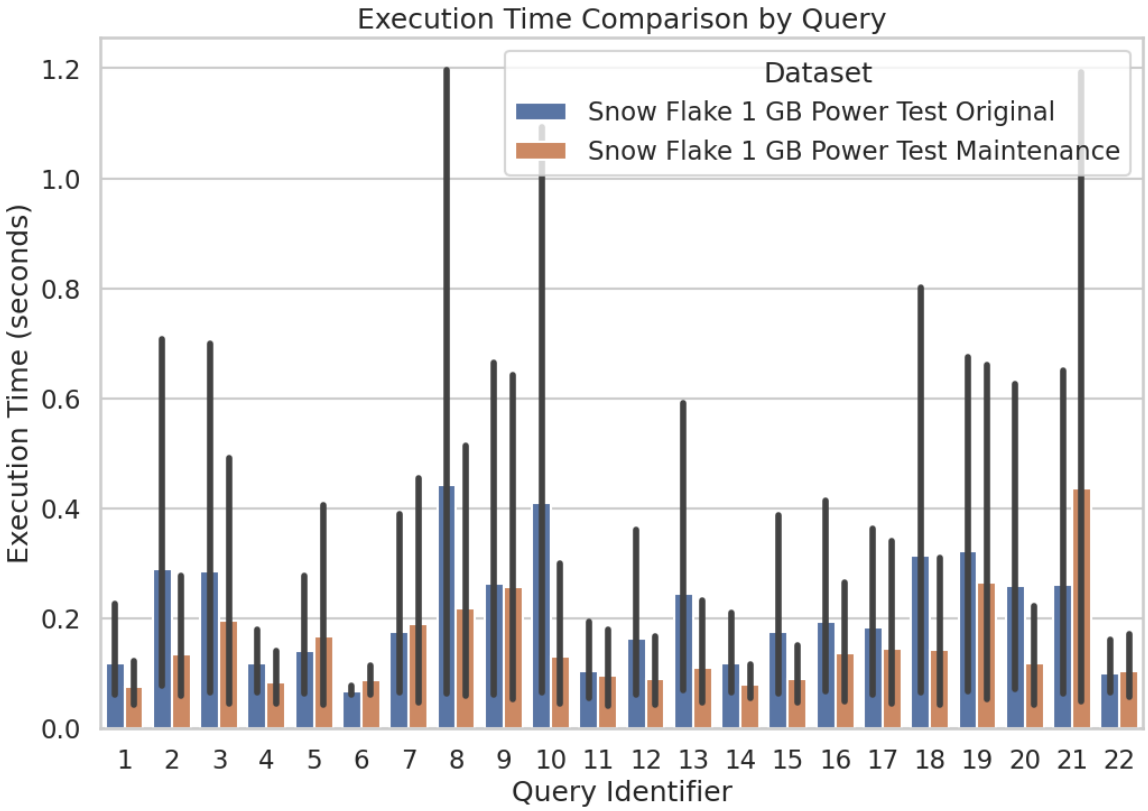


Figure 4.16: Comparison Power Test in 1 GB

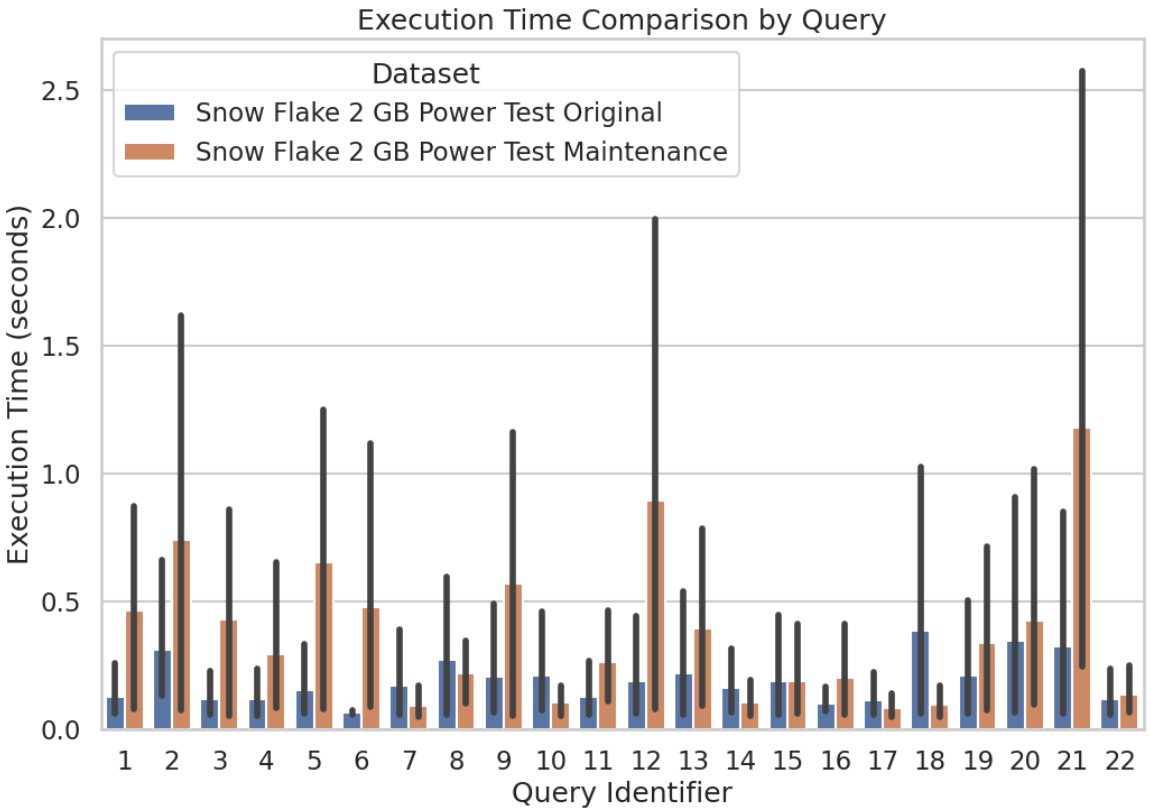


Figure 4.17: Comparison Power Test in 2 GB

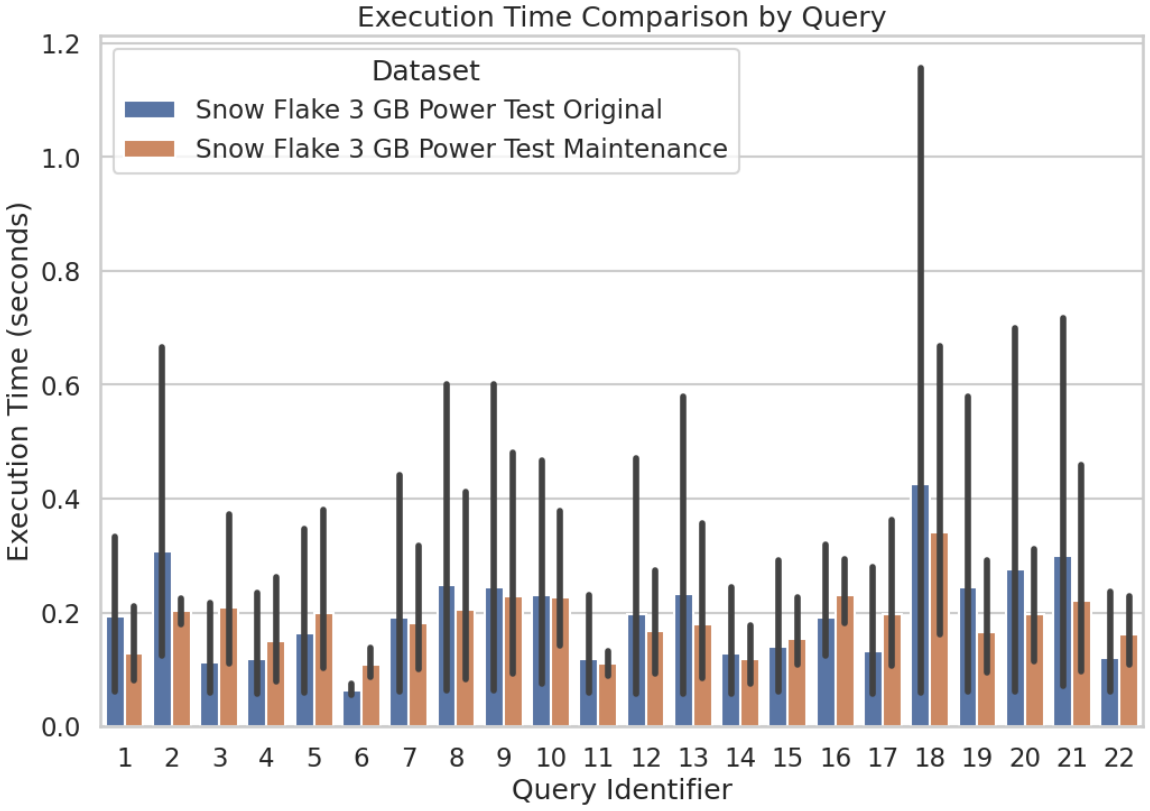


Figure 4.18: Comparison Power Test in 3 GB

**RF2 results**

The execution time for the database size in both BigQuery and Snowflake, as shown in the Figure 4.19.

