ECOLE POLYTECHNIQUE DE BRUXELLES

2023-2024

INFO-H419 - Data Warehouse

Project 1 TPC-DS Benchmark Using DuckDB

Group:

Jintao Ma

Linhan Wang

Iyoha Peace Osamuyi

Hieu Nguyen Minh

Professor

Esteban Zimányi

November 2023

Contents

Listings

List of Figures

Abstract

In this project we have implemented the TPC-DS Benchmark on DuckDB. It is a free and open-source database created by CWI.

In Chapter 1, we presented the general objectives and mission requirements of the project. In Chapter 2, we reviewed the basics of data warehousing and TPC-DS benchmarking and introduced the programming tools and databases we used. In Chapter 3, we described concretely the whole process of how we made the different tests executed in the benchmarking. This contains assumptions and code implementations. In Chapter 4, we used a lot of charts to compare, analyze, interpret, and completed some optimizations. In Chapter 5, we summarized our conclusions.

All experimental and analytical codes are in this github repository:
`https://github.com/woshimajintao/TPC_DS_for_DuckDB.git`

1 Introduction

1.1 Overview

The Project entails performing TPC-DS Benchmarking on a selected Database Management System which include DuckDB, PostgreSQL, SparkSQl, MySQL etc. The Benchmarking is done with several scale factors which determine the size of the resulting data warehouse and then implement some metrics to evaluate the performance of the database. The project is carried out in groups of 4 persons and delivers a report in pdf file explaining the essential aspects of your implementation, and a zip file containing the code of your implementation, with all necessary instructions to be able to replicate your implementation. This report employs DuckDB as the preferred choice of Database Management System on which the TPC-DS benchmark is performed.

1.2 Aim and Objectives

The aim of this report is to implement and evaluate the TPC-DS benchmark on DuckDB, with the following objectives:

- Learning how to properly perform a benchmark efficiently.
- Evaluating the benchmarking performance and analyze the results.
- Optimizing queries that are complex and time consuming.
- Visualizing the result.

- Testing the performance with Apache Jmeter.

1.3 Limitations

The tool used has a limit on the resources and memory size, thereby preventing scaling higher benchmarks. Another Limitation was that the SQL Keyword "**ALTER**" hasn't been fully implemented to handle referential integrity constraints. Lastly, time was a limiting factor for us to thoroughly expose the objectives stated above.

2 Data Warehouse and TPC-DS Benchmarking

2.1 Data Warehouses

In today's data-driven world, organizations are facing challenges of handling enormous information from various sources, including transactions, customers' behavior, web activities, etc. To derive meaningful insights from collected data, people in those organizations resort to decision support tools for analyzing the data. The whole process of data processing involves a lot of steps, from data extraction, data transformation, to data cleaning and data storage. In this regard, data are stored in a common repository where users can access and get immediate responses to complex queries. This repository is called a *data warehouse* [VZ14].

According to Vaisman and Zimányi [VZ14], a data warehouse can be defined as a collection of subject-oriented, integrated, nonvolatile, and time-varying data to support management decisions. These concepts can be broken down as:

- Data is subject-oriented when the data warehouse targets one or several subjects of analysis according to the analytical requirements of the organization.
- It is integrated when data from different sources (operational and external) are integrated together.
- It is nonvolatile in the sense that modification and removal of data is not allowed. This is because the data warehouse accumulates data for a long period of time, which cannot be changed.
- Finally, time-varying, because we are dealing with how the data has evolved over time.

A data warehouse is a central repository designed to store, organize, and manage data from different sources, making it available for data analysis and reporting. Unlike traditional databases, data warehouse are designed for efficient retrieval and processing of large data sets. To be more specific, data tuples (i.e. rows in tables) are used as read-only operations, such as querying and reporting, rather than modifying the data.

2.2 TPC-DS Benchmark

Benchmarking is a process of comparing and evaluating an organization's performance against an external standard or best practice model. It involves measuring and analyzing various aspects of an organization's processes, performance, and practices against those of other similar organizations, with the goal of identifying areas for improvement and setting goals for achieving better results [Mah21]. Benchmarking a database is the process of performing well defined tests on that particular database for the purpose of evaluating its performance [Sam21]. Benchmark test results facilitate means for cross platform comparisons of various database management systems by providing valuable information to database professionals on whether to utilise a particular database product. Hence, we are carrying and evaluating the performance of DuckDB against the TPC-DS Benchmarking standard.

Transaction Processing Performance Council (TPC) is a non-profit organization founded to define transaction processing and database benchmarks and to disseminate objective, verifiable TPC performance data to the industry [1]. In terms of data warehouses, TPC provide two benchmarks: TPC-DS and TPC-DI for the evaluation of decision support and data integration, respectively.

The TPC-DS is a decision support benchmark that models several generally applicable aspects of a decision support system, including queries and data maintenance. The benchmark provides a representative evaluation of the System Under Test's (SUT) performance as a general purpose decision support system [TPC21].

This TPC-DS benchmark depicts a decision support systems that:

- Examine large volumes of data;

[1] https://en.wikipedia.org/wiki/Transaction_Processing_Performance_Council

- Give answers to real-world business questions;
- Execute queries of various operational requirements and complexities (e.g., ad-hoc, reporting, iterative OLAP, data mining);
- Are characterized by high CPU and IO load;
- Are periodically synchronized with source OLTP databases through database maintenance functions.
- Run on "Big Data" solutions.

In addition, a benchmark result measures query response time in single user mode, query throughput in multi user mode and data maintenance performance for a given hardware, operating system, and data processing system configuration under a controlled, complex, multi-user decision support workload [TPC21].

2.3 DuckDB

DuckDB is an in-process SQL OLAP database management system.[2] The first open-source version of it was released by Hannes Mühleisen and Mark Raasveldt,who were researchers in the Database Architectures research group at Centrum Wiskunde & Informatica (CWI) in 2019.[RM19]It was the first purpose built in-process Online Analytical Processing(OLAP)-database management system.[3]It was designed to be fast, reliable and easy to use. There have been some important features of this tool:

2.3.1 Simple and portable

For DuckDB, there is no DBMS server software to install, update and maintain. It does not run as a separate process, but completely embedded within a host process. For the analytical use cases that DuckDB targets, this has the additional advantage of high-speed data transfer to and from the database. In some cases, DuckDB can process foreign data without copying. We can use

[2]https://duckdb.org/

[3]https://www.cwi.nl/en/news/cwi-spin-off-company-duckdb-labs-provides-solutions-for-fast-database-analytics

DuckDB without installing, updating, or maintaining separate DBMS server software. It doesn't run as an independent process but is entirely integrated within a host process. This approach provides added benefits for analytical use cases, notably enabling high-speed data transfer to and from the database.

2.3.2 Feature-rich

DuckDB enables users to run complex SQL queries and provides APIs for Java, C, C++, and more. It is also deeply integrated into Python and R, enabling users to conduct efficient interactive data analysis; thus, you can interact with DuckDB from your preferred programming language. There is also access to extra SQL keywords that make SQL queries easier to write, such as EXCLUDE, REPLACE, and ALL.

2.3.3 Fast

DuckDB can support the Online analytical processing (OLAP).It features a query execution engine that employs columnar vectorization. While a few queries may still be interpreted, a substantial batch of values, known as a "vector," is processed in a single operation. This approach significantly minimizes the overhead found in conventional systems like PostgreSQL, MySQL, or SQLite, which handle each row sequentially. The utilization of vectorized query execution significantly enhances performance in OLAP queries. They used an interactive SQL shell to demonstrate the capabilities of an analytical database in the browser[Koh+22].

2.3.4 Free and Extensible

Because the DuckDB is Open Source, all of the source code is freely available on GitHub. You can acess it by click the link: https://github.com/duckdb They also invite developers from anyone to take issues.

2.4 Other Tools Used

We use the following tools to implement the benchmark:

2.4.1 Python

Python is an interpreted, general-purpose, interactive, object-oriented programming language. It incorporates modules, exceptions, dynamic typing, very high level dynamic data types, and classes. It supports multiple programming paradigms beyond object-oriented programming, such as procedural and functional programming. Python combines remarkable power with very clear syntax. It has interfaces to many system calls and libraries, as well as to various window systems, and is extensible in C or C++ [Pyt23]. In this project, we used python to write scripts for creating the database, loading of tables in our database, querying and performing various evaluation metrics. Also, Jupyter notebook with matplotlib was used to visualize our data. Python version 3.10 was used in this project.

2.4.2 Code Editor: Microsoft Visual Code

Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages and runtimes (such as C++, CSharp, Java, Python, PHP, Go, .NET) [Cod23]. It was useful in this project to write our python scripts and running dsdgen and dsqgen on command line.

2.4.3 DBeaver

DBeaver is a free cross-platform universal database tool for developers, database administrators, analysts, and everyone working with data. It supports all popular SQL databases like DuckDB,MySQL, MariaDB, PostgreSQL, SQLite, Apache Family, and more. It was used to test our queries and validating the output [Com23]

2.4.4 Apache Jmeter

The Apache JMeter is pure Java open source software, which was first developed by Stefano Mazzocchi of the Apache Software Foundation, designed to load test functional behavior and measure performance. You can use JMeter to analyze and measure the performance of web applications or a variety of services. Performance Testing means testing a web application against heavy load, multiple and concurrent user traffic. JMeter originally is used for testing Web Application or FTP application. Nowadays, it is used for a functional test, database server test etc. In this Project, it was used to perform scalability testing on the TPC-DS dataset and Duckdb. [Ham23]

3 Benchmark Implementation Process

3.1 Setting up the Project

Some members of our group used Windows OS, MacOS and Ubuntu OS so we installed DuckDB directly on our computers . The Apple MacBook was the main machine for the benchmark implementation as all the results were obtained from it, while the Hp Elitebook 840 G5 was used for performing the scalability test with Apache Jmeter.

3.1.1 Hardware Specification

In order to carry out the benchmark experimentation, Apple Macbook and HP EliteBook 840 G5 were used. Table 3.1 shows the hardware specification of this computer:

Name	**HP EliteBook 840 G5**	**Apple Macbook**
Operating System	Ubuntu 22.04.3 LTS	MacOS 13.0
System Type	64 Bits	64 Bits
CPU	Intel® Core™ i7-8650U	Apple M1 Pro
CPU Frequency	1.90GHz × 8	3.2 GHz
RAM capacity	24 GB	16 GB
RAM type	DDR4	SDRAM
Hard disk capacity	512 GB	512 GB

Table 3.1: System Specifications.

3.2 Implementing TPC-DS on DuckDB

Figure 3.1 shows a business process model depicting a brief rundown of the implementation the TPC-DS benchmark on DuckDB.

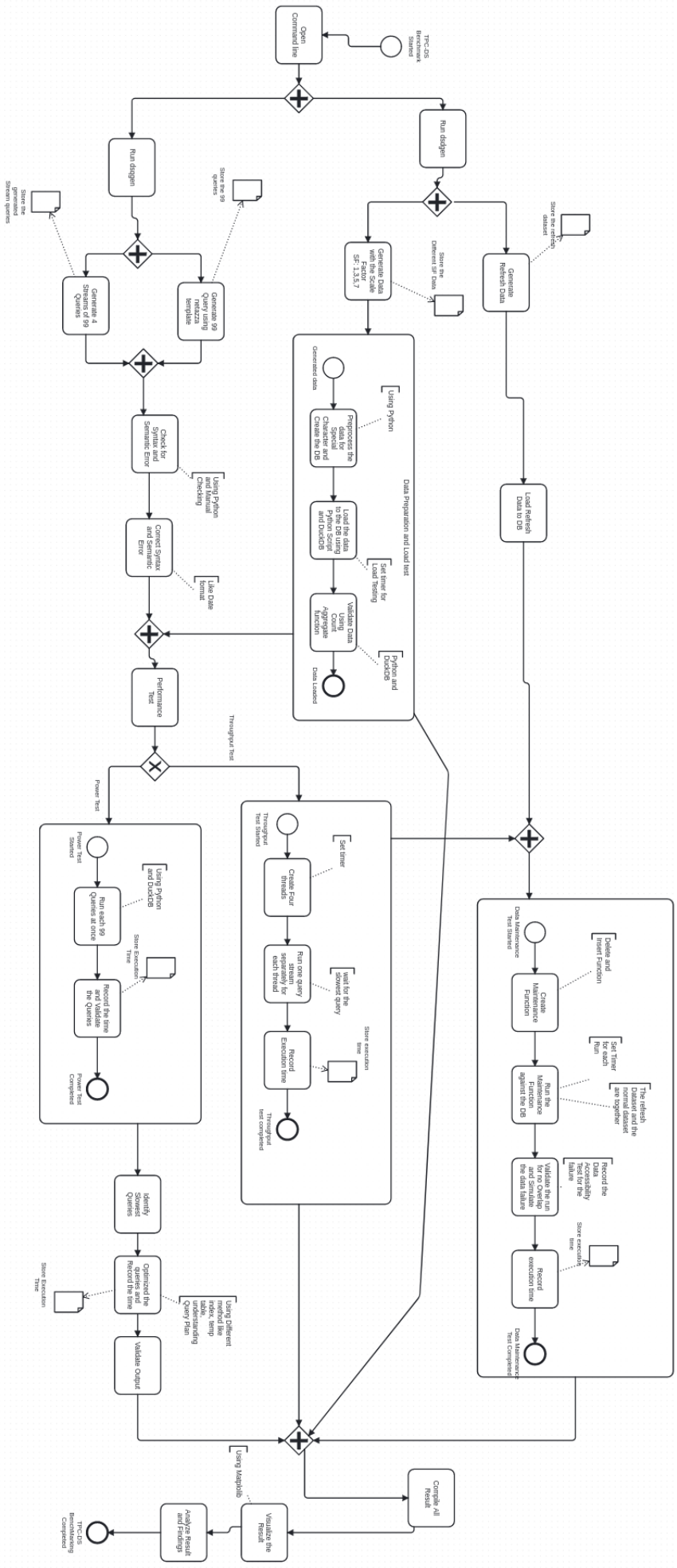


Figure 3.1: TPC-DS Benchmarking BPMN Diagram

From Figure 3.1, some of the process that was done include:

- Using dsdgen Utility from the TPC-DS tool kit to obtain load data for all our scale factor and also getting our refresh dataset too
- Generating 99 queries via the dsqgen using the Netezza SQL template and also four stream queries for our throughput test
- Matching the queries with DuckDB syntax to check for syntax mismatches and error.
- Using python scripts with DuckDB to perform all tests on the queries and compile all results.
- Discover slowest running queries in order to analyze and optimize them.
- Visualize the outputs and provide detailed reports.

3.2.1 Scaling and Database Population

The TPC-DS benchmark defines a set of discrete scaling factors based on the size of the raw data produced by dsdgen (from the TPC-DS toolkit). This raw data will be used to perform the benchmark [TPC21]. The standard set of scale factors defined for TPC-DS is 1TB, 3TB, 10TB, 30TB, 100TB, but due to storage, computing and memory limit , we opted for much smaller scale factors to perform our tests: 1GB, 3GB, 5GB, 7GB

3.3 Implementation Process

The implementation of the TPC-DS benchmarking were done in different stages, which include:

1. Database schema creation
2. Data generation
3. Query generaion
4. Tests for 4 scale factors

3.4 Setup

3.4.1 DuckDB installation

We use `pip`[1] - the package installer for Python - to install the DuckDB.

```
pip install duckdb==0.9.1
```

Listing 3.1: DuckDB installation.

3.4.2 TPC-DS kit installation

We install the TPC-DS kit from the tpcds- toolkit downloaded from TPC website[2].

3.4.3 Apache Jmeter Installation

On Linux

Installing Apache JMeter on Ubuntu and other Linux Distributions requires java installed. You can either use JRE or JDK.

1. To install the default java on Ubuntu, type: $ sudo apt install default-jre
2. Download Apache JMeter by visiting Apache JMeter website to download the latest version of JMeter.
3. Extract Apache jmeter Archive file from Downloads Folder
4. Run the Jmeter by first changing the directory as follows: $ cd apache-jmeter-5.6.2/bin
5. Now run the JMeter using the following command: $ ./jmeter

[1] https://duckdb.org/docs/installation/index
[2] https://www.tpc.org/tpcds/

On Windows

Installing Apache JMeter on Windows requires java installed.

1. If Java is not install, Download and Install it [3]
2. Download Apache JMeter by visiting Apache JMeter website[4] to download the latest version of JMeter.
3. Unzip Apache jmeter Archive file from Downloads Folder
4. Go to the jmeter directory and the bin folder, double click on the jmeter.bat file

3.5 Database schema creation

The first step in the process is to connect to DuckDB and create a database:

```
import duckdb
con = duckdb.connect('sc_1.db') # SF-1
```

Listing 3.2: Database setup

Then we use the scripts in `tpcds-kit/tools/tpcds.sql` to create the database scheme. Below is an example of table creation in the scripts:

```
con.sql(
"create table dbgen_version
(
    dv_version                varchar(16)                   ,
    dv_create_date            date                          ,
    dv_create_time            time                          ,
    dv_cmdline_args           varchar(200)
);"
)
```

Listing 3.3: Database setup

[3] https://www.oracle.com/java/technologies/downloads/
[4] https://jmeter.apache.org/download_jmeter.cgi

3.6 Data generation

We generated data for evaluation with the TPC-DS program dsdgen. At this step, we are in the `tpcds-kit/tools` folder. We ran the following command lines to generate data in 4 scale factors and store them in the `generated-data` folder.

```
./dsdgen -scale 1 -dir "../../generated-data/scale_1" -delimiter "," # SF 1
./dsdgen -scale 3 -dir "../../generated-data/scale_3" -delimiter "," # SF 3
./dsdgen -scale 5 -dir "../../generated-data/scale_5" -delimiter "," # SF 5
./dsdgen -scale 7 -dir "../../generated-data/sale_7" -delimiter "," # SF 7
```

Listing 3.4: Data generation.

After this stage, 25 `.csv` files are generated regardless of the scale factor. They only differ in size for each scale. We show an example of a row in `call_center.csv` in :

```
1|AAAAAAAABAAAAAAA|1998-01-01|||2450952|NY Metro|large|2|1138|8AM-4PM|Bob
   Belcher|6|More than other authori|Shared others could not count fully
   dollars. New members ca|Julius Tran|3|pri|6|cally|730|Ash Hill|Boulevard|
   Suite 0|Midway|Williamson County|TN|31904|United States|-5|0.11|
```

Listing 3.5: Generated data example.

As can be seen from Listing 3.5, data fields are separated by the `"|"` delimiter. A `null` value is represented by `"||"`, and we leave it by default because DuckDB is able to identify it and produce a `NULL` value.

Also, We generated refresh data for all scale factors (1,3,5,7) using dsdgen for separate refresh runs, so that we can implement the data maintenance test. Refresh runs must be executed using refresh data sets in the order generated by dsdgen.

3.7 Query generation

We generated data for evaluation with the TPC-DS program dsqgen. We are still in the `tpcds-kit/tools` folder and ran the following command lines to generate 99 queries for SF 1 (we do similarly for other scales) and store them in the `generated-queries` folder.

```
dsqgen -directory "../query_templates/" -input "../query_templates/templates
    .lst" -dialect netezza -scale 1 -output_dir "../../generated-queries"
```

Listing 3.6: Data generation.

This command generates a file `query_0.sql` that contains 99 SQL queries with netezza dialect.

3.7.1 Separating queries from the templates

We have developed a python script called split_queries.py. This script basically splits the generated queries created by the dsqgen called query_0.sql and puts them in 99 different files, named query_n.sql, where n is from 1, ..., 99.

3.7.2 Query Modification

After generating the queries,we had to fix syntax in some SQL files so that we could generate queries that can work well in our DuckDB Database (i.e. uses DuckDB SQL dialect). In particular, we edited SQl files for queries 5, 12, 16, 20, 21, 30, 32, 37, 40, 58, 77, 80, 82, 92, 94, 95, 98. Here are the specific changes we made:

- Adding intervals to the cast(as date) function– to identify the interval of days.
- Changing the reserved keyword used as a variable name e.g in query 77.
- Fixing some ambiguous variable names in query 30 and 58.

3.7.3 Running Queries

DuckDB is an **embedded, in-process**, relational, OnLine Analytical Processing (OLAP) DataBase Management System (DBMS). This means that the DBMS features are running from within the application you're trying to access from instead of an external process your application connects to. In our case , we use Python API to run the queries and also we used DBeaver to test the queries, modify the queries and execute them.

3.8 Load Test

The generated data was loaded into the tables of DuckDB to measure how good the system does this job. This is called **Load Test**. This data loading procedure has to be done automatically, not involving manual interaction so as to measure the time accurately. **The Database Load Time** is the difference between **Load Start Time** and **Load End Time**, where

- Load Start Time is defined as the timestamp when the first character is read from any of the flat files (dat or csv files)
- Load End Time is defined as the timestamp taken when the DuckDB Database is fully populated.

The pseudo code for data loading and load test is as follows:

```
Assumption:
     The schema defined in tpcds.sql in official tools has been loaded to
   the database.

Algorithm:
    set timer start
    load data into database
    set timer end
```

Listing 3.7: Load Test Pseudocode

We excute the following Python script to perform the load test:

```
import duckdb
import os
import time

sc = '1' # scale factor
# specify the path for dataset
csv_folder_name = 'generated_data/sc_'+ sc
csv_list = os.listdir(csv_folder_name)

# set the timer start
start_time = time.time()
for csv_file in csv_list:
```

```
    table_name = csv_file[:-4]
    # load data into tables
    copy_command = "COPY "+table_name+" FROM "+"'"+csv_folder_name+"/"+
   format(csv_file,"")+"'"
    con.sql(copy_command)
# set the timer end
end_time = time.time()
time_taken = end_time - start_time
print(time_taken)
```

Listing 3.8: Python script for load test.

3.9 Power Test

The Power Test measures the ability of the system to process a sequence of queries in the least amount of time in a single stream. This Test is executed immediately following the load test. Only one query can be executed at a time during the power test. The **elapsed time of the Power test** is the difference between **Power test start time** and **Power test end time**, where

- Power Test Start Time, which is the timestamp that must be taken before the execution of the first query of Stream 0.
- Power Test End Time, which is the timestamp that must be taken right after the execution of the last query of Stream 0.

Below is the pseudocode and the Python scripts for the power test.

```
Assumption:
    - The Load Test has already been executed.

Algorithm:
set timer start
    for every query in 99 generated queries:
        set timer start
        run the query
        set timer end
set timer end
```

Listing 3.9: Power test pseudocode.

```
import duckdb
import os
import fnmatch
import time
import pandas as pd
import pickle

sc = "1" # scale factor
start_time = time.time()
conn = duckdb.connect("sc_"+sc+".db")

list_time = {}
# read and run all the queries
for root, dirnames, filenames in os.walk("generated-queries/queries_0"):
        for filename in fnmatch.filter(filenames, '*.sql'):
            t = time.time()
            sql = open(root+ "/"+ filename,'r')
            sql = sql.read()

            # run all the queries
            result = conn.execute(sql)
            close = time.time()
            list_time[filename] = close-t

# Calculate the elapsed time
end_time= time.time()
elapsed_time = end_time -start_time
```

Listing 3.10: Power test Python script.

3.10 Throughput Test

The Throughput Tests measure the ability of the system to process the most queries in the least amount of time with multiple users. It measures the Database's multi-user functionalities. There are two Throughput tests during the benchmarking, Throughput Test 1 immediately follows the Power Test while Throughput Test 2 immediately follows Data Maintenance Test 1. During the test, only one query shall be active on any of the sessions at any point of time.

The **elapsed time of Throughput Test 1** is the difference between **Throughput Test 1 Start Time** and **Throughput Test 1 End Time**.

- Throughput Test 1 Start Time: timestamp taken before the first query of the first user starts.
- Throughput Test 1 End Time: timestamp taken after the last query of the last user finishes.

The elapsed time of Throughput Test 2 is the difference between **Throughput Test 2 Start Time** and **Throughput Test 2 End Time**

- Throughput Test 2 Start Time is defined as a timestamp taken before the first query of the first user starts immediately after Data Maintenance Test 1 End Time.
- Throughput Test 2 End Time, is defined as the timestamp taken after the last query of the last user finishes.

To accomplish this, We used 4 different threads having access to the database at the same time to simulate different users. Also, 4 query streams are generated to simulate different concurrent requests. Below are the pseudocode and the Python script for the throughput test.

```
Assumption:
    - 4 query streams are generated according to the specified order of
   queries defined in the documentation.

Algorithm:
    create 4 threads
    set timer start
    for each thread
        run one query stream separately
    wait for the slowest query
    set timer end
```

Listing 3.11: Throughput test pseudocode

```
import duckdb
import os
import fnmatch
import time
import threading
```

```
6
7  sc = "1" # scale factor
8  db_connection = duckdb.connect("sc_"+sc+".db")
9
10 def execute_query(query, file_name):
11     global db_connection  # Use the shared database connection
12     start_time = time.time()
13
14     cursor = db_connection.cursor()
15     cursor.execute(query)
16
17     # Calculate the elapsed time
18     end_time = time.time()
19     elapsed_time = end_time - start_time
20
21 def main():
22     TP_test_start_time_1 = time.time()
23     queries = []
24     threads = []
25
26     for root, dirnames, filenames in os.walk('data/throughput_queries'):
27         for filename in fnmatch.filter(filenames, '*.sql'):
28             sql_file = open(os.path.join(root, filename), 'r')
29             query = sql_file.read()
30             queries.append((query, filename))
31
32     # create threads and assign query stream to threads
33     for query, file_name in queries:
34         thread = threading.Thread(target=execute_query, args=(query,
    file_name))
35         threads.append(thread)
36         thread.start()
37
38     # Wait for all threads to complete
39     for thread in threads:
40         thread.join()
41
42     # calculate the time for throughput test
43     TP_test_end_time_1 = time.time()
44     TP_test_time_1 = TP_test_end_time_1 - TP_test_start_time_1
```

```
45      output = f'Throughput test time = {TP_test_time_1}\n'
46
47      with open("throughput_test/output.txt", 'w+') as f:
48          f.write(output)
49
50  if __name__ == "__main__":
51      main()
```

Listing 3.12: Throughput test Python script.

3.11 Data Maintenance Test

The Data Maintenance Tests is another benchmark that measures the ability to perform desired data changes to the TPC-DS data set. Data Maintenance Test 1 immediately follows Throughput Test 1 and Data Maintenance Test 2 immediately follows Throughput Test 2.

To achieve this purpose, refresh data sets are generated by dsdgen, and different maintenance functions are applied to the database:

- Fact insert data maintenance: new facts are inserted into the database.
- Fact delete data maintenance: some facts are deleted from the database, using date filters.
- Inventory delete data maintenance: some data is deleted from the inventory table, using date filter

The Durable Medium failure required as part of the Data Accessibility Test must be triggered during Data Maintenance Test 1 and also during the Data Maintenance Test 2.
The elapsed time of **Data Maintenance Test 1** is the difference between:

- **Data Maintenance Test 1 Start Time**, defined as the starting timestamp of the first refresh run in Data Maintenance Test 1 and
- **Data Maintenance Test 1 End Time**, defined as the ending timestamp, of the last refresh run in Data Maintenance Test 1.

The elapsed time of **Data Maintenance Test 2** is the difference between:

- **Data Maintenance Test 2 Start Time**, defined as the starting timestamp of the first refresh run in Data Maintenance Test 2 and
- **Data Maintenance Test 2 End Time**, defined as the ending timestamp, of the last refresh run in Data Maintenance Test 2.

```
Assumption:
    - Refresh dataset are generated.
    - All data maintenance functions are implemented.

Algorithm:
    set timer start
    for every refresh run:
        load the refresh dataset
        for every delete function:
            set timer start
            run the query
            set timer end
        for every insert function:
            set timer start
            run the query
            set timer start
    set timer end
```

Listing 3.13: Data maintenance test pseudocode.

Refresh Dataset Generation

During the Data Maintenance Test, the dataset is also generated by dsdgen but different from the dataset we used in previous tests.

```
dsdgen -scale 1 -dir "../../generated-data/scale_1" -delimiter "," update 1
dsdgen -scale 3 -dir "../../generated-data/scale_3" -delimiter ","  update 1
dsdgen -scale 5 -dir "../../generated-data/scale_5" -delimiter ","  update 1
dsdgen -scale 7 -dir "../../generated-data/sale_7" -delimiter ","   update 1
```

Listing 3.14: Refresh Data generation.

We use the refresh dataset to populate the schema defined in TPC-DS official tools.

Maintenance Functions

In order to test the maintenance of DuckDB, we implemented 11 functions following the TPC-DS documentation. The functions are listed in the Table 3.2.