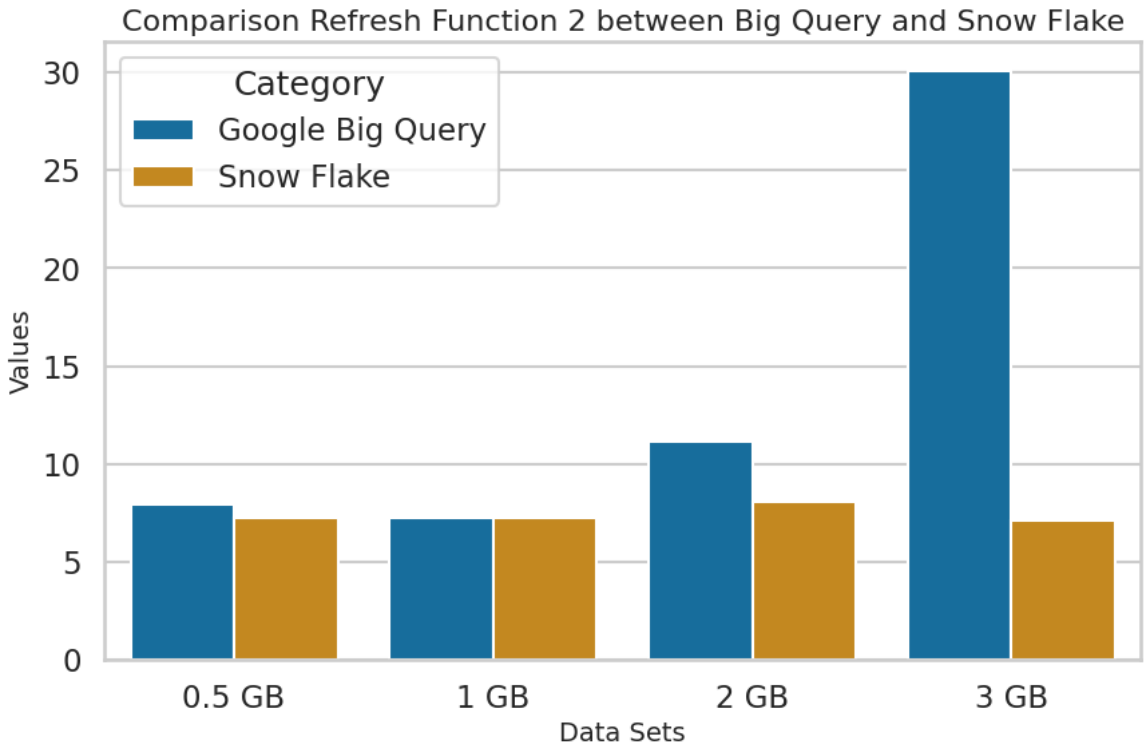


Figure 4.19: Comparison of RF2 in Big Query and Snow Flake

In Google BigQuery, executing DELETE queries in databases of sizes 0.5GB, 1GB, 2GB, and 3GB typically doesn't proportionally increase the time taken with database size growth. The execution time for executing DELETE queries on the 0.5GB dataset is longer than on the 1GB dataset.

This is because BigQuery's architecture and execution methodology differ significantly from traditional databases. DELETE operations don't usually directly remove rows. Instead, they mark rows as deleted and undergo optimization and compression in the background to reclaim space. This method of soft deletion means that the relationship between DELETE operation performance and database size isn't linear.

Moreover, BigQuery's execution process may influence the performance of DELETE operations. Larger databases might leverage more parallel processing resources, resulting in reduced execution time. BigQuery optimizes execution plans utilizing its robust distributed computing and storage capabilities, potentially making operations on larger databases complete faster than on smaller ones.

Overall, the execution methodology and optimization strategies in BigQuery can cause DELETE operation time not to follow a linear relationship with database size, potentially resulting in shorter

processing times on larger databases.

In Snowflake, the execution time of delete queries may not necessarily increase linearly with the database size due to its optimized architecture and parallel processing capabilities. Factors contributing to this could be:

1. Data Storage and Compression Optimization: Snowflake uses columnar storage and efficient compression techniques. In larger databases, there might be more data redundancy and better compression ratios, leading to faster delete operations.

2. Distributed Architecture and Parallel Processing: Snowflake employs a distributed architecture and parallel processing, enabling efficient parallel execution of operations on massive datasets. With larger databases, Snowflake can effectively utilize compute resources and parallel processing, potentially speeding up operations.

3. Data Partitioning and Optimization Techniques: Snowflake allows data partitioning and indexing, enabling faster access and manipulation of specific data partitions in larger databases, and reducing delete operation times.

In summary, Snowflake's optimized strategies and distributed architecture might not result in a linear increase in execution time for delete operations across databases of different sizes. In some cases, larger databases may even execute delete operations faster due to data structures, optimization strategies, and underlying architecture.

We can observe significantly different execution times between INSERT and DELETE operations. This scenario might involve the internal workings and optimization methods of Google BigQuery. BigQuery is a distributed, managed data warehouse service, and its execution of queries and operations might lead to this behavior.

When executing a DELETE command, BigQuery might employ different optimization strategies, such as using varying deletion methods (like partitioned deletion or index optimization). For smaller databases, simpler and faster deletion strategies could be utilized, while larger databases might employ more complex yet efficient deletion algorithms to expedite the operation.

On the other hand, during INSERT command execution, typically, as the database size increases, the time taken for insertion might also increase. This is due to the increased volume of data, necessitating more resources for data writing, constraint checking, maintaining indexes, and other operations, leading to increased insertion time.

Overall, BigQuery's optimization strategies may vary based on data volume and operation types. In certain cases, operations on larger datasets might be quicker compared to smaller ones, potentially due to BigQuery using more efficient processing methods and optimization strategies for larger datasets.

In Snowflake, the performance differences between DELETE and INSERT operations might stem from the database engine's workings and data storage approach.

1. DELETE Operation: Snowflake utilizes a log-structured storage approach. When executing DELETE, it doesn't immediately remove data but rather marks the rows to be deleted in the background, termed "soft deletion." Thus, the time taken for DELETE operations might not significantly increase across databases of different sizes since it involves changes only to metadata. For larger databases, the duration might be slightly longer but not linearly increased, as soft deletion involves merely recording deleted metadata.

2. INSERT Operation: When performing INSERT in Snowflake, data is written into the underlying storage layer, possibly involving data sharing and distributed storage operations. With an increase in database size, data insertion might necessitate more disk space and distributed computing resources, leading to an escalation in insertion time.

The stable performance of DELETE operations is because they don't physically remove data but handle metadata in the background. In contrast, INSERT operations involve actual data writing and distributed storage, and hence, the required resources and time might escalate with an increase in database size.

### 4.4.2 Throughput results

#### Refresh Stream results

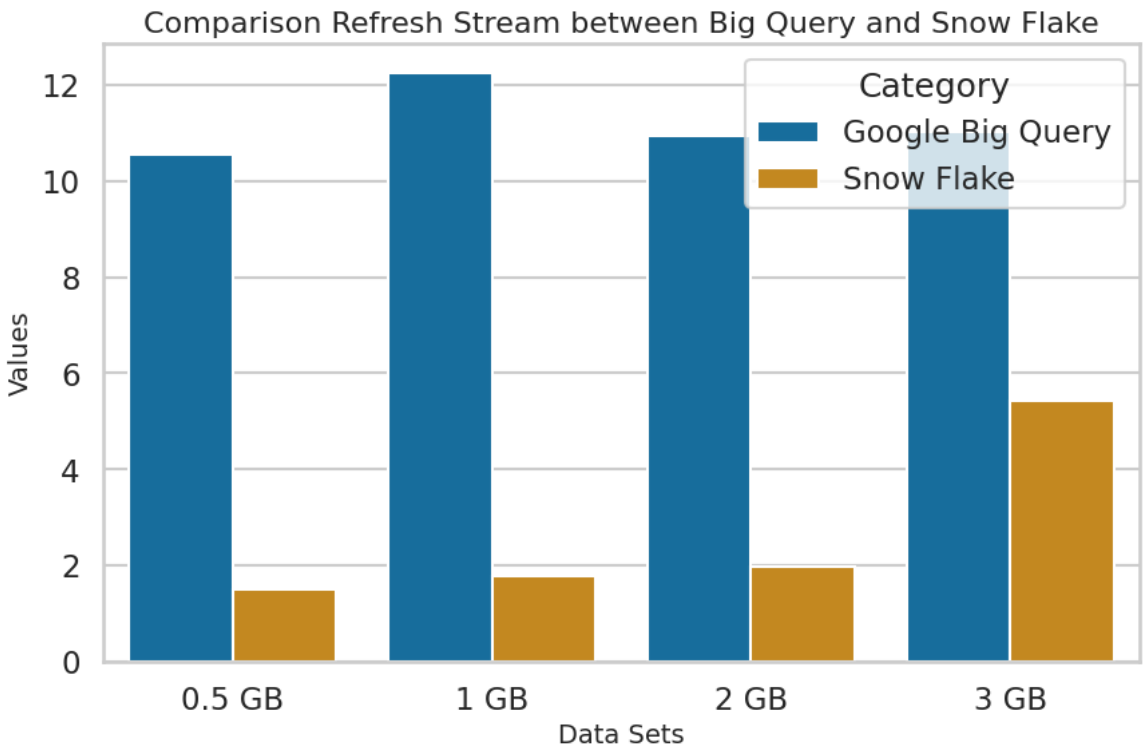


Figure 4.20: Comparison of RF Stream in Big Query and Snow Flake

The execution results of the refresh stream are shown in the Figure 4.20. It's evident that the execution time of the refresh stream exhibits a linear trend across 0.5GB, 1GB, 2GB, and 3GB datasets on Snowflake, whereas there is no clear trend on BigQuery.

#### Query Stream results

Figure 4.21, Figure 4.22, Figure 4.23, Figure 4.24 respectively display the execution time of the query stream on 0.5GB, 1GB, 2GB, and 3GB for Google BigQuery.