Table 3.2: Data Maintenance Functions

ID	Function	Type	Target Table	Source Schema Tables
1	LF_CR	Insert	catalog_returns	s_catalog_returns
2	LF_CS	Insert	catalog_sales	s_catalog_order, s_catalog_lineitem
3	LF_I	Insert	inventory	s_inventory
4	LF_SR	Insert	store_returns	s_store_returns
5	LF_SS	Insert	store_sales	s_purchase, s_purchase_lineitem
6	LF_WR	Insert	web_returns	s_web_returns
7	LF_WS	Insert	web_sales	s_web_order, s_web_order_lineitem
8	DF_CS	Delete	catalog_sales, catalog_returns	-
9	DF_SS	Delete	store_sales, store_returns	-
10	DF_WS	Delete	web_sales, web_returns	-
11	DF_I	Delete	inventory	-

Below is the Python script for the data maintenance test.

```
import os
import time
import duckdb
import threading

sc = "1" # scale factor
con = duckdb.connect("sc_"+sc+".db")

# simulate the database failure for the data accessibility test
def simulate_duckdb_failure():

    # Record the start time before triggering the failure

    failure_start_time = time.time()

    # Simulate a failure event by stopping DuckDB
    print("Simulating DuckDB failure during Data Maintenance Test 1...")

    time.sleep(20)

    # Record the end time after triggering the failure
```

```
    failure_end_time = time.time()
    print(f"DuckDB failure time: {failure_end_time - failure_start_time:.2f} seconds")

# Function to run all maintenance functions in a folder
def run_sql_queries_in_folder(folder_path):
    global con
    if folder_path.endswith("2"):
        run = 2
    else:
        run = 1

    # Get a list of SQL files in the folder
    py_files = [file for file in sorted(os.listdir(folder_path)) if file.endswith(".py")]
    time_dict = {}

    # Loop through the SQL files and execute them
    for py_file in py_files:
        with open(os.path.join(folder_path, py_file), "r") as sql_script:
            t=time.time()
            py_query = sql_script.read()

            try:
                exec(py_query)
                print(f"successfully executed! {py_file} with time {time.time()-t} seconds")
                time_dict[py_file] = time.time() - t
            except Exception as e:
                print(f"problem executing {py_file}! : {e}")
                ttime = time.time()

                # Simulate DuckDB failure in a separate thread
                failure_thread = threading.Thread(target=simulate_duckdb_failure)
                failure_thread.start()

                # Wait for the failure to complete
                failure_thread.join()
                print("Durable media failure completed.")
```

```
                #Add end time
                tftime= time.time()
                time_dict["failure"] = tftime-ttime
                print(f'completed: {tftime-ttime}')
    return time_dict

# List all subdirectories in the root directory
b_time= time.time()
subdirectories = [d for d in os.listdir(refresh-data) if os.path.isdir(os.
    path.join(refresh-data, d))]

# record the result for data maintenance test
result_run_1 = run_sql_queries_in_folder("maintenance_functions")
result_run_2 = run_sql_queries_in_folder("maintenance_functions_2")
e_time= time.time()
print(f"This is the end of the Data maintenance query: {e_time-b_time}")
```

Listing 3.15: Python script for maintenance test.

3.12 Data Accessibilty Test

The Data Accessibility Test is performed by causing the failure of a Durable Media during the execution of the first Data Maintenance Test. In our benchmarking Project, we caused an exception by introducing a file with wrong syntax and errors so as to simulate the data failure process.
The Data Accessibility Test is successful if all in-flight data maintenance functions, subsequent queries and data maintenance functions complete successfully after the above durable media have failed.

3.13 Scalibility Test

Scalability Testing measures performance of a system or network when the number of user requests are scaled up or down. The purpose of Scalability testing is to ensure that the system can handle projected increase in user traffic, data volume, transaction counts frequency, etc. It tests the system's ability to meet the growing needs. For our Project we wanted to know how the

database can scale up when the user increases and at what point will the system become unresponsive. To achieve this, we had to implement the TPC-DS dataset and DuckDB database on Apache Jmeter

Using Apache JMeter

To test your database, you need to

- Create a test plan
- Execute the test plan

Create a Test Plan

A basic test plan consists of the following elements:

- Download DuckDB Connector
- Thread Group
- JDBC Connection Configuration
- JDBC Request
- JDBC Listener

JMeter has a default test plan, which you can use if you don't want to create multiple test plans. However, there's no elements in the default test plan, so we need to add the missing elements. Also, add a jar path in the library to the duckdb_jdbc jar file you downloaded.

DuckDB JDBC Connector

We use the DuckDB database to create our Database. First, you have to connect the DuckDB database with JMeter using the DuckDB JDBC connector. First, download the duckdb_jdbc-0.9.1.jar

from the Maven website[5]. Copy “duckdb_jdbc-0.9.1.jar” file (Version may differ) from the folder and paste it into the “lib” folder of JMeter and Restart the JMeter so that it will work with the library provided by DuckDB connector.

Create Thread Group

The Thread Group is a set of threads that performs a test scenario. In this screen, we can set the number of users and other similar settings to simulate the user requests. We right-click on the TestPlan and then we select the Add->Threads (Users)->Thread Group

Add JDBC Connection Configuration

This is used to create a valid database connection. You add JDBC Connection Configuration by following the steps : "Add > Config Element > JDBC Connection Configuration"
Then configure your DuckDB Database with this:
Database URL: jdbc:duckdb:/home/pce/Pictures/duckdb_terminal/scale_1.db
where "/home/pce/Pictures/duckdb_terminal"- directory of where the DB is created and scale_1.db is the name of the DB
Database Class: org.duckdb.DuckDBDriver

JDBC Request

JDBC request element helps to define a SQL query that will be executed by the test user(s). To work with data from the database, You need to Add JDBC Request. So add JDBC Request By Clicking Add > Sampler > JDBC Request we set the Variable Name of Pool declared in JDBC Connection Configuration parameter. This parameter value has to be same as the JDBC connection pool name

JDBC Listener

Before starting our test, we need a Listener that helps to monitor and analyze the result of the test. For this test, we will use the View Results Tree, of listeners. We right-click on the JDBC Request

[5]https://repo1.maven.org/maven2/org/duckdb/duckdb_jdbc/0.9.1/

and select Add->Listener->View Results Tree to monitor the detailed result of the test. At the same time, we add a Summary Report that helps to monitor a summarized result of the test. To add a summary report, we right-click on the JDBC Request and select Add->Listener->Summary Report to monitor the detailed result of the test.

Test your database performance by clicking the **Run button on the menu bar**.

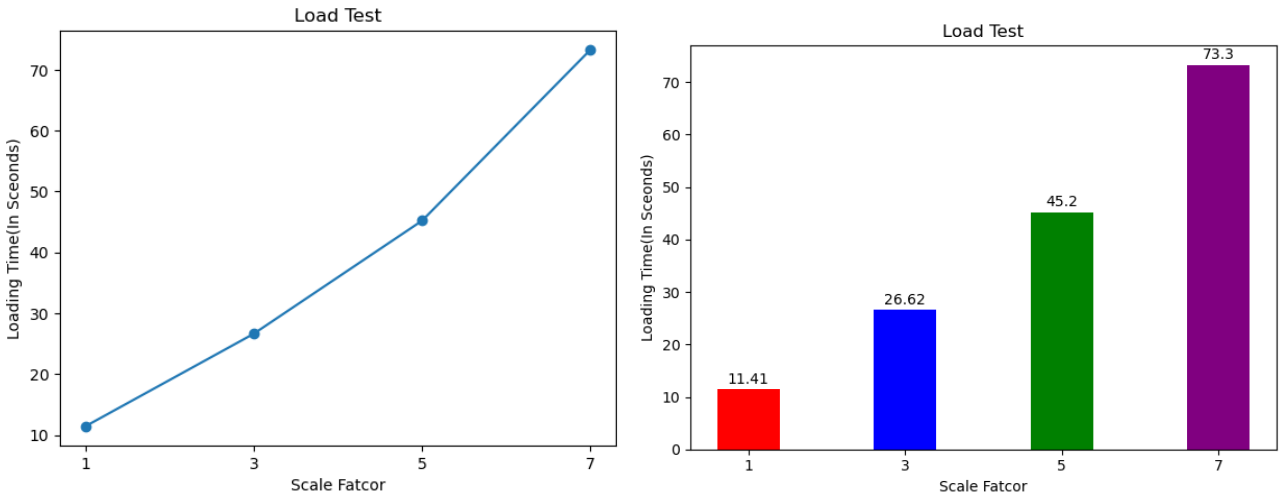
4 Results and Discussion

As mentioned in the previous chapter, the benchmark was performed on a local machine, with a total of four different scale factors (1, 3, 5 and 7) and our scale factors increased linearly. In this chapter, we will be focusing on a few different criteria, namely, discussing the test result, the evolution of the run times as the scale factor of the database increases and making comparison between optimized and original query performances.

.

4.1 Load Test

We ran the load test script for 4 scale factors and show the results in Figure 4.1. The charts provided the loading time for 4 scale factors. The bar chart presents the information about the Load Test Time in seconds for four scale factors. As can be seen from the chart there is a significant difference between Load Test Time for each scale factor. The time increases from 11.41 seconds for SF 1 factor up to 73.3 seconds for SF 7. The Load Test indicator increased by 7 times. This is reasonable considering the data size of 7 GB and 1 GB. The line chart also shows that the loading time increased almost linearly as the data was increased from 1GB to 7GB.

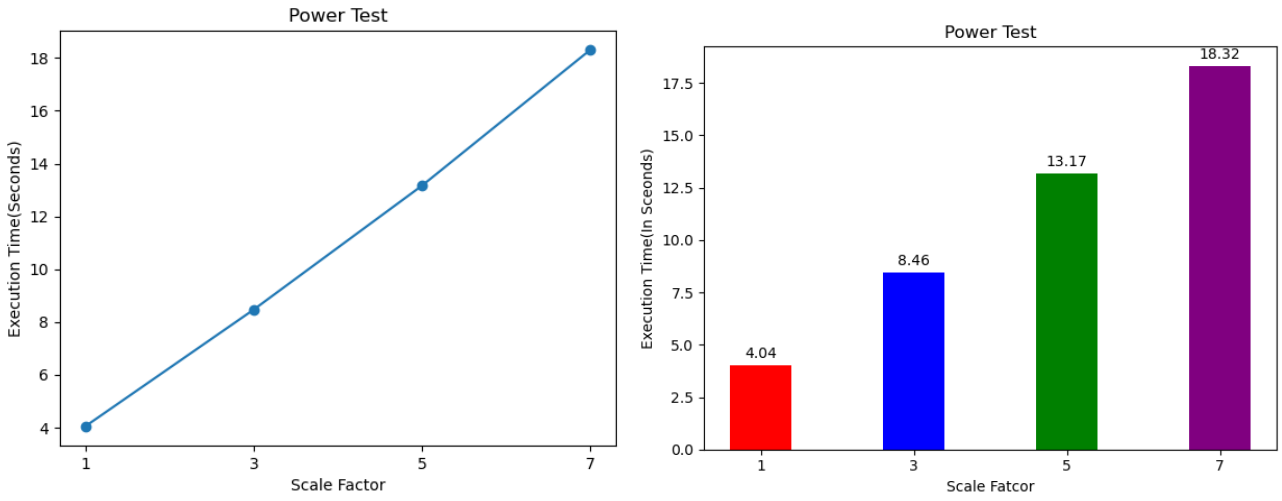


(a) Line chart (b) Bar chart

Figure 4.1: Load test results.

4.2 Power Test

The results of the Power test can be found in Figure 4.2. We can see the Power Time for all scale factors in the figure. From the bar chart,we can find the time increasing from 4.04 seconds for SF-1 , 8.46 seconds for SF-3 to 18.32 seconds for SF-7.From the line chart, we found the increase was really close to be linear, meaning that the system was scaling nearly optimally in terms of query processing .



(a) Line chart (b) Bar chart

Figure 4.2: Power test results.

4.2.1 Load and Power Test Comparison

It can observed that Power and Load Test are close to being linear. But critically looking it at from the fig 4.3 and 4.4 below, We can clearly see how the load test presents a higher increase than the power test. DuckDB was able to process the query faster because it is an analytical database optimized for read operations and even faster when processed with good hardware, in our case Apple Macbook Pro.

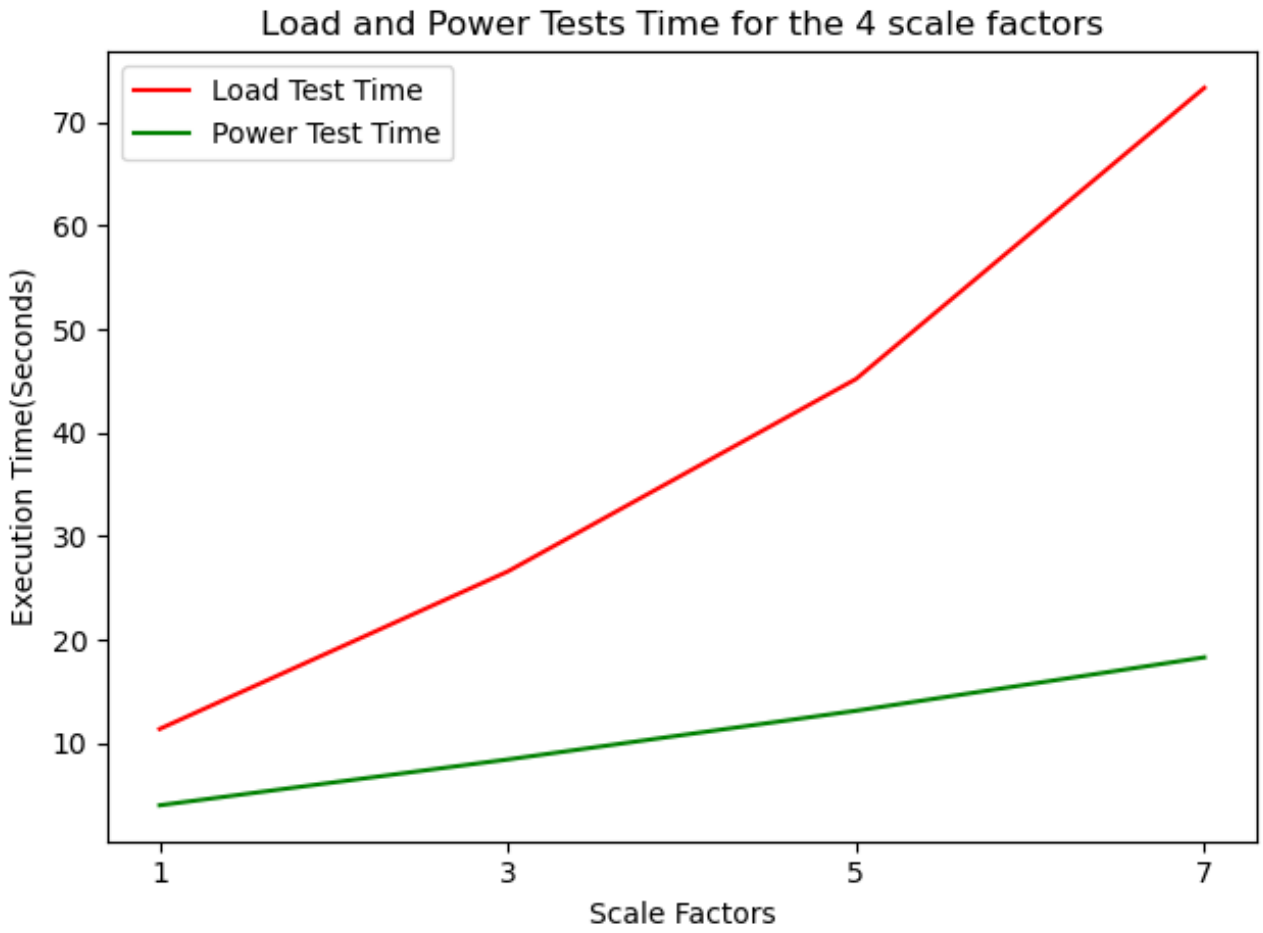


Figure 4.3: Load test and Power Test comparison(a).