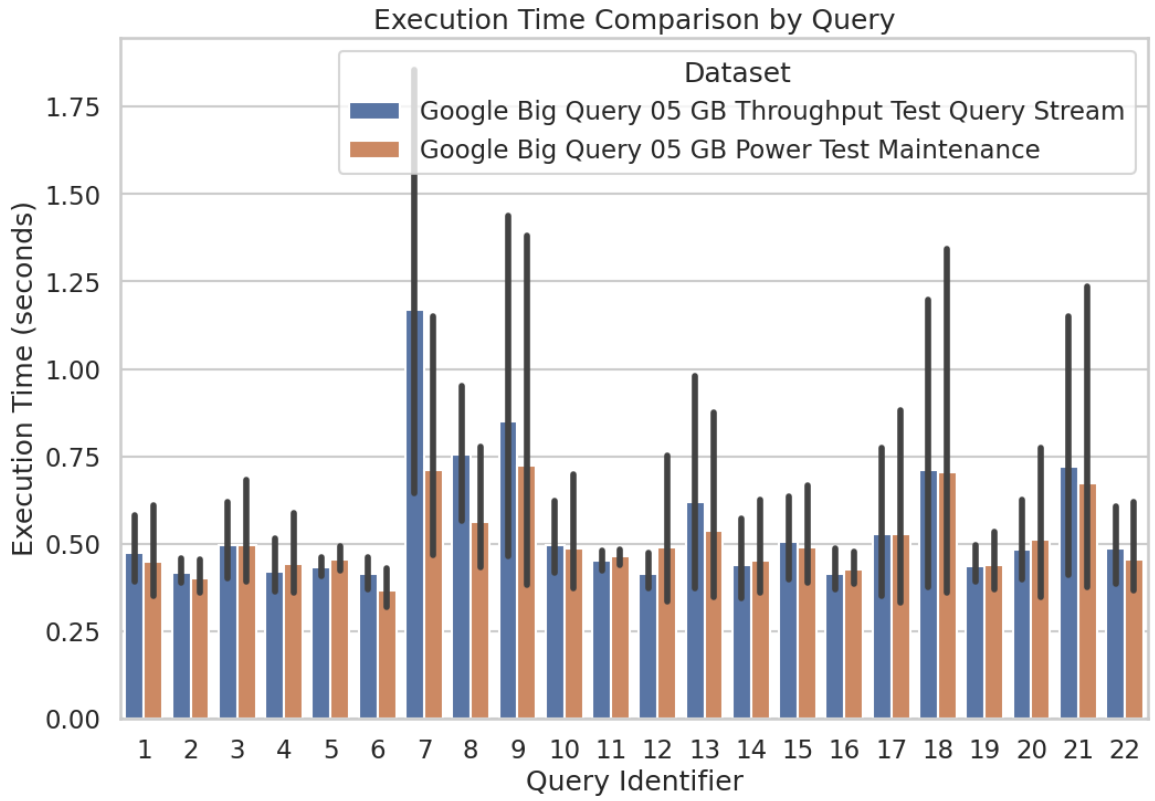


Figure 4.21: Comparison Throughput Test in 0.5 GB

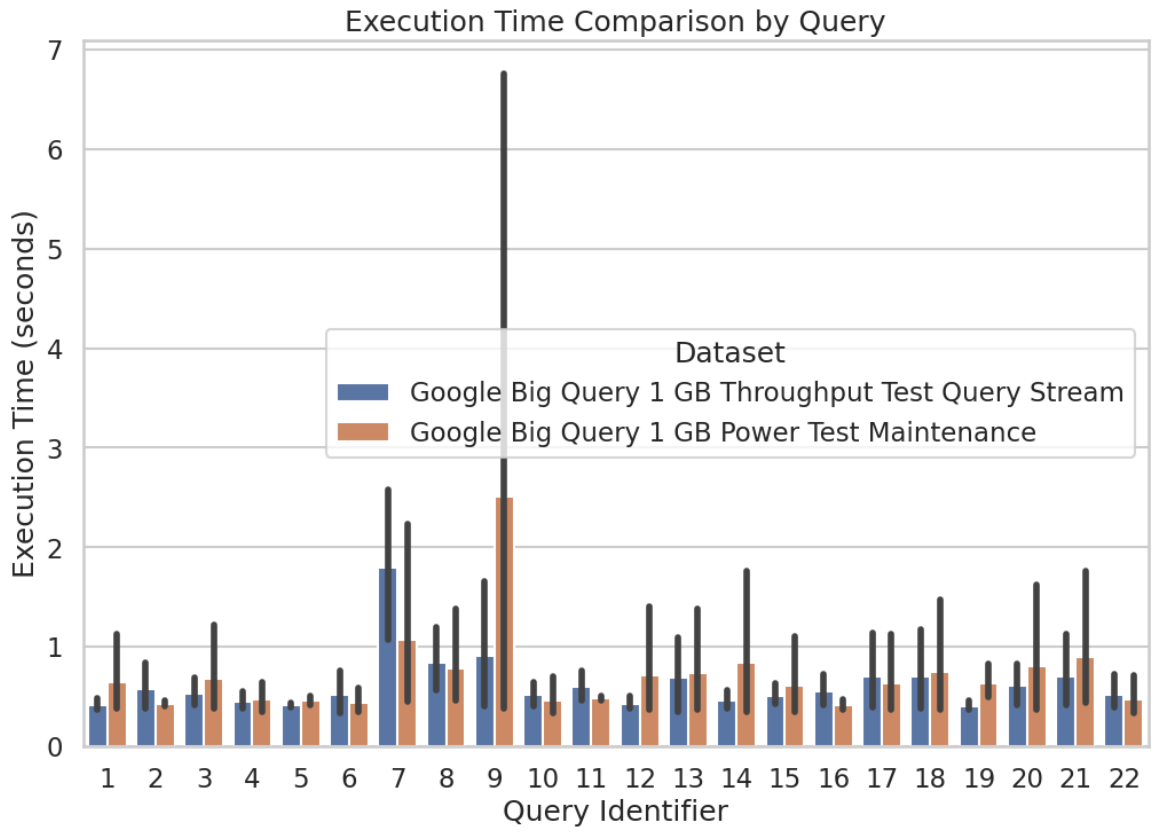


Figure 4.22: Comparison Throughput Test in 1 GB

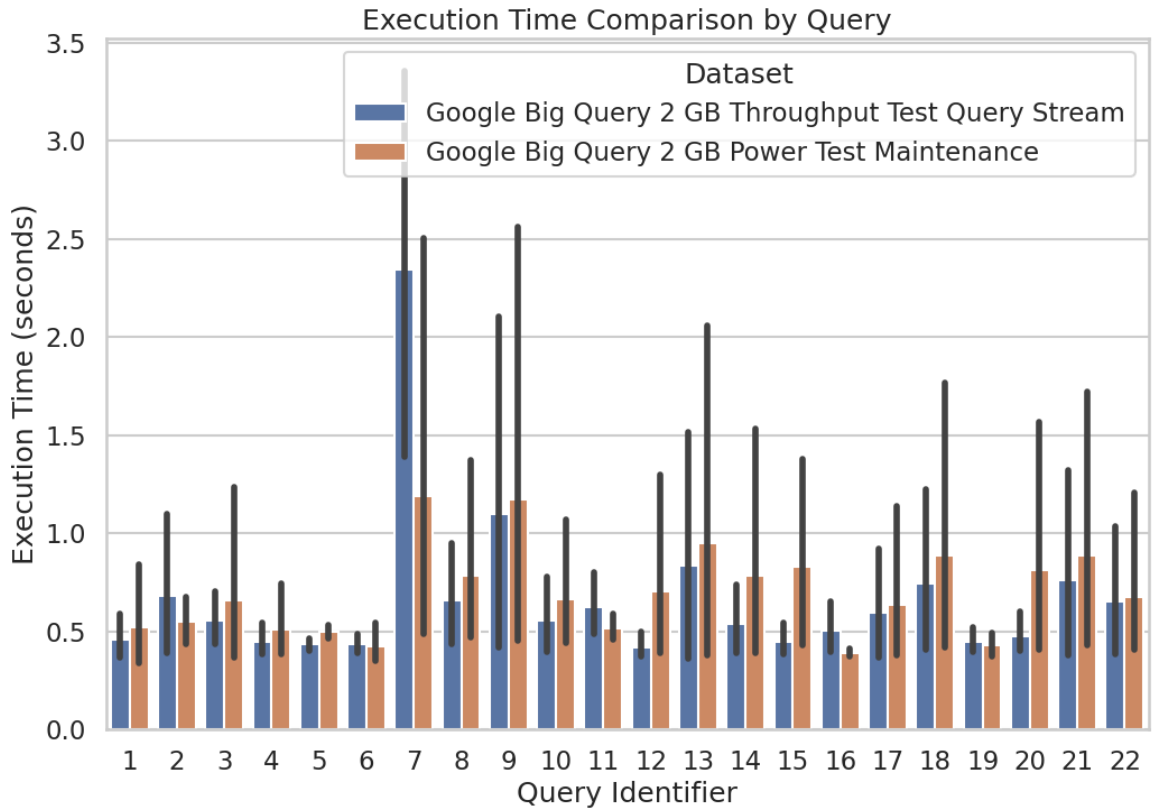


Figure 4.23: Comparison Throughput Test in 2 GB

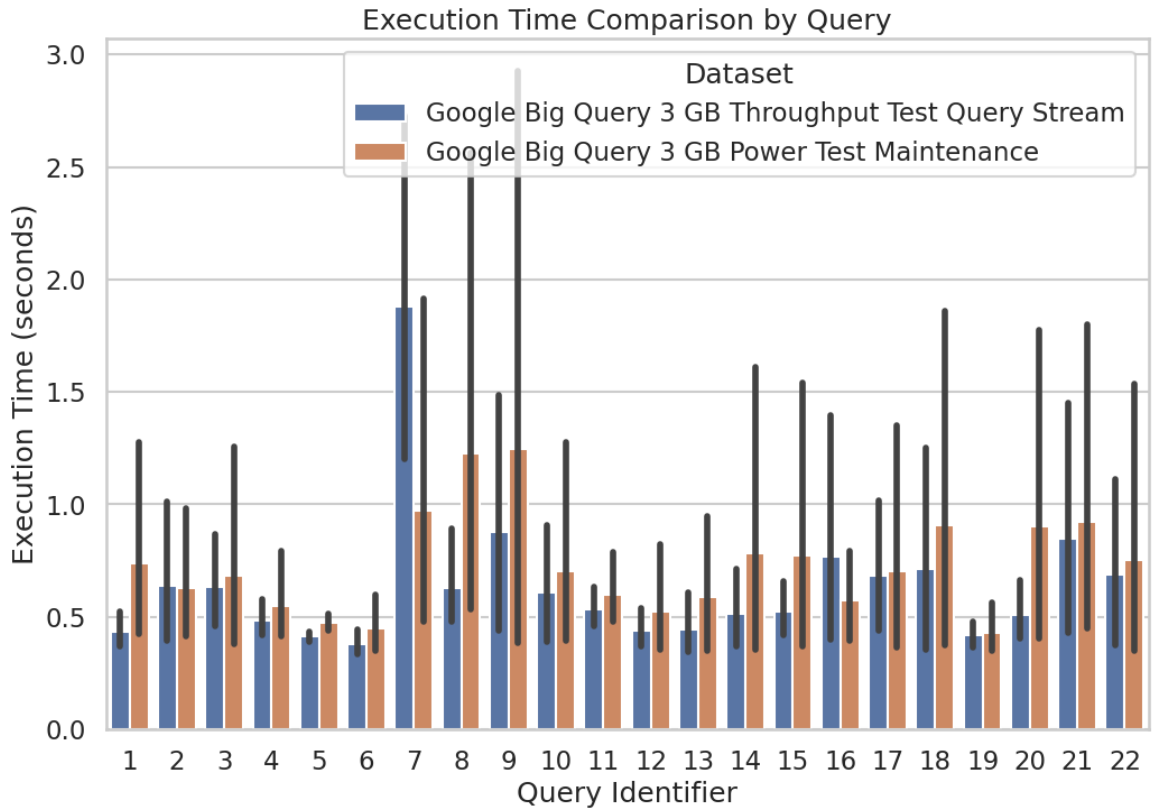


Figure 4.24: Comparison Throughput Test in 3 GB

Let's observe Figure 4.21, Figure 4.22, Figure 4.23, Figure 4.24. Running two query streams

in parallel doesn't necessarily increase execution time; in fact, it can reduce the time for certain queries, such as queries 4, 5, 12, etc. The reason parallel execution of query streams in the BigQuery database doesn't take longer than running a single query stream alone might stem from its underlying parallel processing capabilities. BigQuery efficiently handles parallel tasks by distributing the workload across multiple nodes and utilizing its scalable architecture. As a result, the overall execution time for concurrent query streams remains similar to or even shorter than that of a single query stream due to its optimized parallel processing and resource allocation.

Figure 4.25, Figure 4.26, Figure 4.27, and Figure 4.28 respectively display the execution time of the query stream on 0.5GB, 1GB, 2GB, and 3GB for Snowflake. We can observe a similar pattern, Which means running streams in parallel doesn't necessarily increase query execution time. In Snowflake's database, the parallel execution of query streams doesn't necessarily take longer than running a single query stream alone due to its optimized architecture and robust parallel processing capabilities. Snowflake leverages a distributed computing model, allowing it to efficiently handle concurrent tasks by dividing the workload across multiple nodes. This parallel execution ensures that the overall execution time for multiple query streams remains comparable to or even faster than executing a single query stream. Snowflake's parallel processing and resource allocation enables efficient query execution, maintaining consistent or improved performance across concurrent queries.