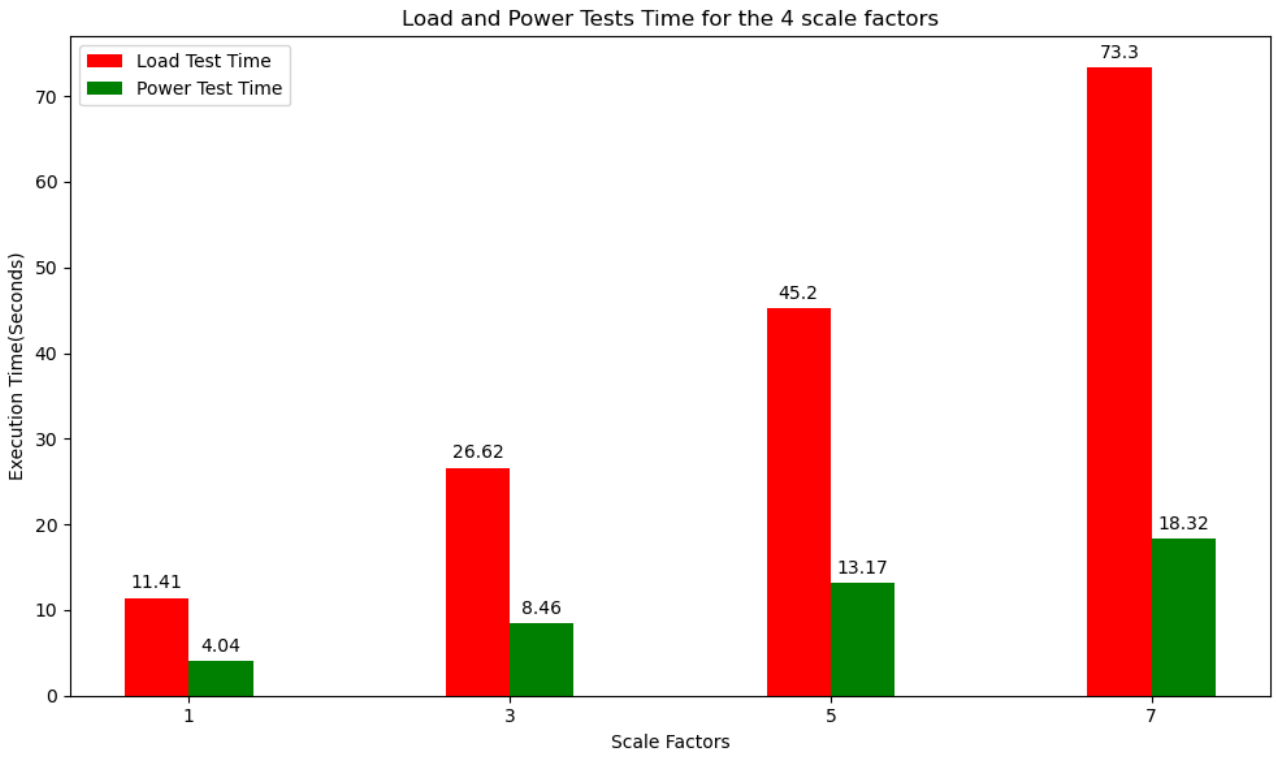


Figure 4.4: Load test and Power Test comparison(b).

4.2.2 Individual Scale Execution

We analyzed the execution times for each query and each scale factor. The power test results for 4 scales are shown in Figures 4.5, 4.8, 4.11, and 4.14. Figure 4.17 compares execution time for the first 10 queries over 4 scale factors. Some queries with large running time are 23, 23b, 67, and 95. Besides, the query running time increases when the data size is bigger.

Scale 1

In Figure 4.5, it showed how much time it took to run the 102 queries on scale factor 1. Query 23 is the longest outlier with around 0.19935 seconds of execution times. Query 4,14,14b,22,23,23b,47,57,64, 67,69,95 are longer than other queries.The average times is about 0.03949 seconds for all the queries. So in Figure 4.7, we plotted the execution times of all the queries except for query 4,14,14b,22,23,23b, 47,57,64,67,69,95 to see their values more clearly. Query 11 is another longest outlier with around 0.10838 seconds.And the average times is about 0.02603 seconds after removing the queries.

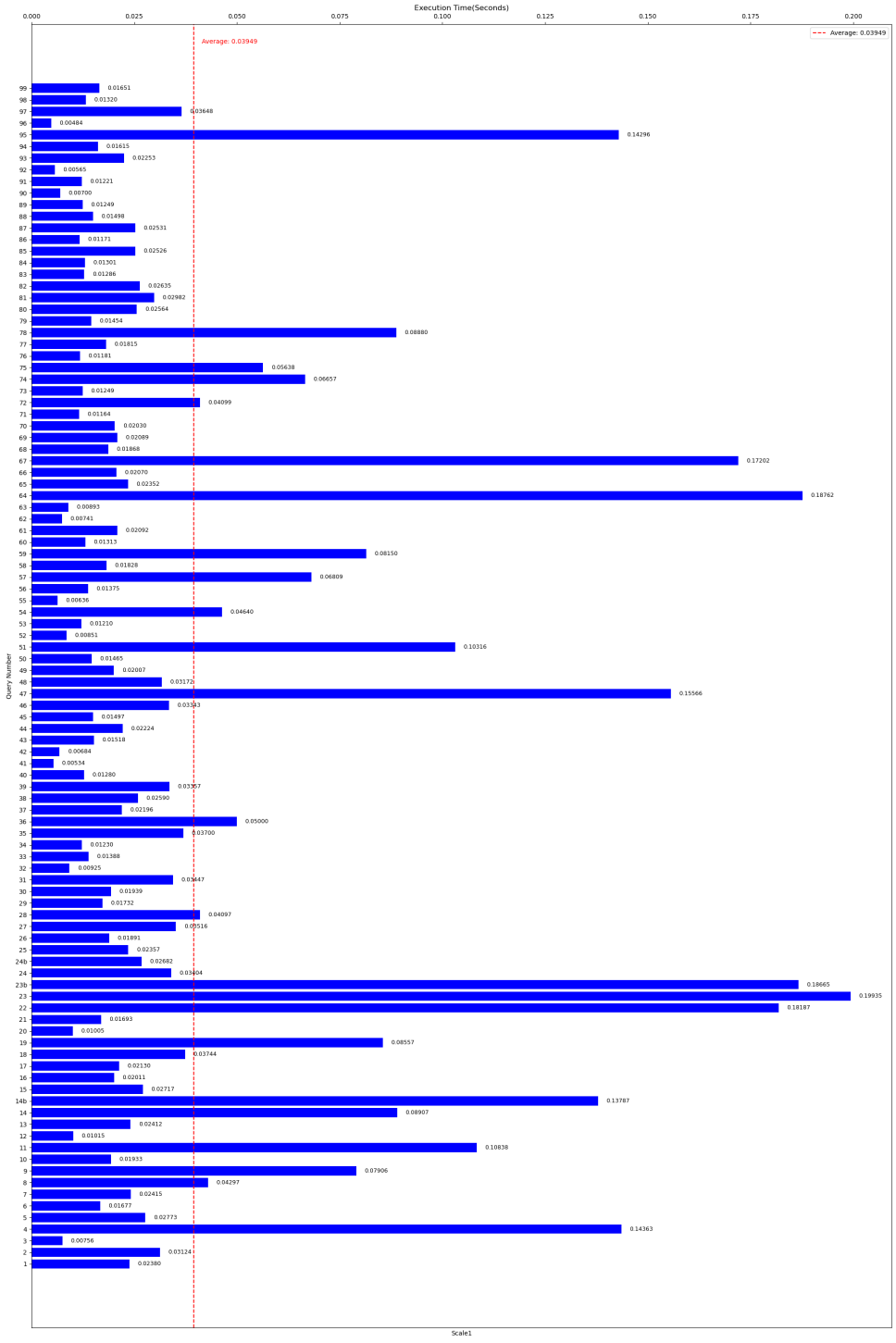


Figure 4.5: Power test result for scale factor 1.

Figure 4.6: Deleted Queries for scale factor 1.

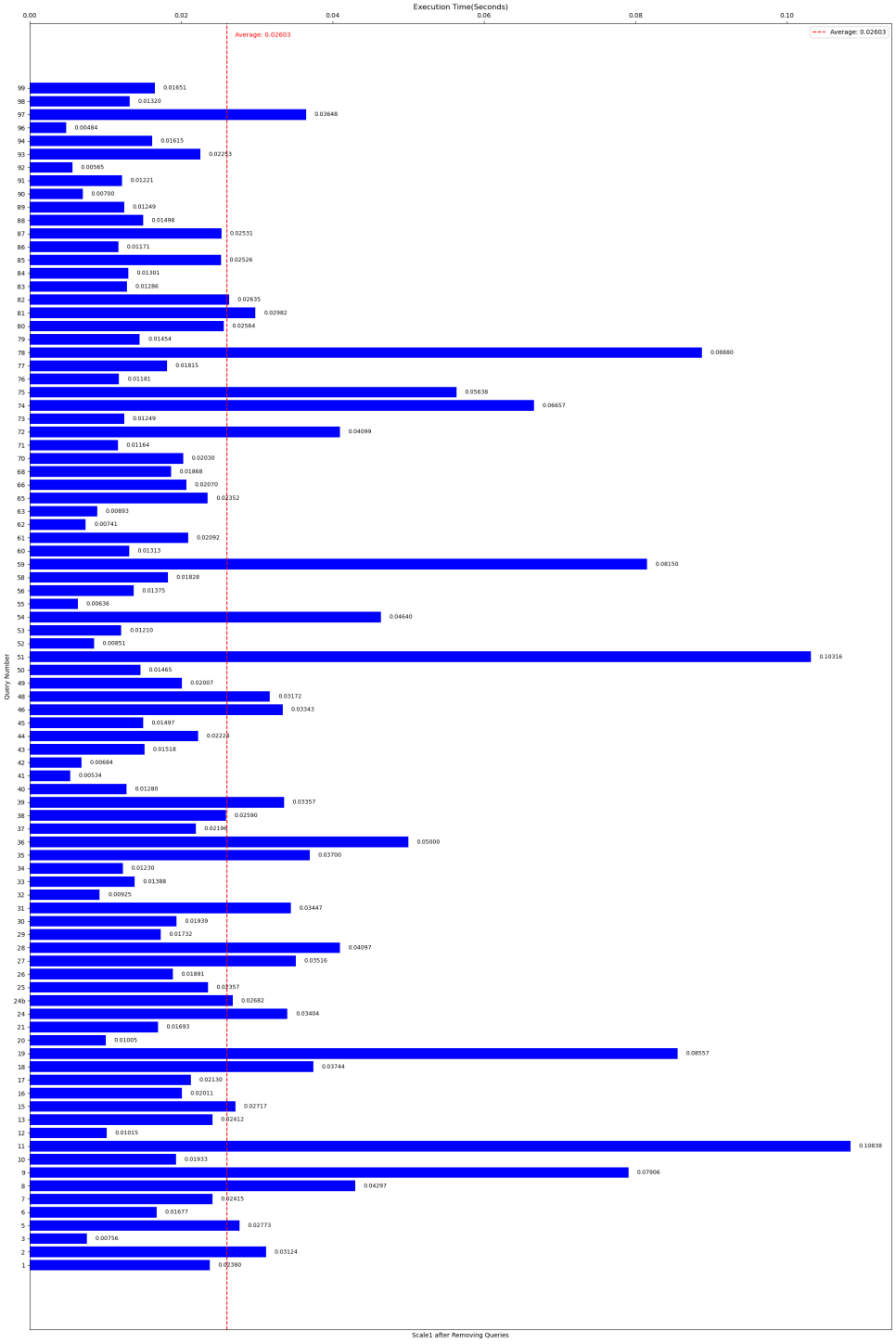


Figure 4.7: After Deleting Queries for scale factor 1.

Scale 3

In Figure 4.5, it showed how much time it took to run the 102 queries on scale factor 3. Query 23b is the longest outlier with around 0.52983 seconds of execution times. Query 4,14,14b,22,23,23b,47,57, 64,67,69,95 are longer than other queries. The average times is about 0.08282 seconds for all the queries. So in Figure 4.10, we plotted the execution times of all the queries except for query 4,14,14b,22,23,23b,47,57,64,67,69,95 to see their values more clearly. Query 19 is another longest outlier with around 0.26662 seconds.And the average times is about 0.05186 seconds after removing the queries.

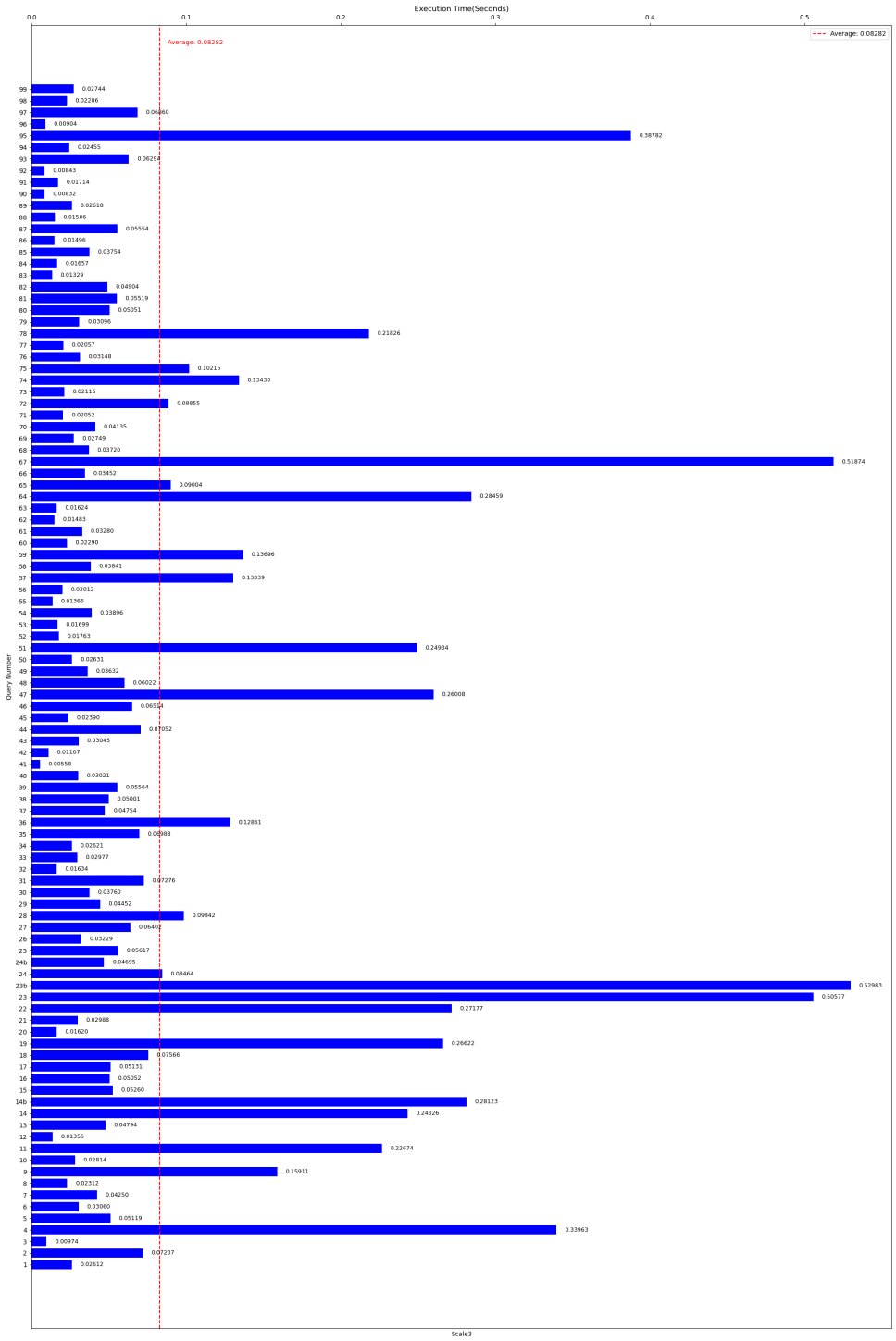


Figure 4.8: Power test result for scale factor 3.