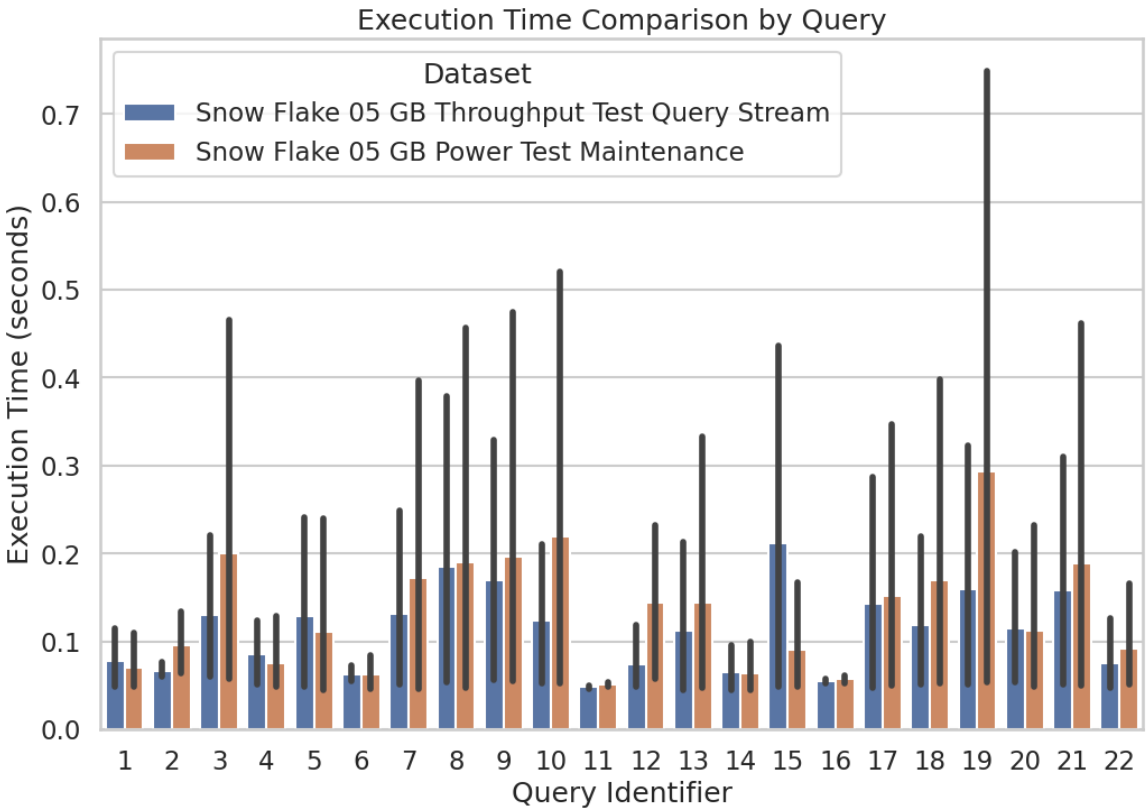


Figure 4.25: Comparison Throughput Test in 0.5 GB

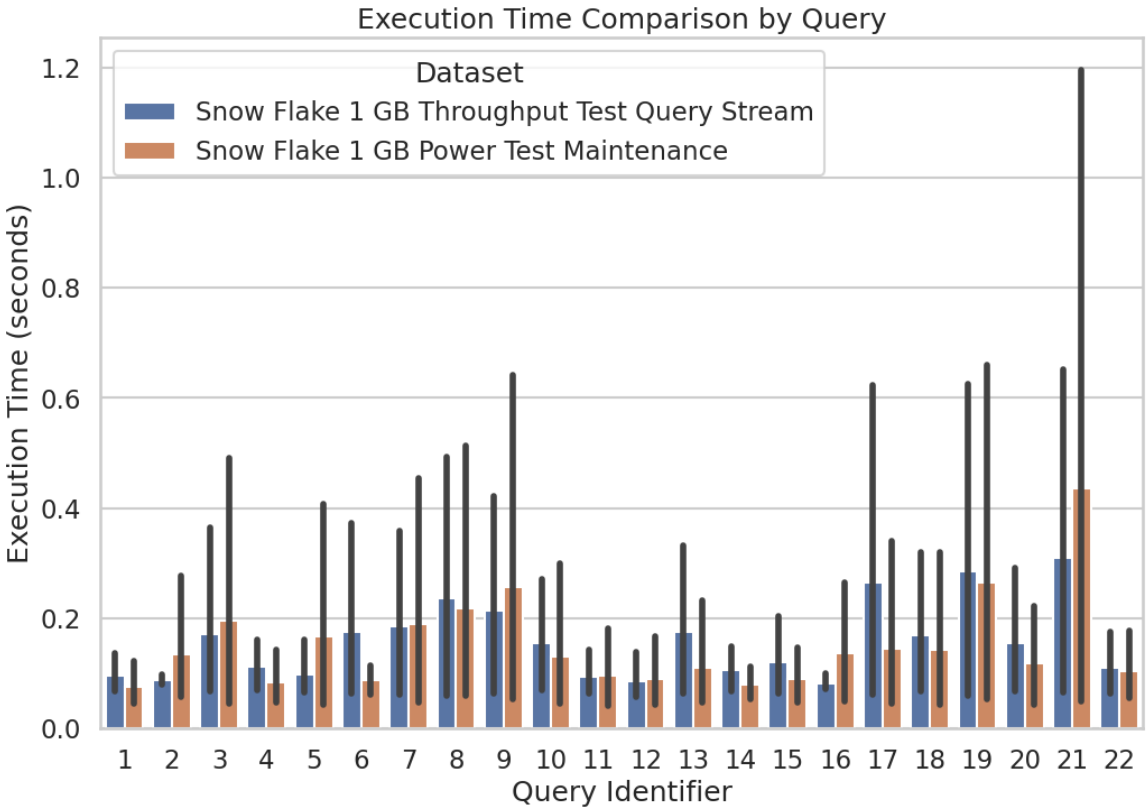


Figure 4.26: Comparison Throughput Test in 1 GB

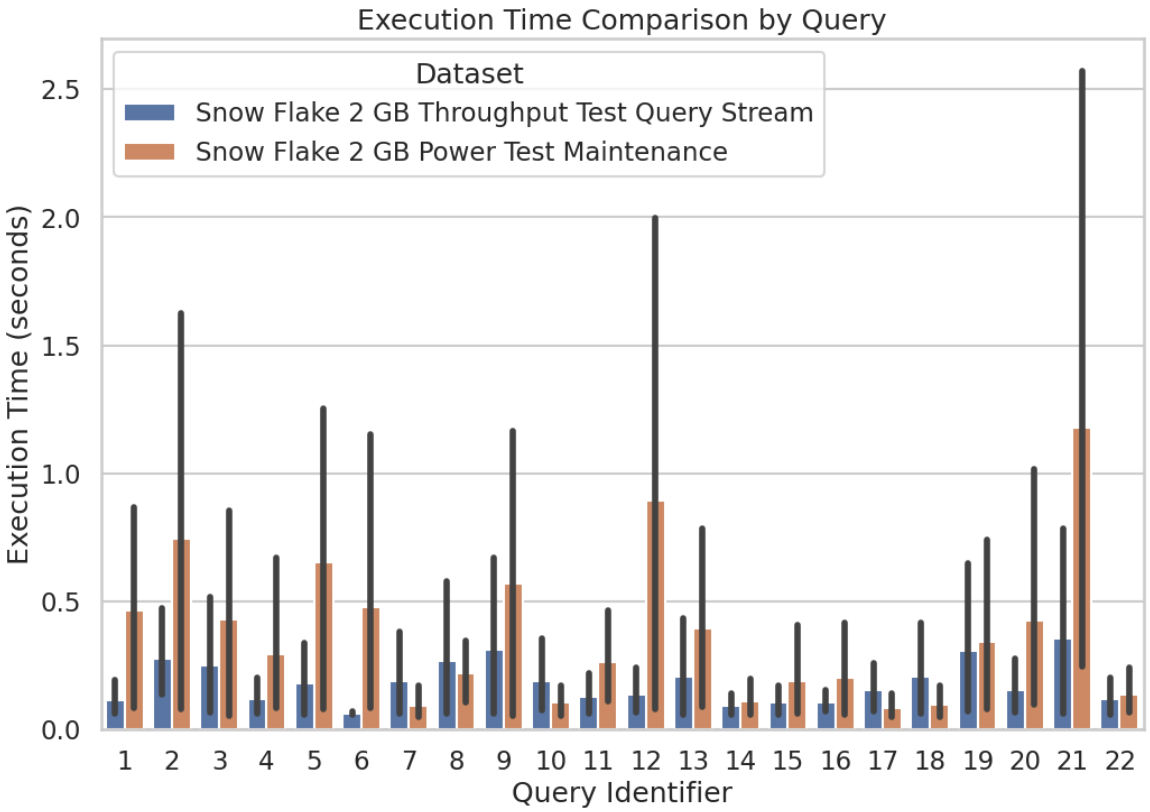


Figure 4.27: Comparison Throughput Test in 2 GB

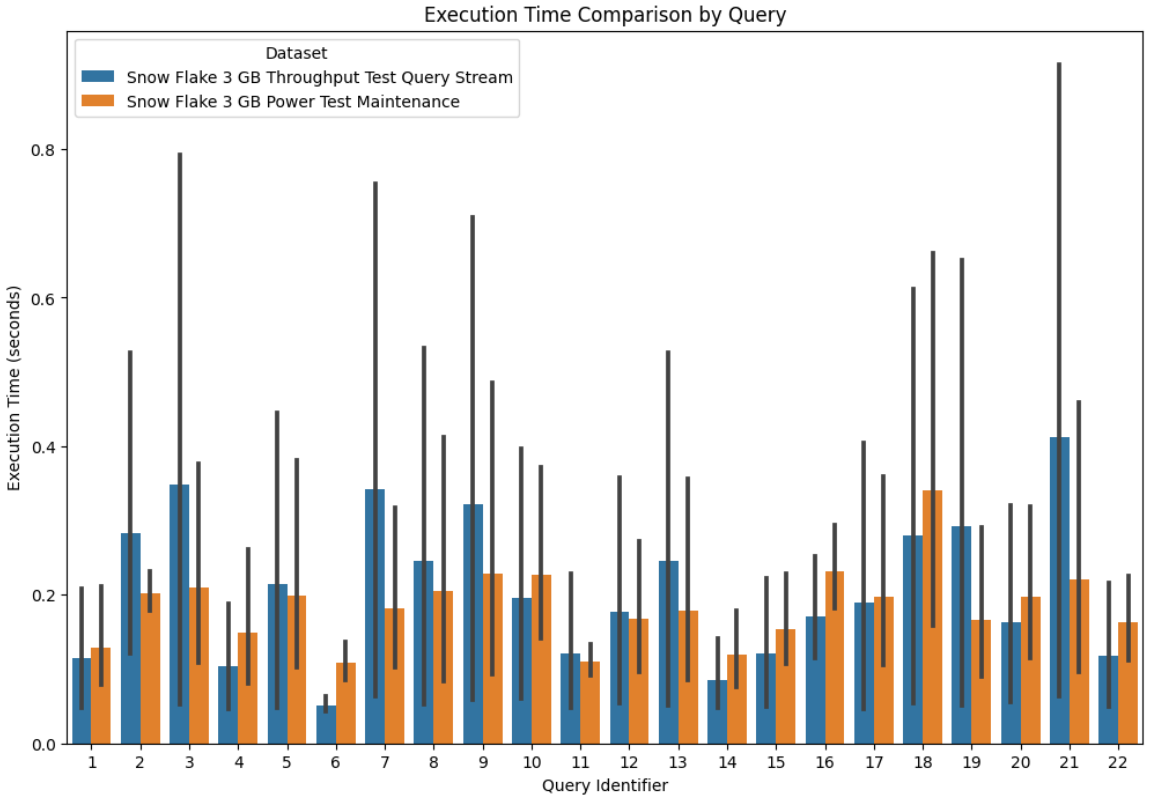


Figure 4.28: Comparison Throughput Test in 3 GB

# 5 Conclusion

## 5.1 Conclusion

In this project, we investigate the performance of two cloud databases, Google BigQuery and Snowflake, and compare their performance. We began with a throughout investigation of the two cloud warehouses as elaborated in Chapter 1. Then the detailed requirement of the benchmark TPC-H was introduced in Chapter 2. The implement process is conducted after and based on those theories, as shown in Chapter 3. The empirical results and their corresponding analyses were consolidated in Chapter 4.

It is obvious that the performance in load test, power test and throughput test of Google BigQuery is inferior to that of Snowflake. Snowflake demonstrates superior performance in loading time and query execution time, and it also outperforms Google BigQuery in handling multiple users. However, Google BigQuery may exhibit better performance on larger data sets, which can be explored in future research.

In regard of maintenance test, in relatively smaller datasets, such as the tested datasets ranging from 0.5G to 3G, in BigQuery, the execution times for inserting and deleting data tend to increase as the database size grows. However, in the case of Snowflake, the execution times for inserting and deleting data might not necessarily increase with the database size. We suppose this is attributed to Snowflake's unique log-structured storage approach and the underlying storage layer structure, which also can be explored in future research.

In our test, the reduced execution times observed in certain queries after data insertion could be a result of the relatively minimal size of the inserted data. This limited impact on execution times compared to the original dataset might be the reason behind this occurrence. We can validate this hypothesis by inserting substantial volumes of data in the future.

Another point not to be overlooked is that both BigQuery and Snowflake are distributed data

processing engines capable of effectively distributing workloads across multiple computational resources and executing multiple queries simultaneously. Hence, running query streams in parallel does not significantly increase the overall execution time; instead, it might potentially accelerate query processing to some extent.

# Articles

McKnight, William and Jake Dolezal (2020). “High-Performance Cloud Data Warehouse Performance Testing”. In: URL: https://www.actian.com/wp-content/uploads/2020/10/GigaOm-high-performance-cloud-data-warehouse-performance-testing.pdf.

# Online Sources

Cloud, Google (n.d.). *Google Cloud*. URL: https://cloud.google.com/learn/what-is-a-data-warehouse.

Google (n.d.). *Google BigQuery*. URL: https://cloud.google.com/bigquery.

McKnight, William (2019). *Cloud Data Warehouse Performance Testing*. URL: https://gigaom.com/report/cloud-data-warehouse-performance-testing/?utm_source=xp&utm_medium=blog&utm_campaign=content#post-id-960578.

Snowflake (n.d.). *Snowflake*. URL: https://www.snowflake.com/).

*TPC-H* (n.d.). URL: https://www.tpc.org/tpch/default5.asp.

# Books

Vaisman, Alejandro and Esteban Zimányi (2022). *Data Warehouse Systems Design and Implementation*. Springer.