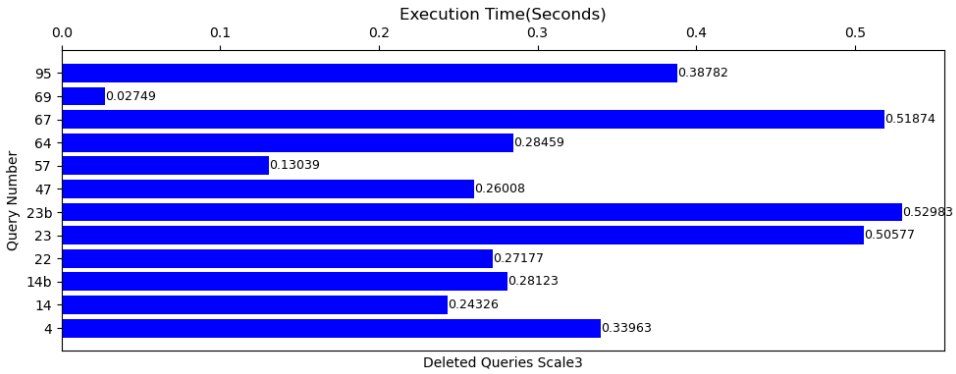


Figure 4.9: Deleted Queries for scale factor 3.

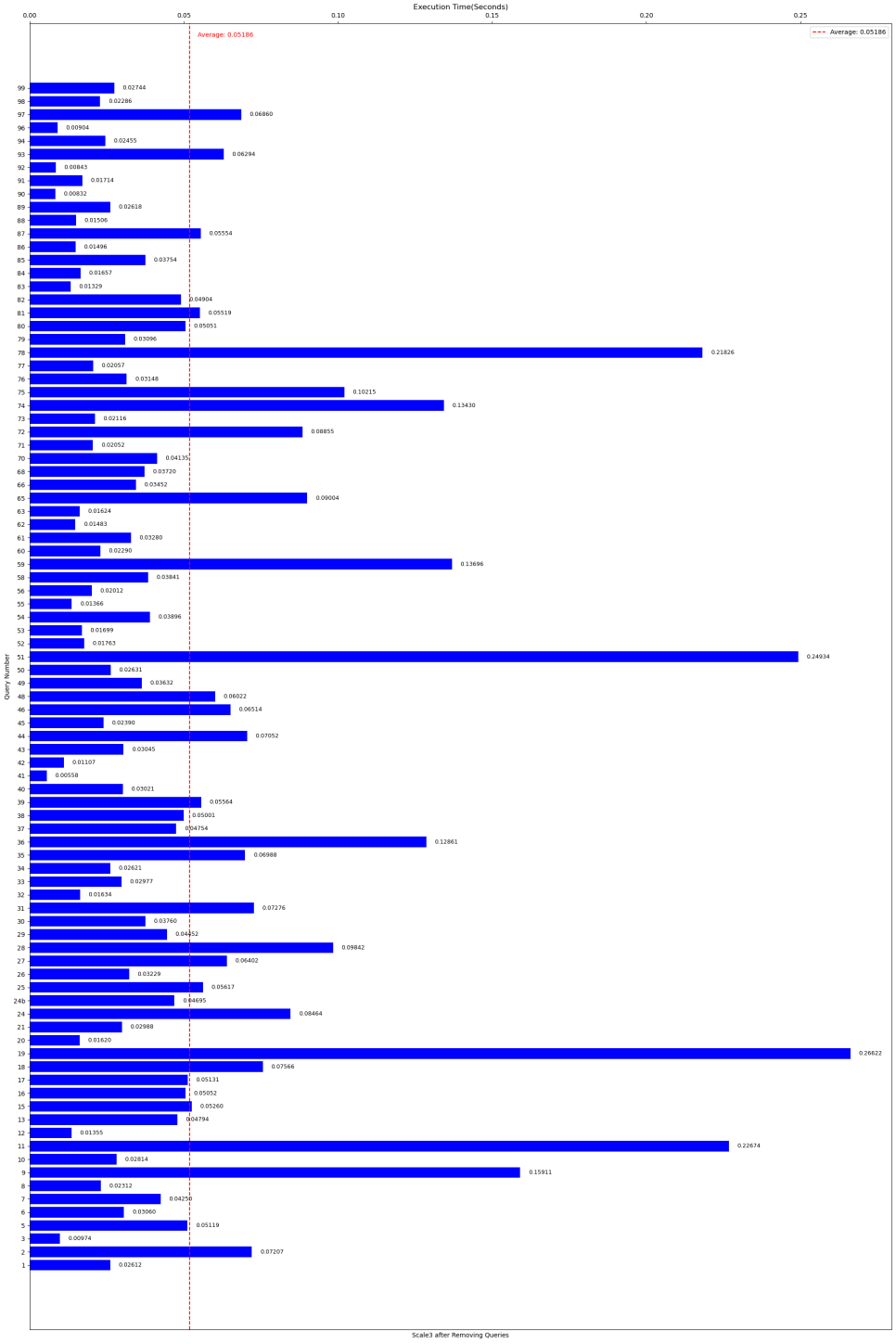


Figure 4.10: After Deleting Queries for scale factor 3.

Scale 5

In Figure 4.11, it showed how much time it took to run the 102 queries on scale factor 5. Query 95 is the longest outlier with around 1.12474 seconds of execution times. Query 4,14,14b,22,23,23b,47,57,64, 67,69,95 are longer than other queries.The average times is about 0.12894 seconds for all the queries. So in Figure 4.13, we plotted the execution times of all the queries except for query 4,14,14b,22,23,23b,47,57, 64,67,69,95 to see their values more clearly. Query 51 is another longest outlier with around 0.39623 seconds.And the average times is about 0.07611 seconds after removing the queries.

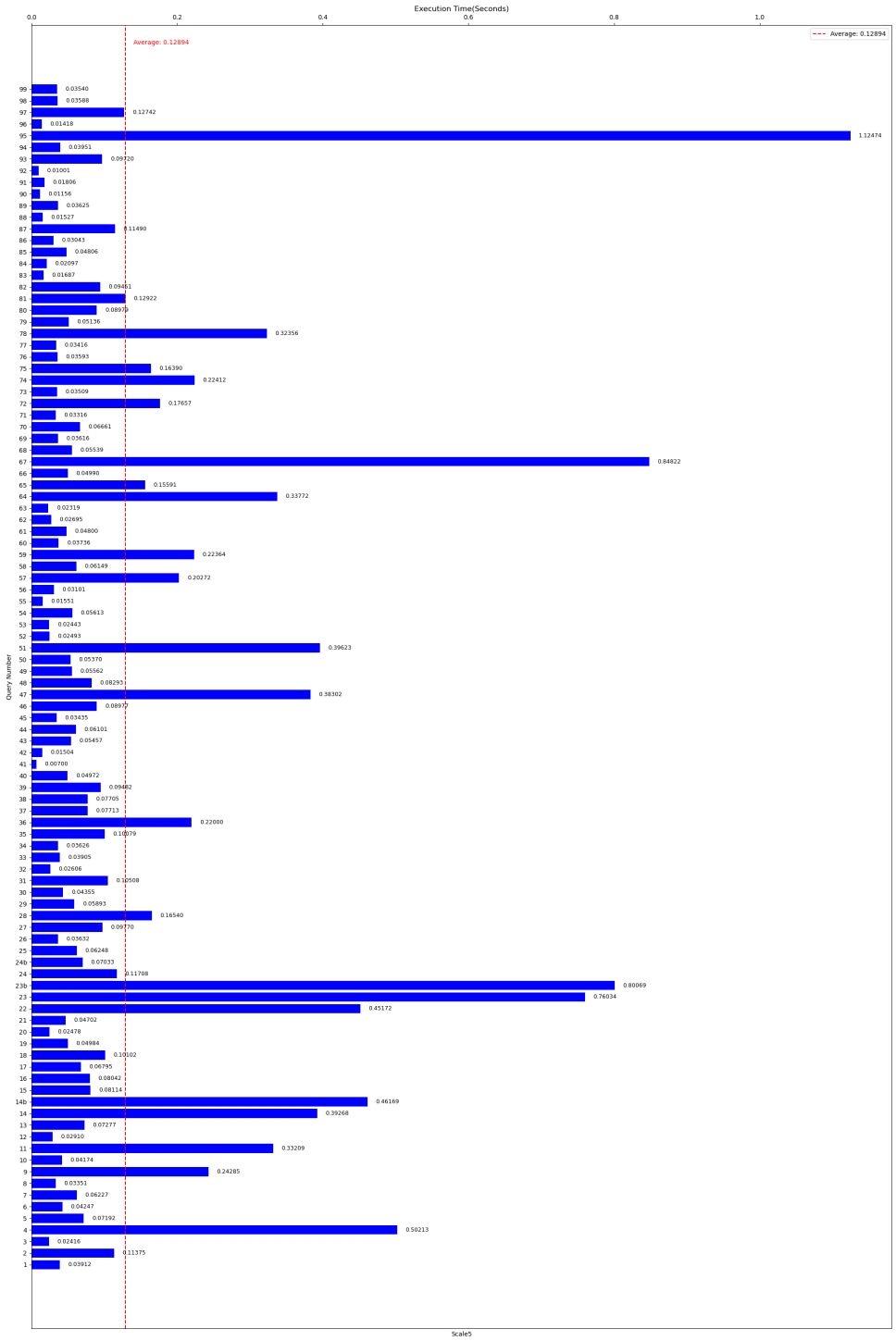


Figure 4.11: Power test result for scale factor 5.

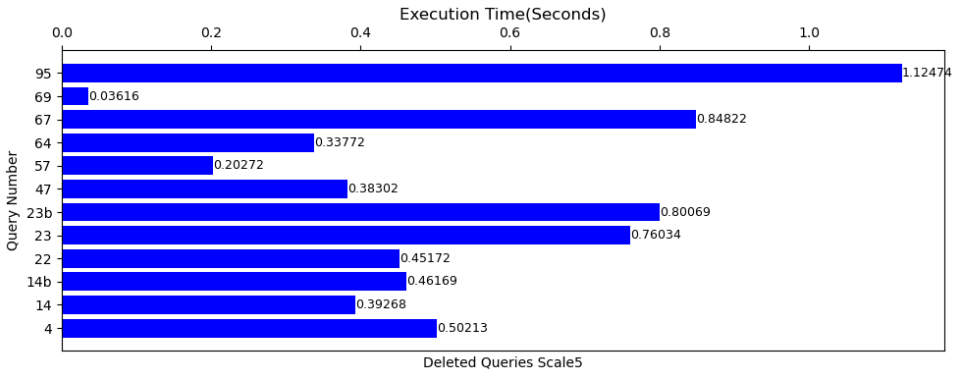


Figure 4.12: Deleted Queries for scale factor 5.

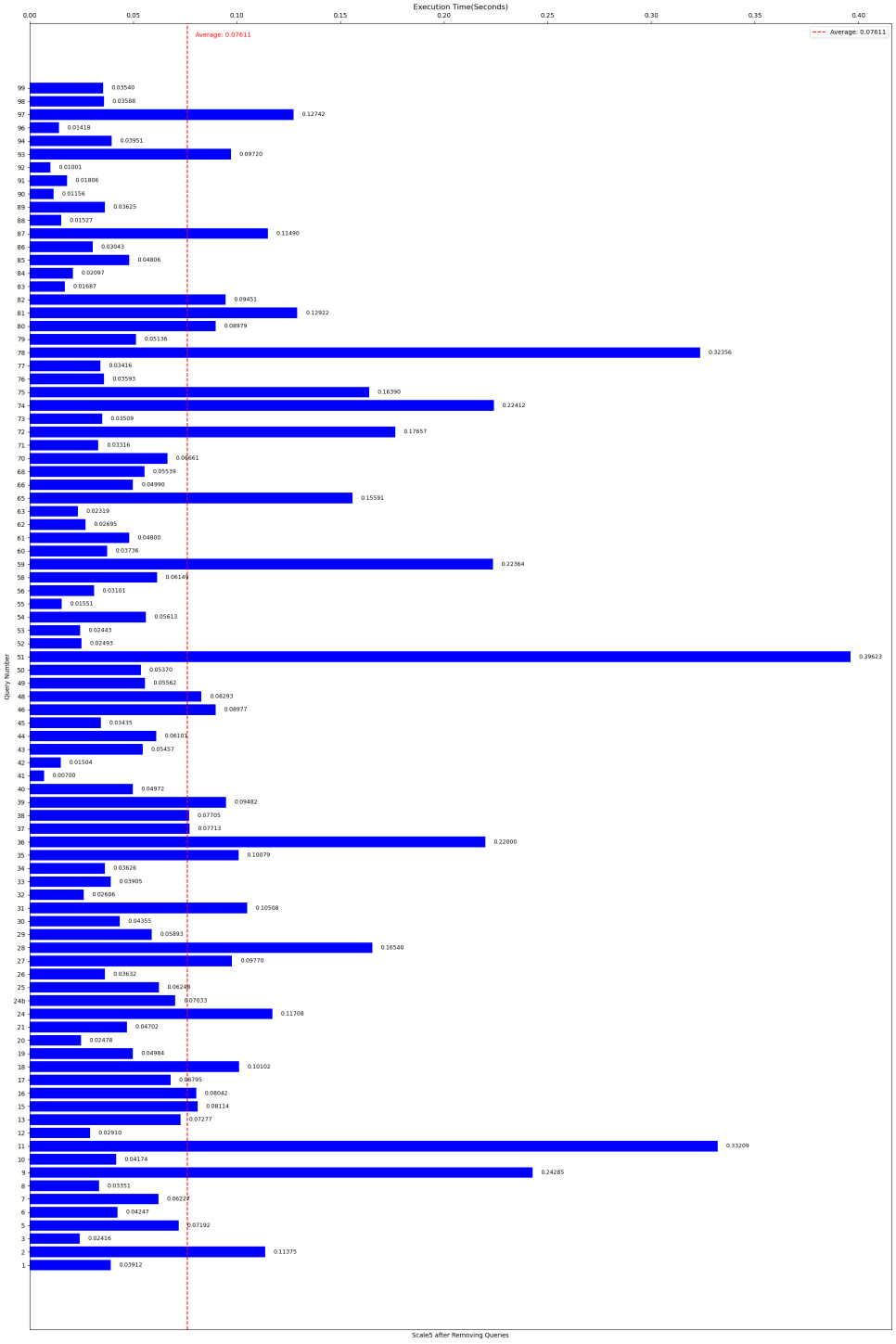


Figure 4.13: After Deleting Queries for scale factor 5.