# 6 Appendix

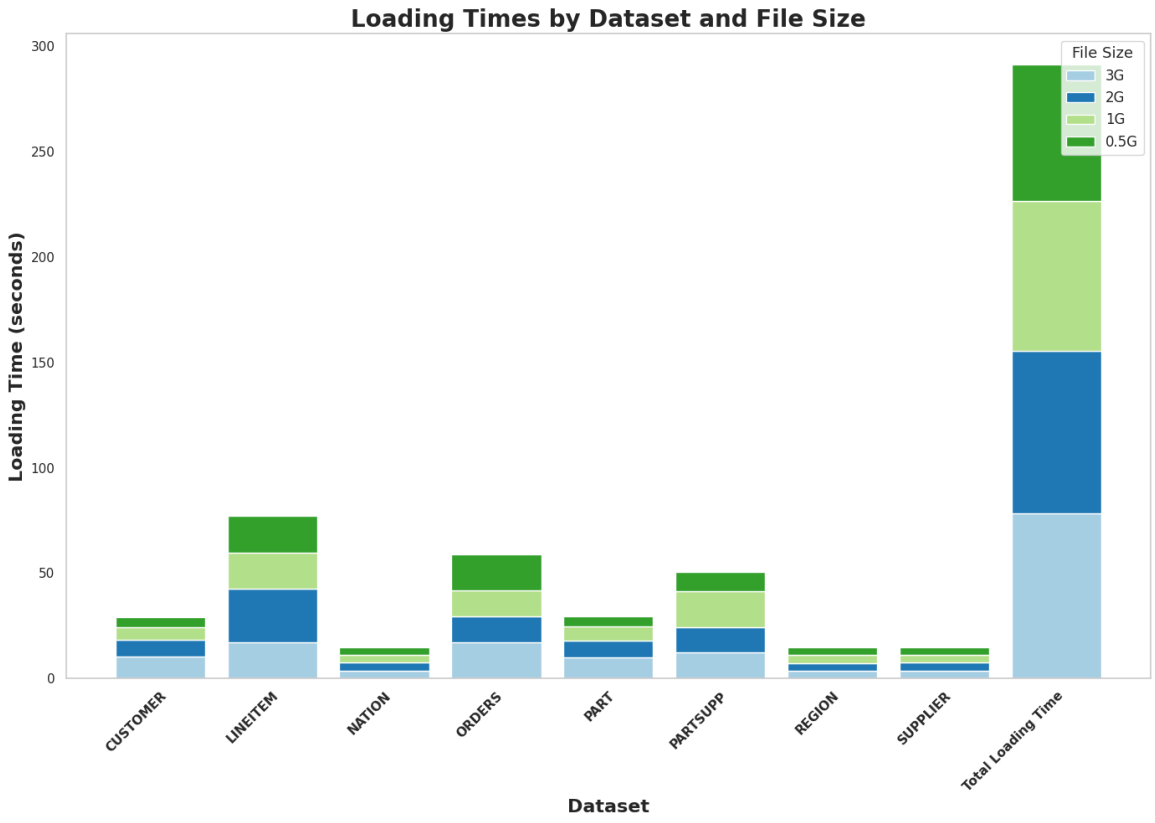


Figure 6.1: Big Query Load Time

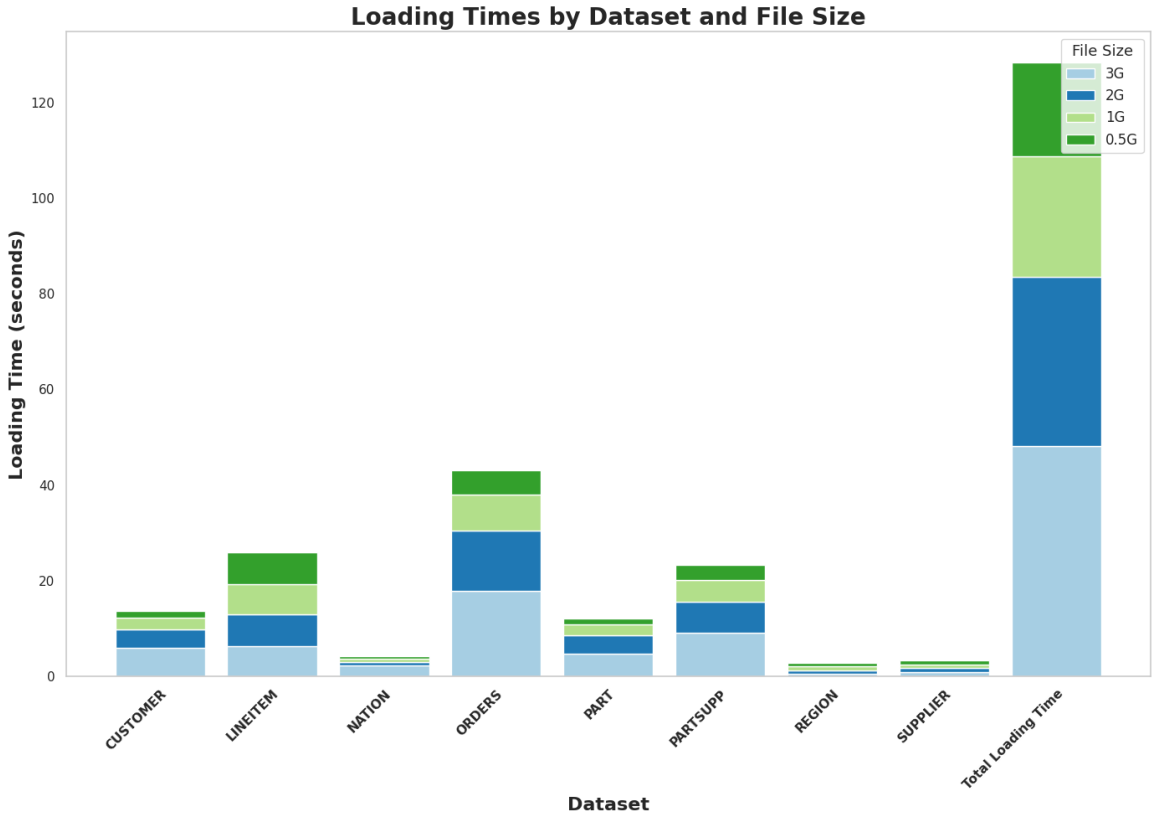


Figure 6.2: Snow Flake Load Time

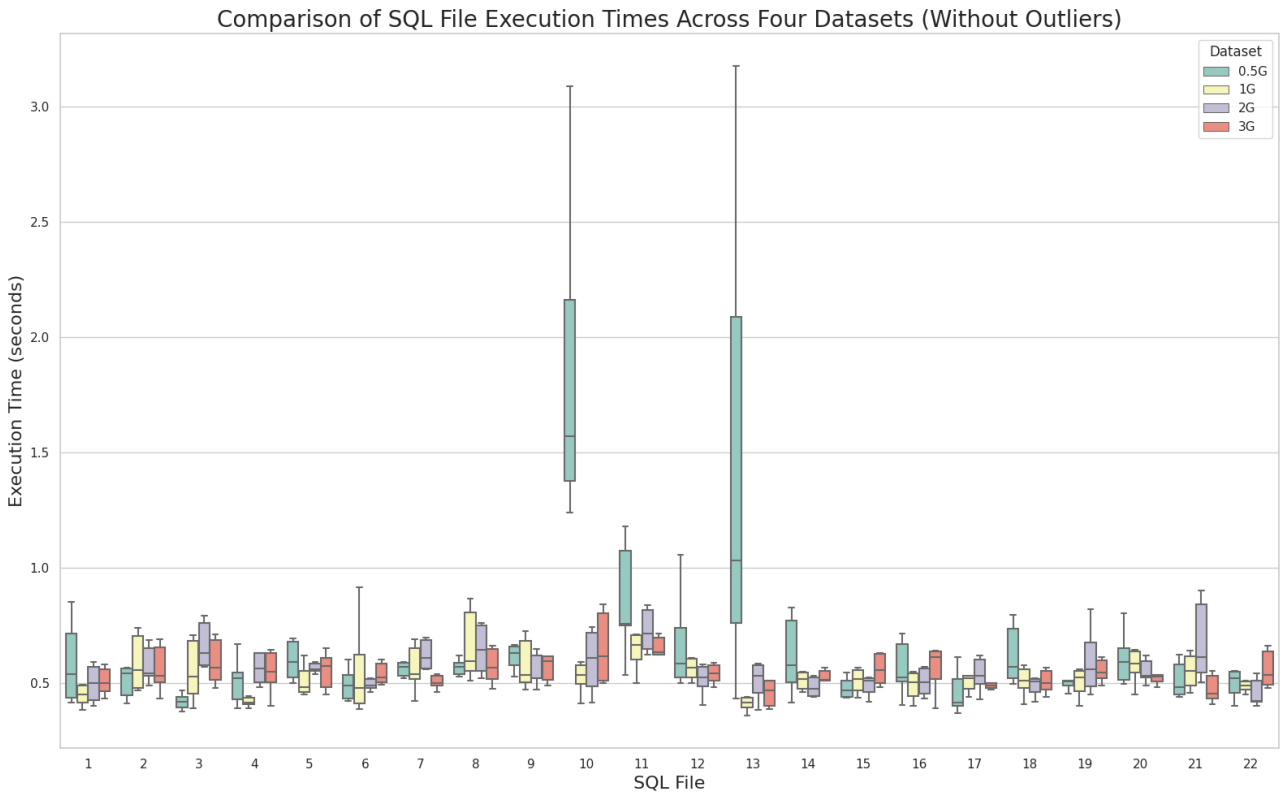


Figure 6.3: Big Query Power Test

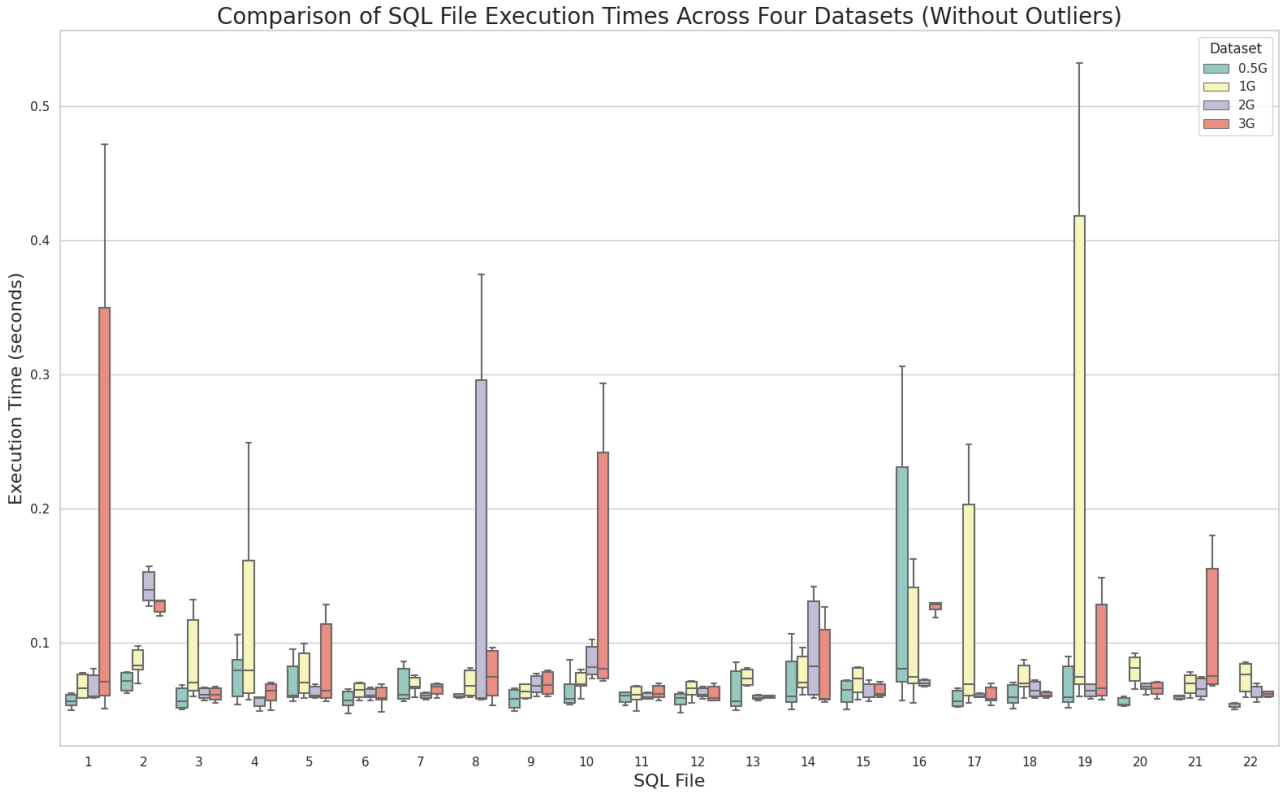