Scale 7

In Figure 4.14, it showed how much time it took to run the 102 queries on scale factor 7. Query 95 is the longest outlier with around 1.77944 seconds of execution times. Query 4,14,14b,22,23,23b,47,57, 64,67,69,95 are longer than other queries. The average times is about 0.12894 seconds for all the queries. So in Figure 4.16, we plotted the execution times of all the queries except for query 4,14,14b,22,23,23b,47,57,64,67,69,95 to see their values more clearly. Query 51 is another longest outlier with around 0.52974 seconds. And the average times is about 0.09893 seconds after removing the queries.

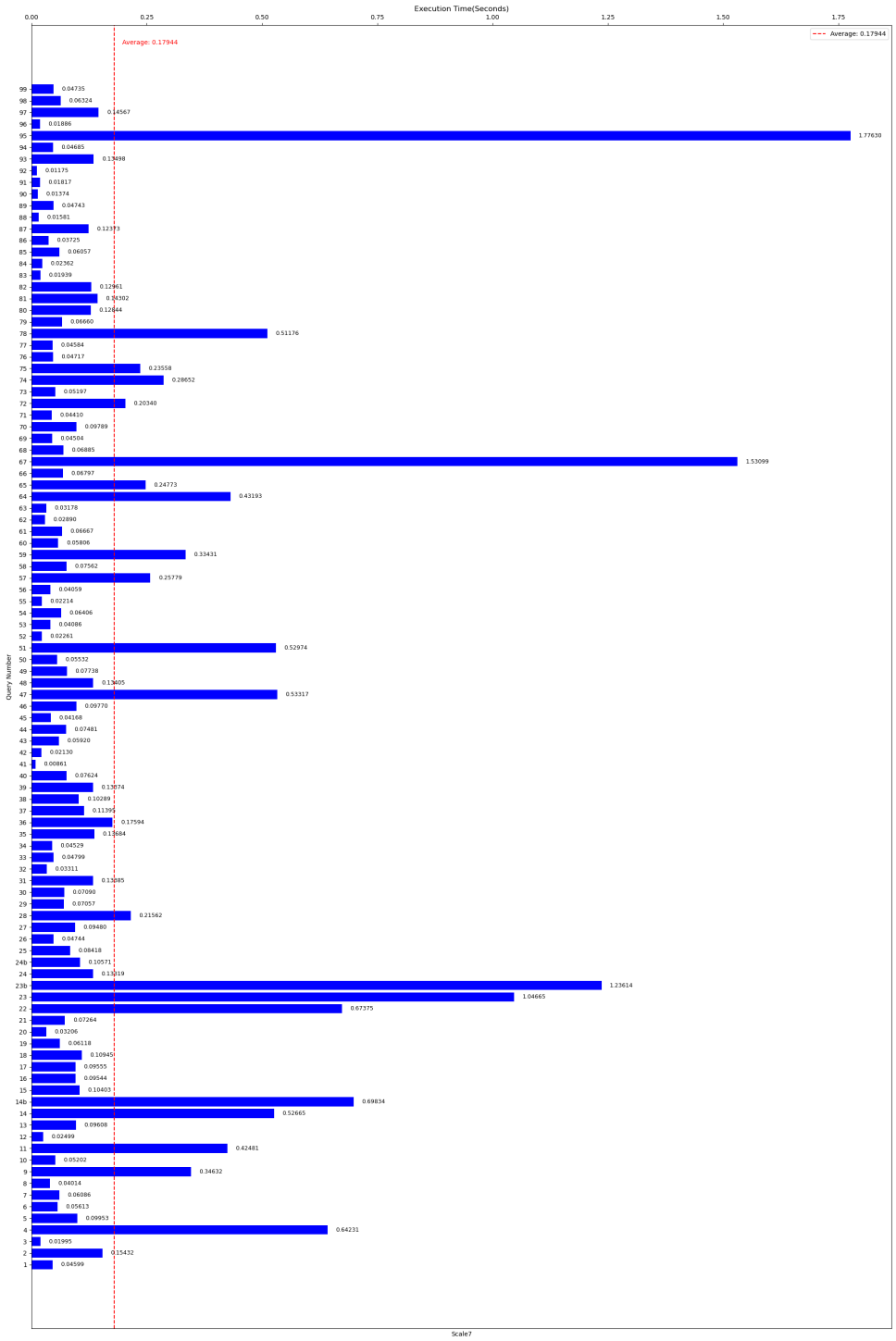


Figure 4.14: Power test result for scale factor 7.

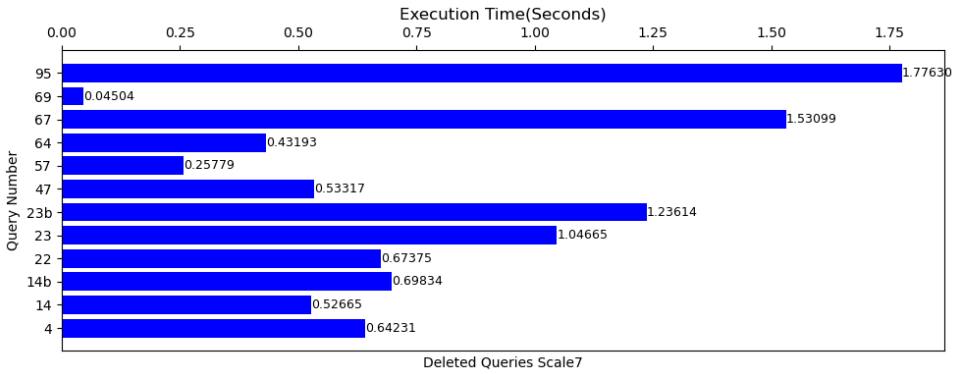


Figure 4.15: Deleted Queries for scale factor 7.

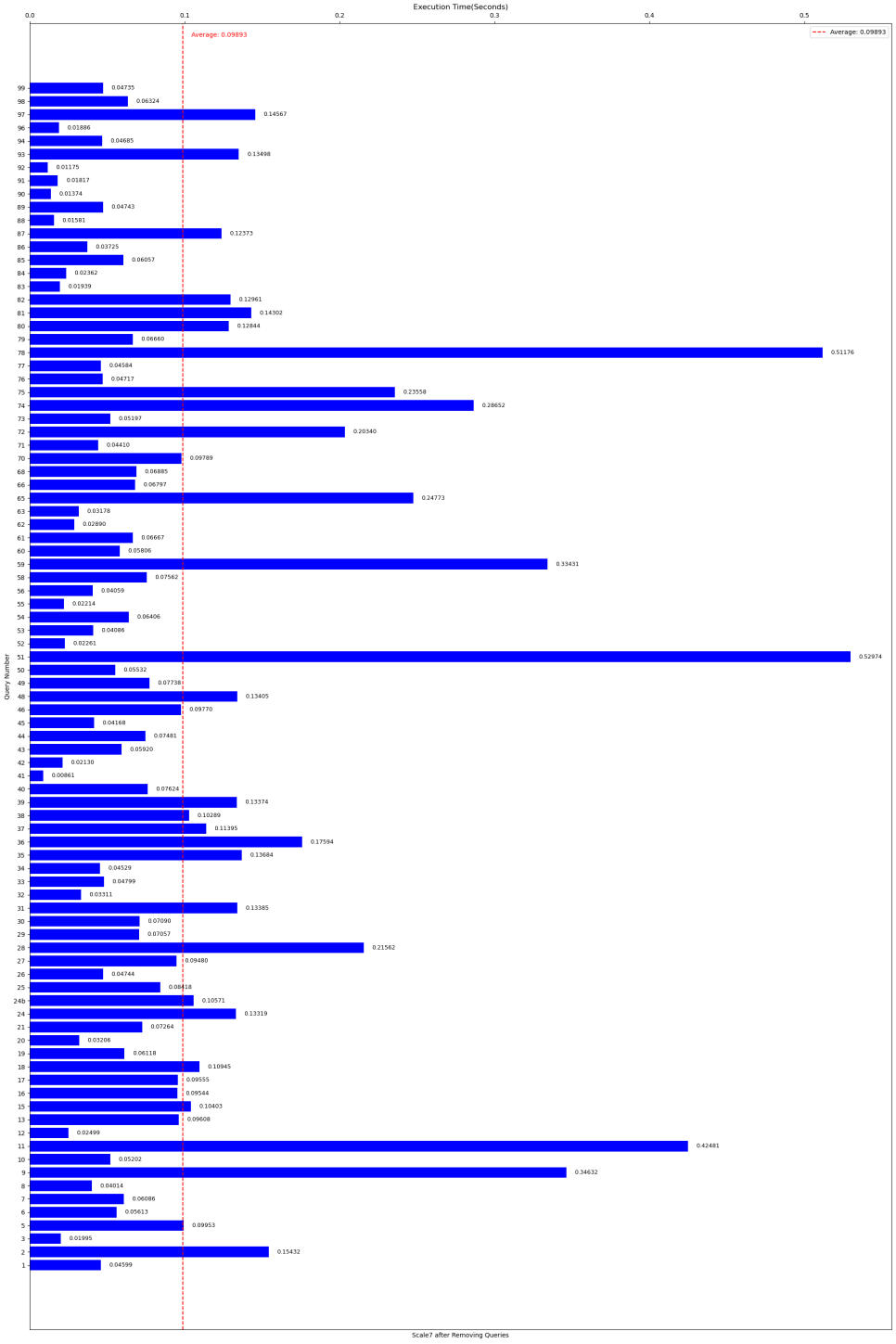


Figure 4.16: After Deleting Queries for scale factor 7.

4.2.3 Execution Times in Different Scale Factors

To better demonstrate the relationship between execution times and scale factors, we chose to draw a comparative bar chart of the first ten queries generated. As can be seen from the figure, the execution time of each query grows along with the size factor. And it is worth noting that this growth is close to linear. This is also very much in line with our expectations. Because We thought to choose the scale factors to be 1, 3, 5 and 7 and our scale factors increase linearly. We tried to plotted additional bar graphs for the other groups, and almost all of them showed this linear increase.

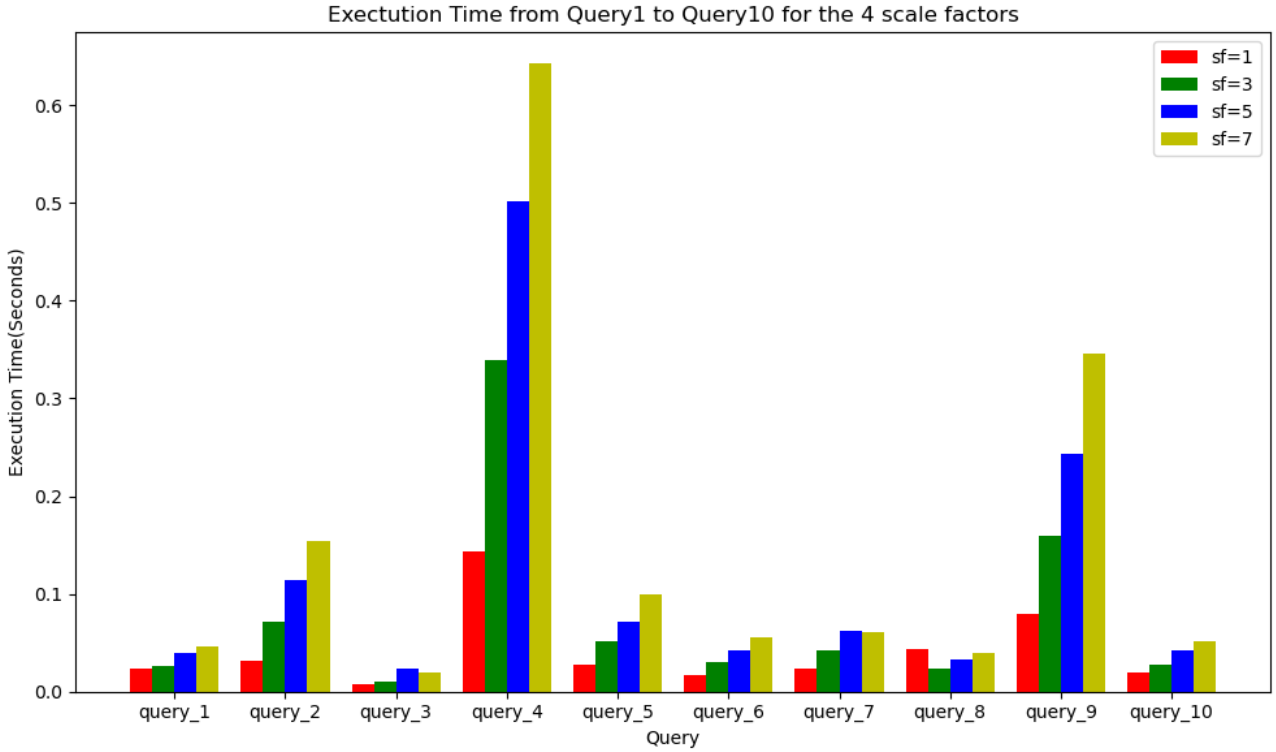


Figure 4.17: Comparison of power test results for the first 10 queries over 4 scale factors.

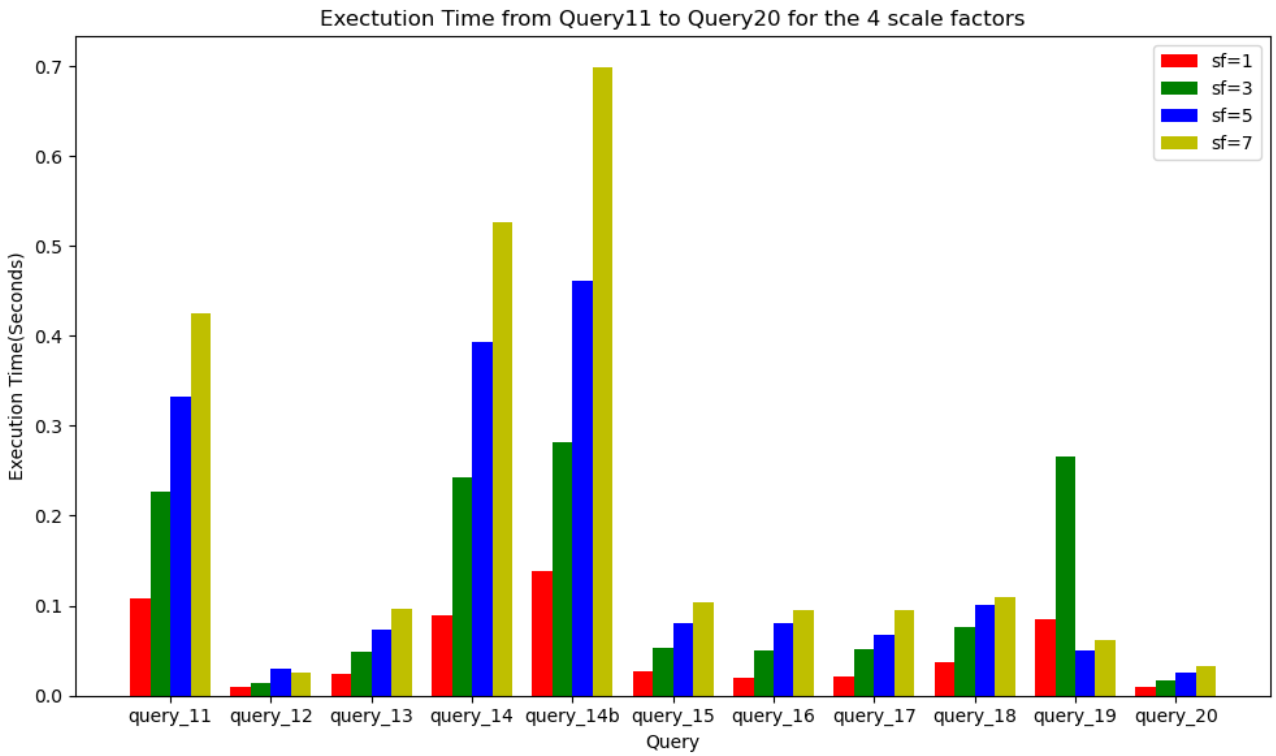


Figure 4.18: Comparison of power test results for the queries 10 to 20 over 4 scale factors.

In addition to this, we encountered a number of queries that were different from the linear growth case. These outliers represent roughly 10% of the overall queries.

We thought the performance of this test does not necessarily increase linearly for several possible reasons:

- Complexity of Queries: TPC-DS queries have varying degrees of complexity. Some queries may involve a lot of data processing and computation, while others may be relatively simple. This means that the execution time may vary significantly from one query to another.
- Resource Competition: In a database system, various resources such as CPU, memory, and storage can become bottlenecks. Some queries may place a higher demand on a particular resource, while other resources may be relatively idle.
- Data and Index Layout: The physical layout of the data and index structure can greatly affect query performance. For example, for a large number of table scanning operations, good data localization can improve performance.
- System Concurrency: Although Power Test evaluates the performance of a single query, there

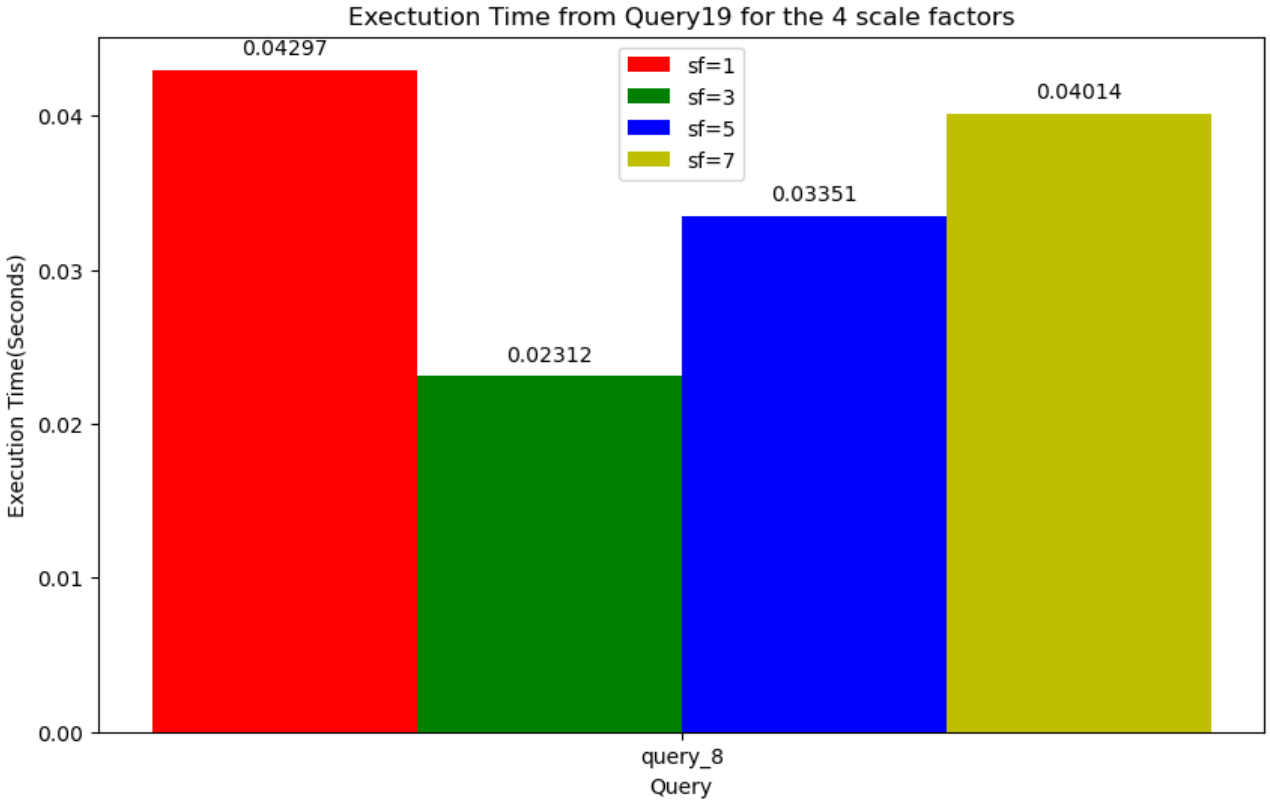


Figure 4.19: Query 8

may be other background tasks and operations in the system that may affect query performance.

In summary, due to these and other possible factors, the performance of TPC-DS Power Test is not guaranteed to be linear. Different queries, data layouts, system configurations, and resource constraints can all lead to non-linear variations in performance.

4.3 Throughput Test

Figure 4.21 below shows the results for our implementation of the Throughput Test. During the throughput testing phase, we executed two different refresh run twice.The throughput test 1 was done after the power test while The throughput test 2 was after data maintenance test. Figure 4.21 compares the execution time of throughput 1 and 2 over 4 scale factors. The results of both throughput tests were very close and both showed linear growth. This might indicate that the system is good in multi-user scenarios. Also, after performing the data maintenance test 1, we observed the throughput test maintained same linear trend, showing that even during deletion

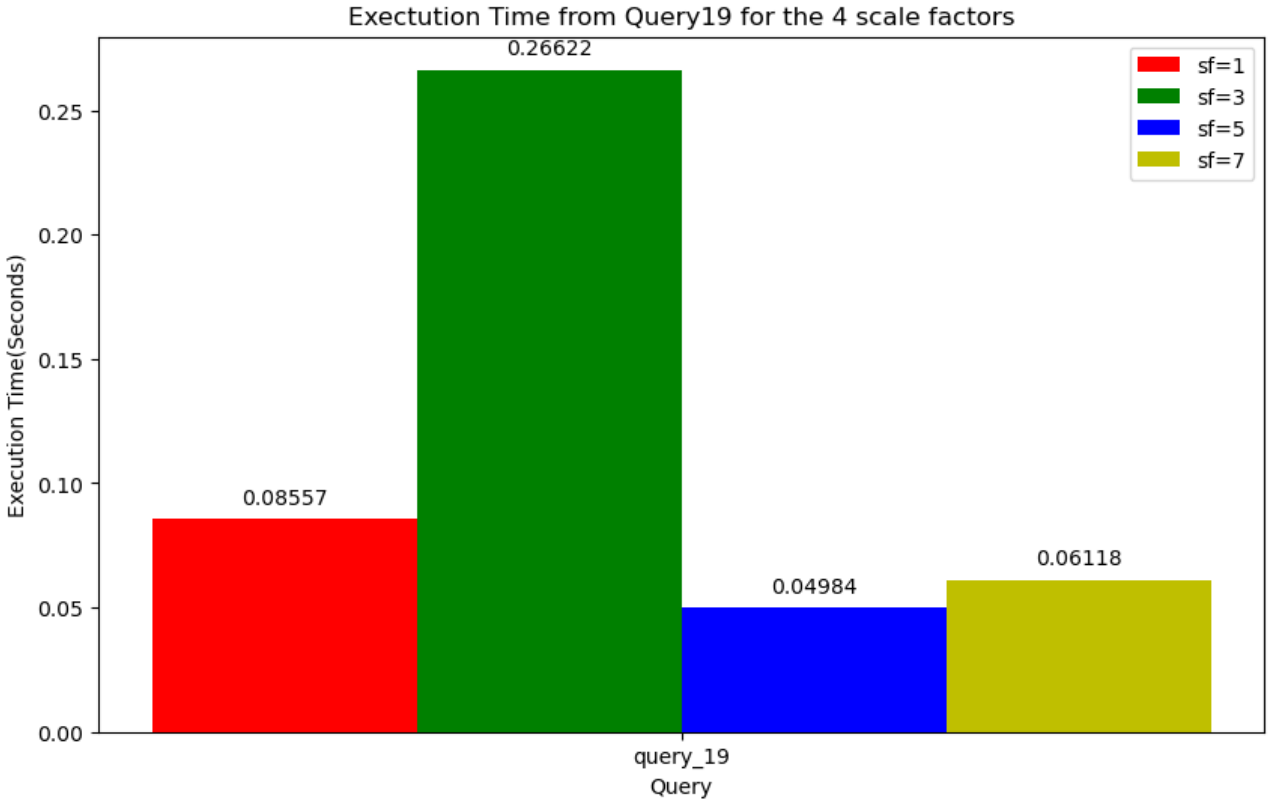
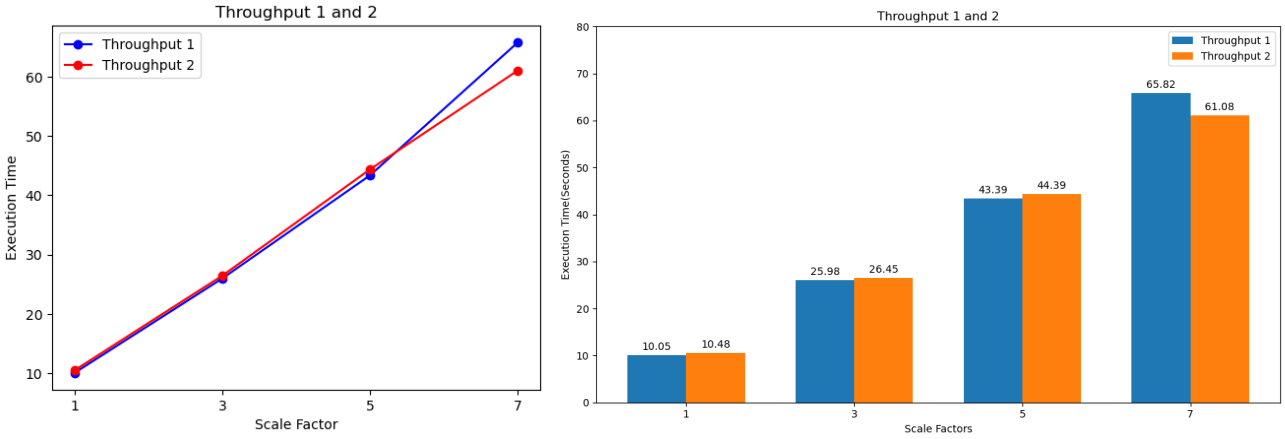


Figure 4.20: Query 19

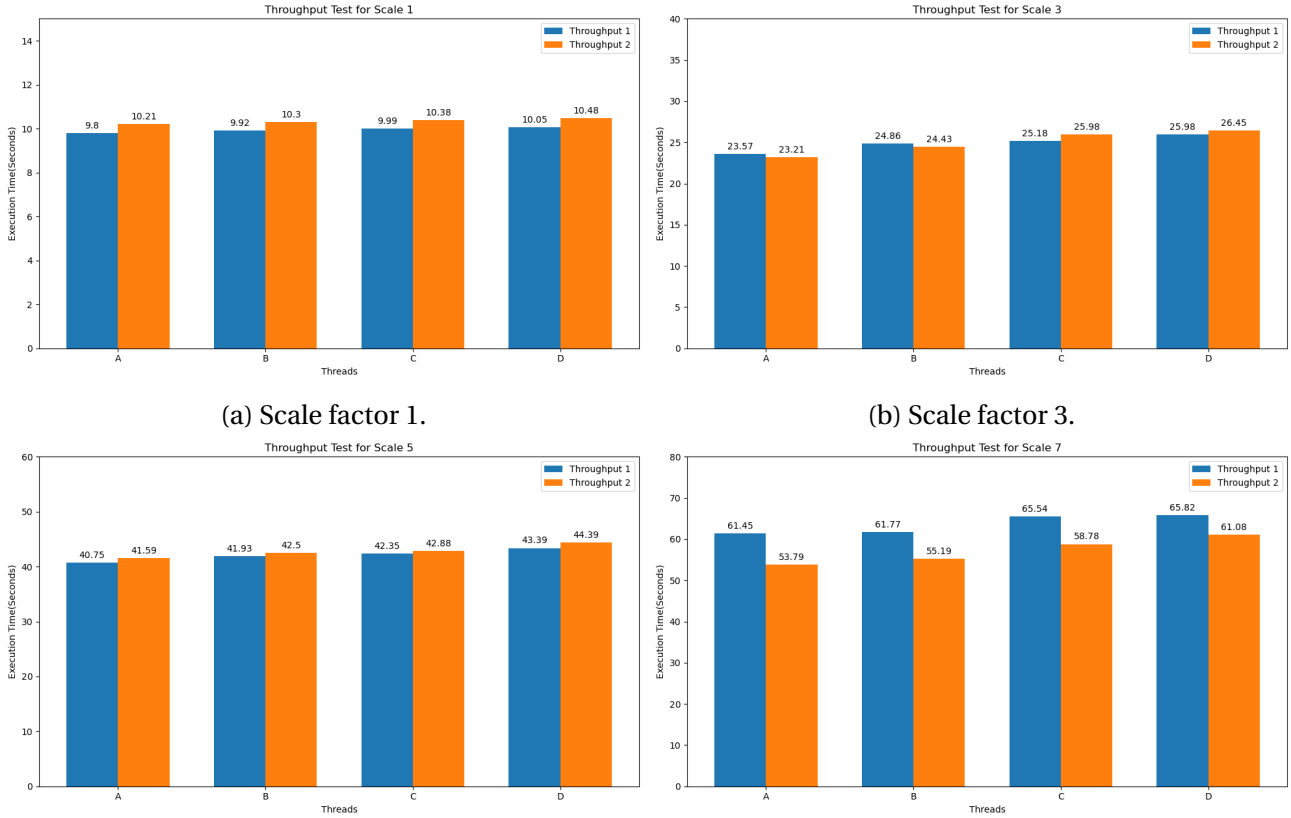
and insertion, the system can still perform well. We also used 4 different threads having access to the database at the same time to simulate different 4 users. We had named it 'A','B','C','D'.Figure 4.22 shows the throughput test results in 4 scale factors.