Figure 6.4: Snow Flake Power Test

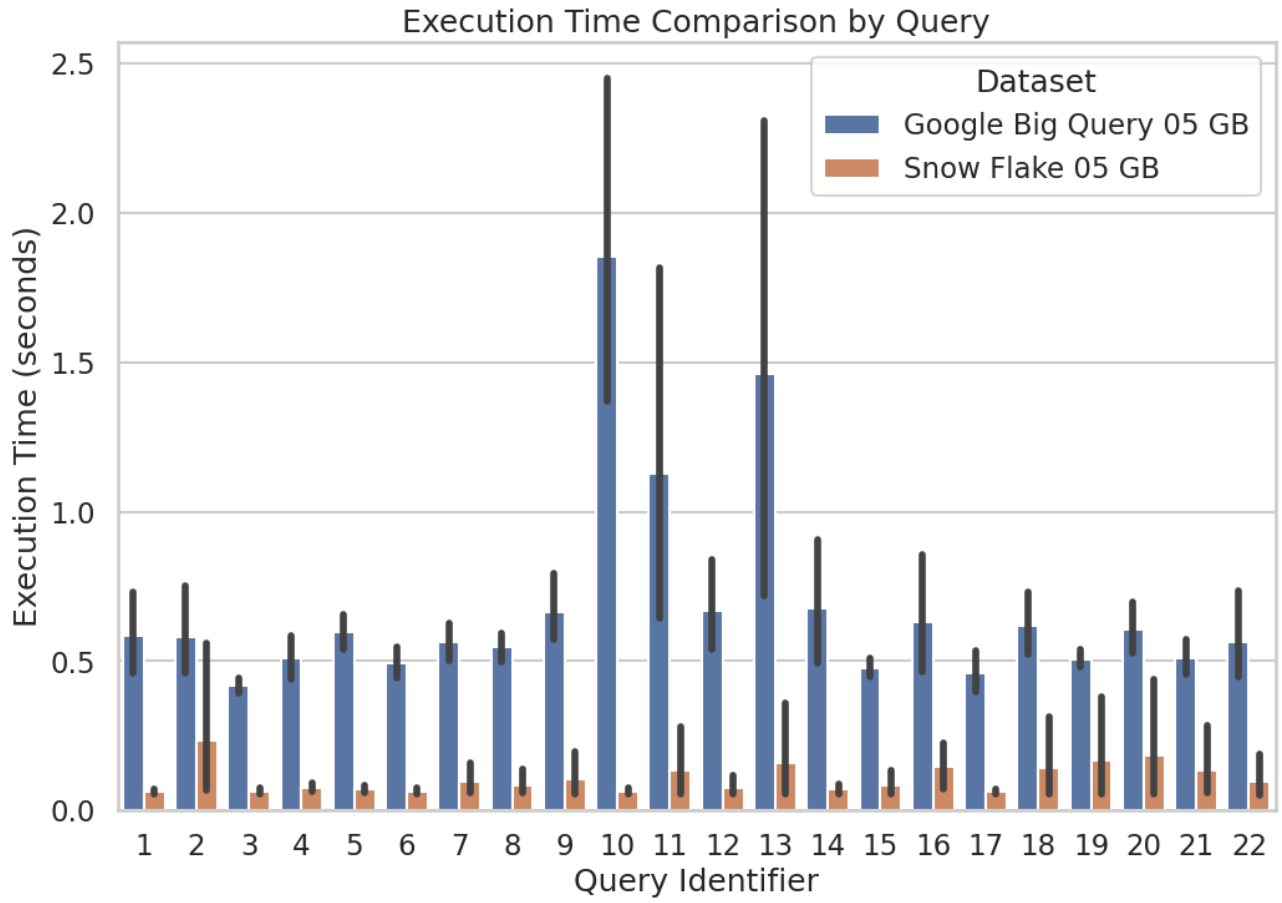


Figure 6.5: Comparison of Big Query and Snow Flake Power Test at 0.5 GB

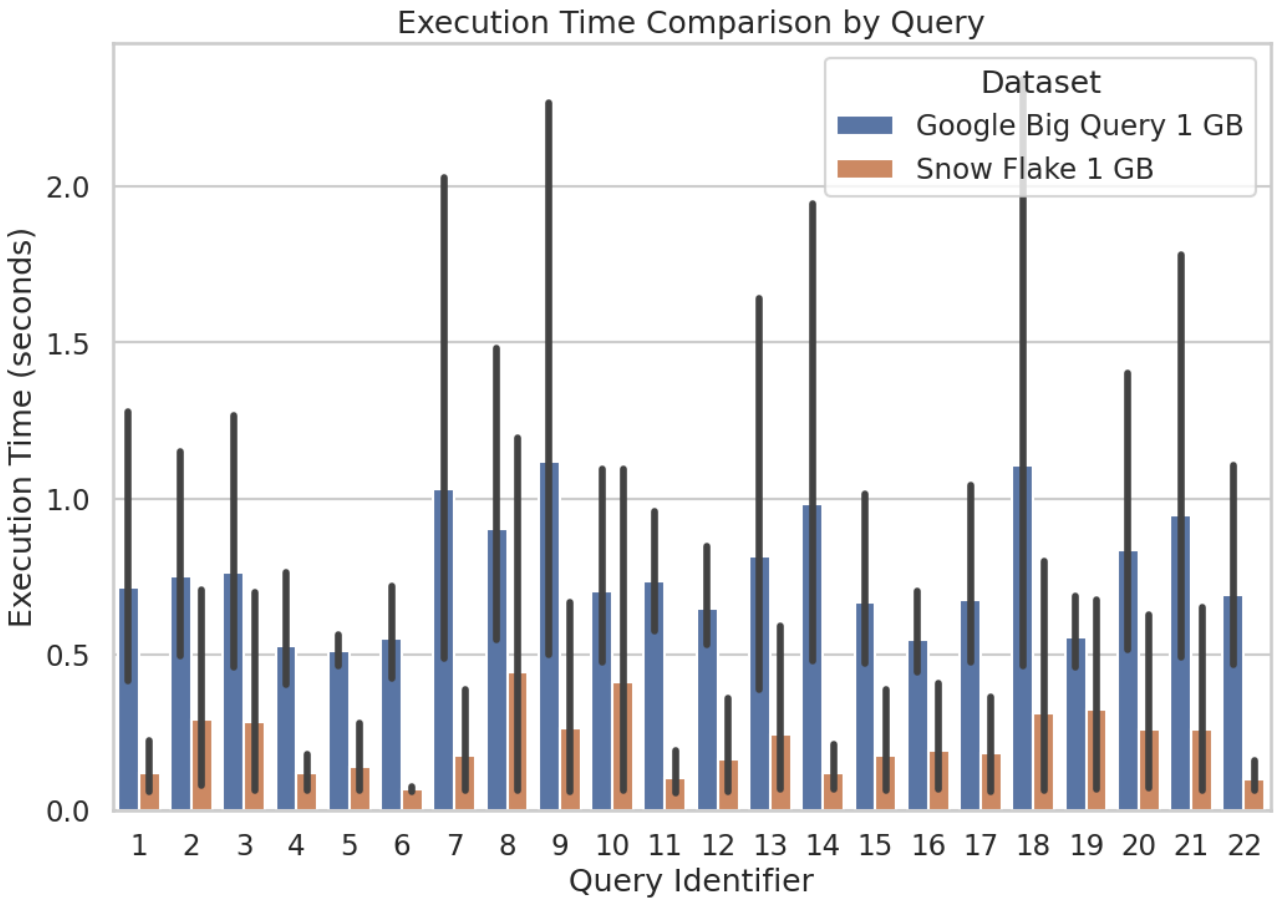


Figure 6.6: Comparison of Big Query and Snow Flake Power Test at 1 GB

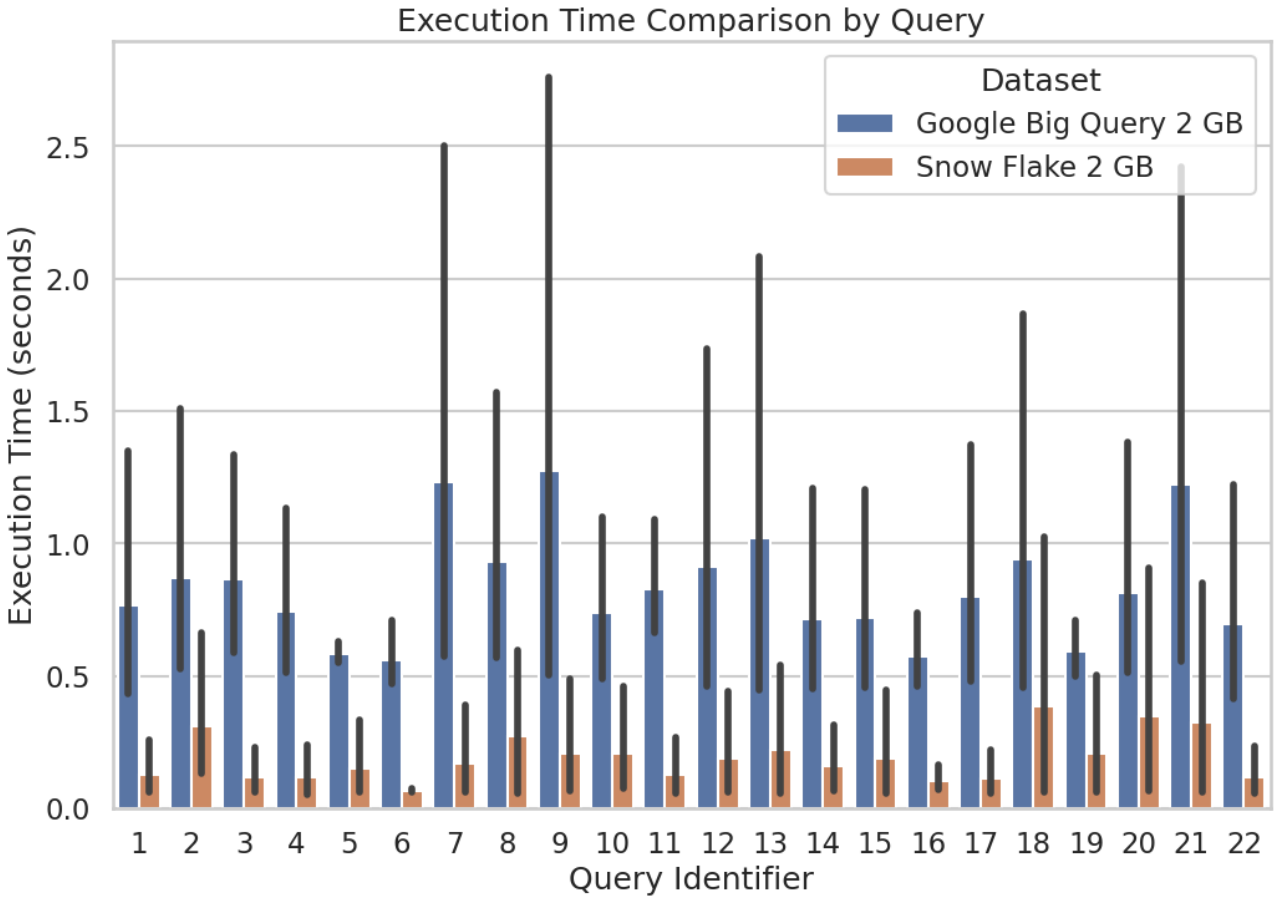


Figure 6.7: Comparison of Big Query and Snow Flake Power Test at 2 GB

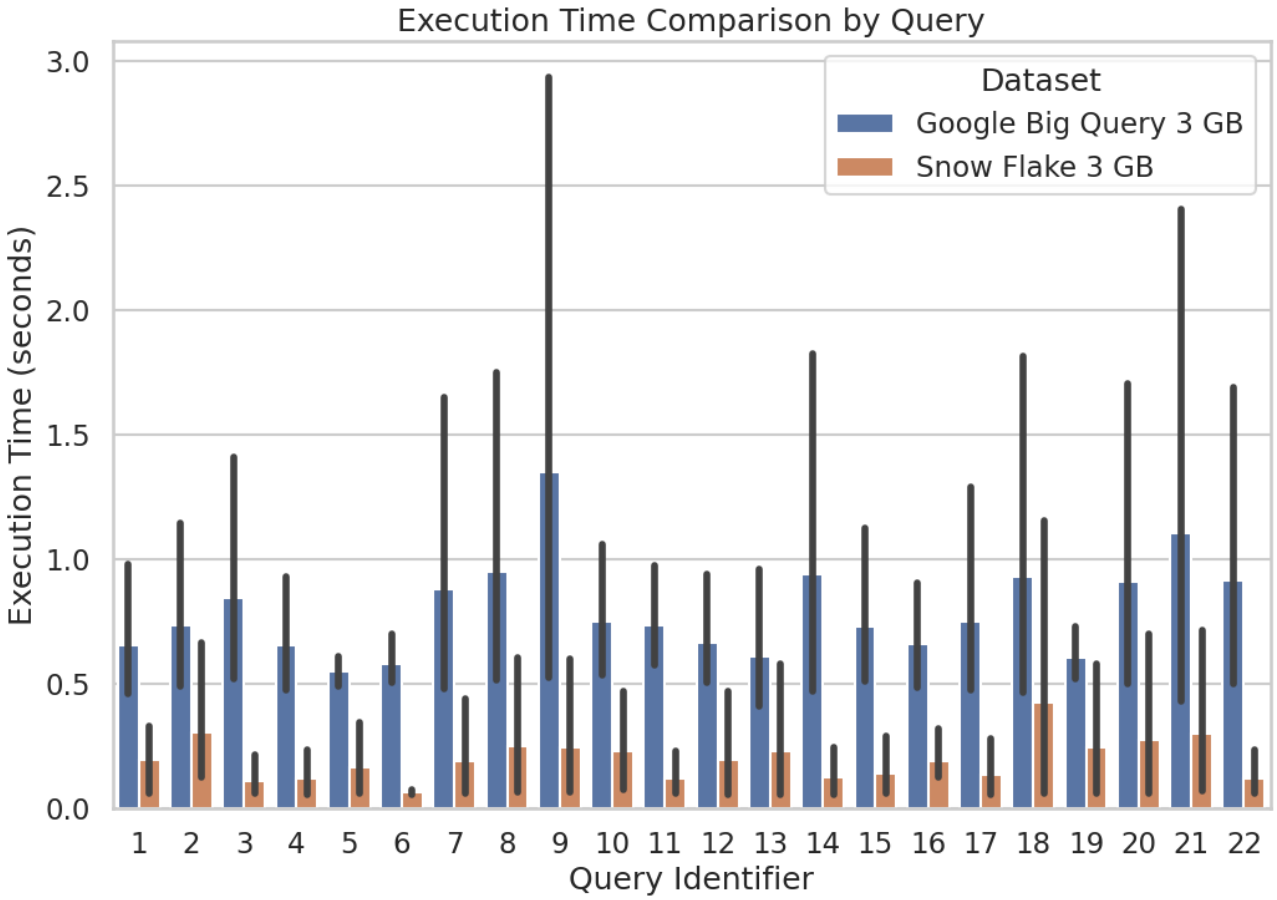


Figure 6.8: Comparison of Big Query and Snow Flake Power Test at 3 GB

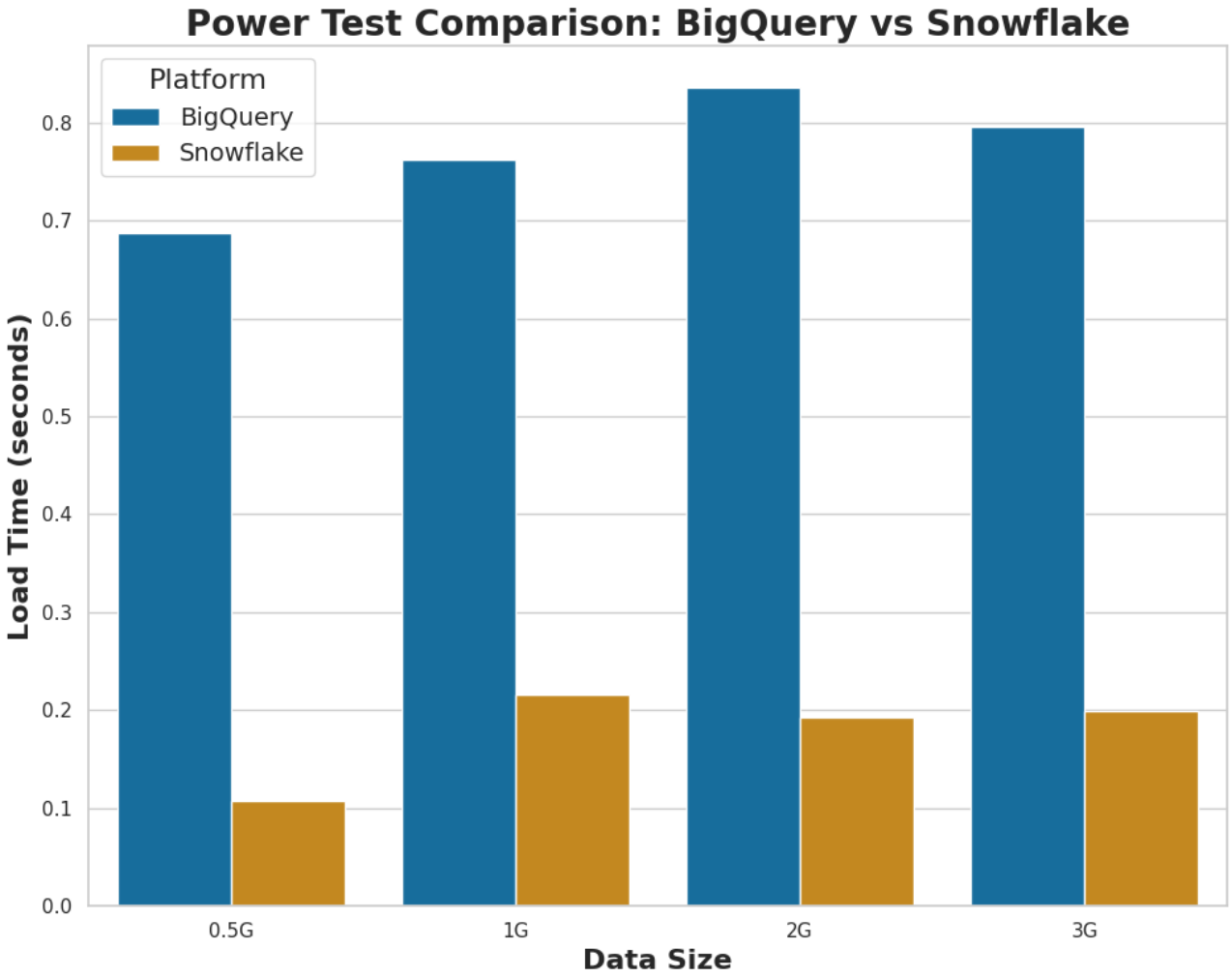


Figure 6.9: Comparison of Big Query and Snow Flake Power Test at All Scles