(a) Line chart

(b) Bar chart

Figure 4.21: Throughput 1 and 2.



(a) Scale factor 1.

(b) Scale factor 3.

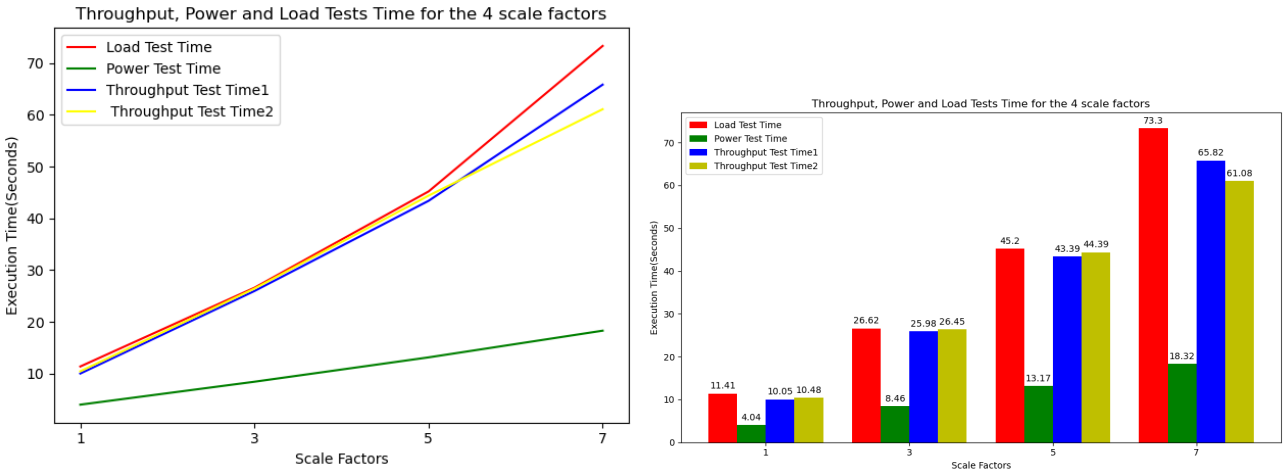
(c) Scale factor 5.

(d) Scale factor 7.

Figure 4.22: Throughput test results.

4.3.1 Comparison for Throughput, Power and Load Tests

In Figure 4.23, We examined the execution times for two Throughput Tests and compared them to the Power and Load Tests. At scale factor 1, the duration of the Throughput Test was about twice that of the Power Test, as anticipated. For higher scale factors, however, the Throughput Test's execution time increased more than the Power Test's. This discrepancy is evident in the bar chart, showing a duration increase that's over double but under four times. Such a trend implies our system not only excels in sequential run efficiency but also effectively manages the Throughput Test.We thought the efficiency of multi-user operations needed to be improved.



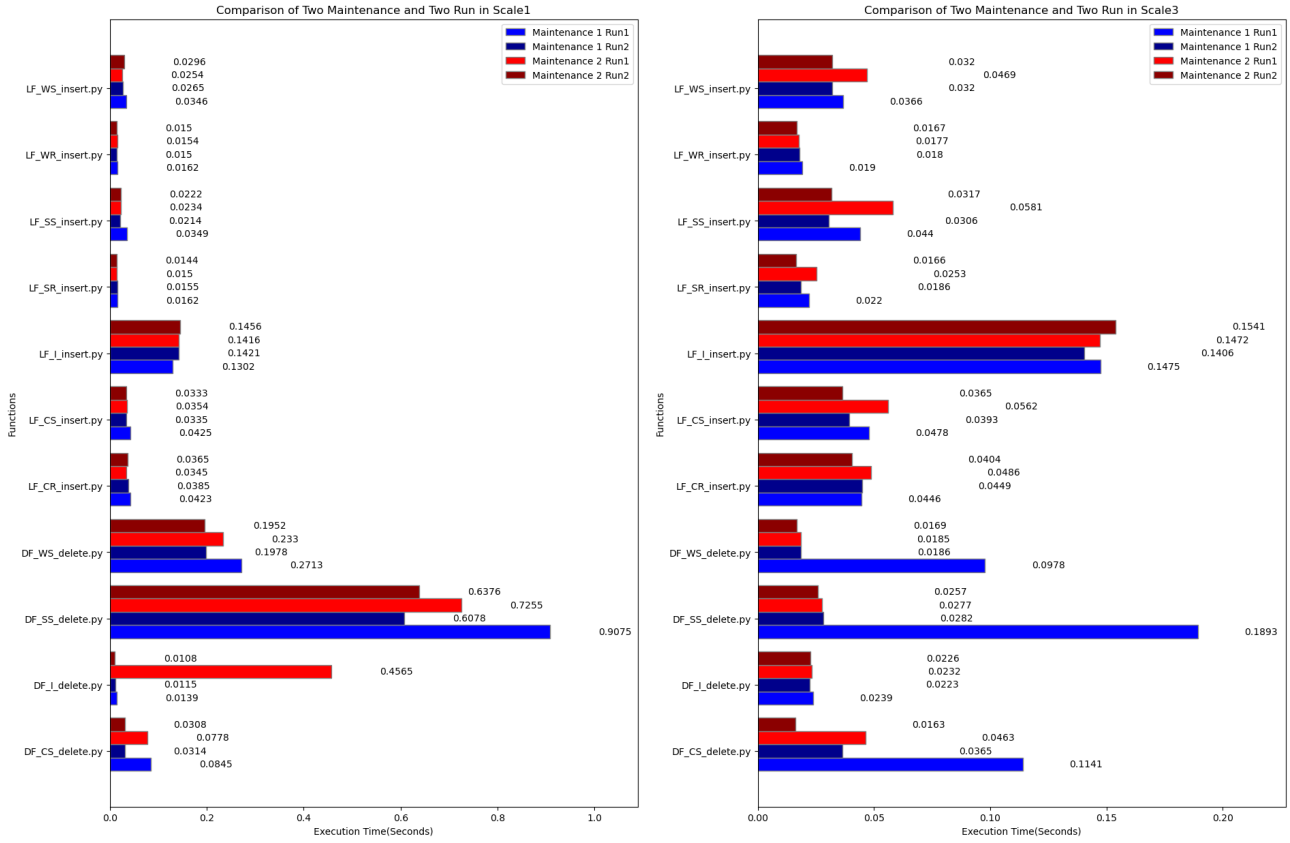
(a) Line Chart

(b) Bar Chart

Figure 4.23: Comparison for Throughput, Power and Load Tests Time for the 4 Scale Factors

4.4 Data Maintenance Test

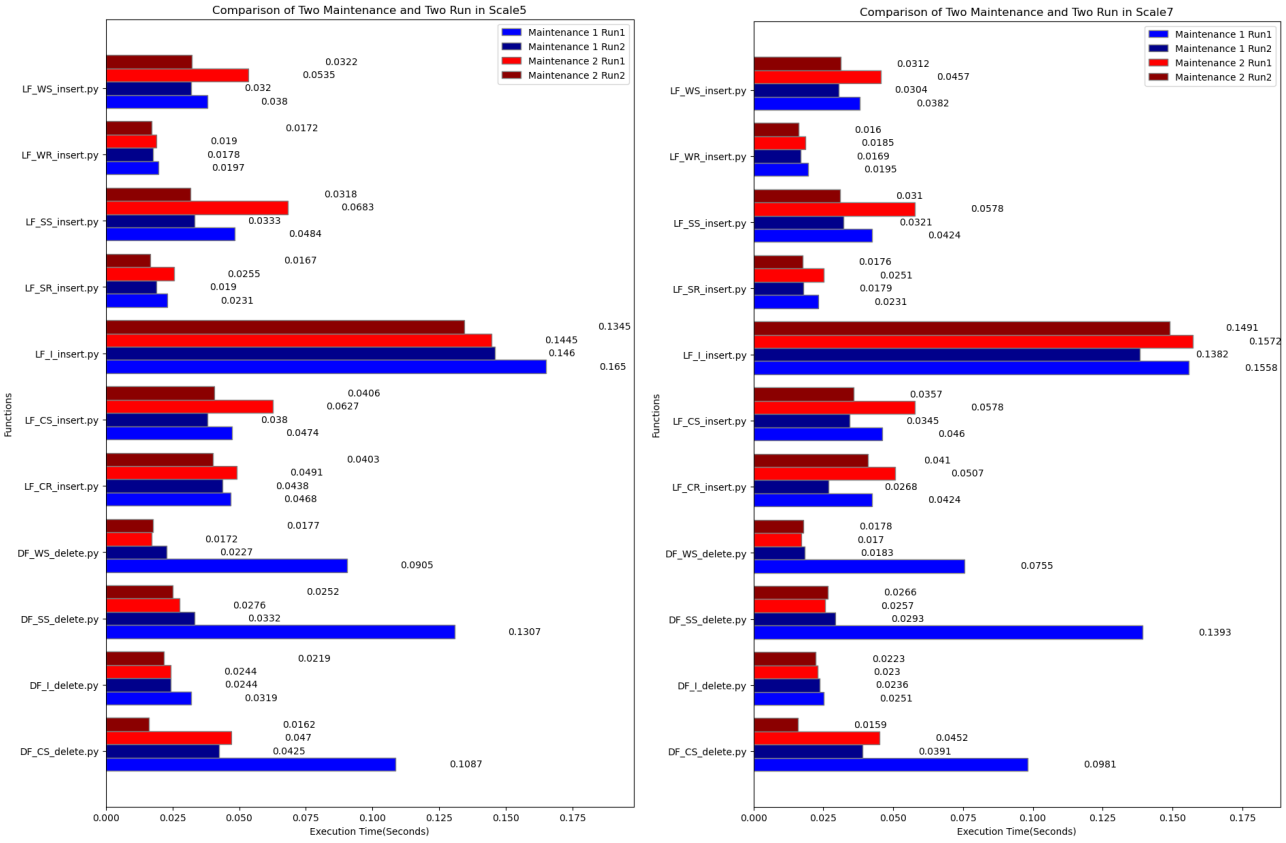
Figure 4.24, and 4.25 and 4.26 show the results for the data maintenance test 1, data maintenance test 2 and their different run respectively. It shows the execution time for each maintenance function for all scale factors. It can be noted that the deletion at the store sale fact table and insertion at the inventory fact table took more time because of the data contained in them.



(a) Scale 1

(b) Scale 3

Figure 4.24: Data Maintenance test results.



(a) Scale 5

(b) Scale 7

Figure 4.25: Maintenance test results.

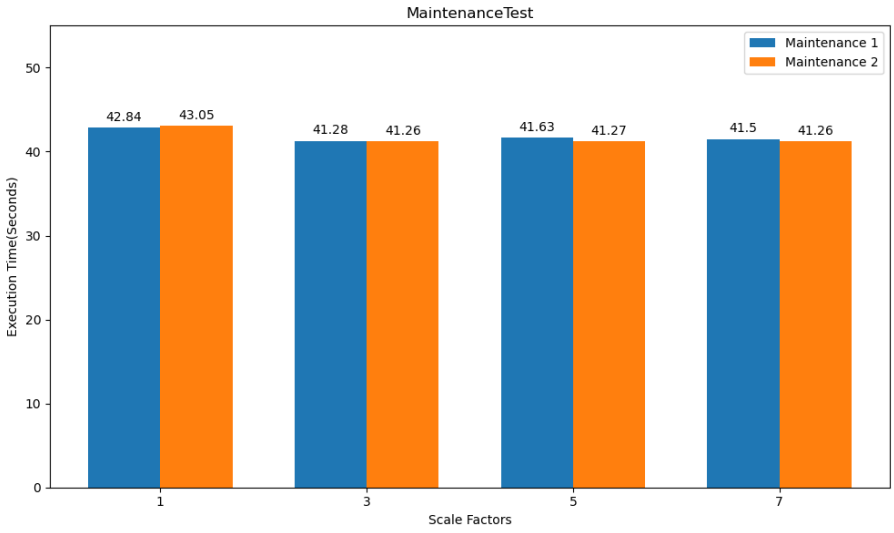
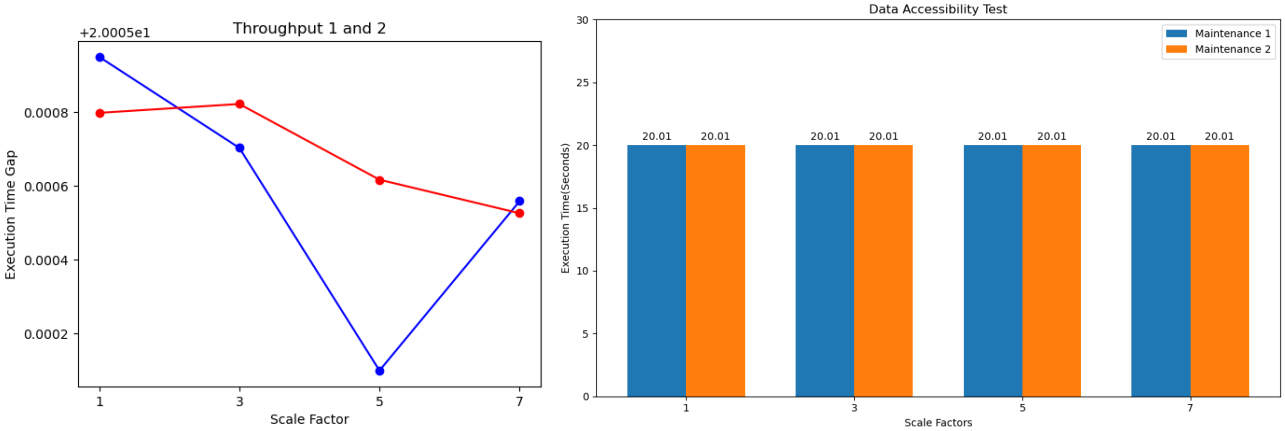


Figure 4.26: Maintenance test

4.5 Data Accessibility Test



(a) Line chart

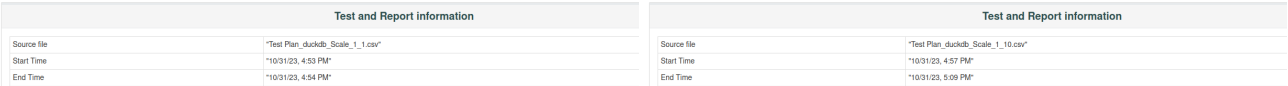
(b) Bar chart

Figure 4.27: Data accessibility test results.

s

4.6 Scalability Test

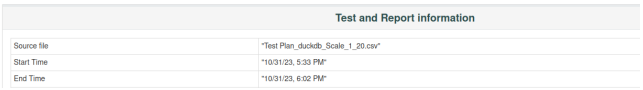
We used Apache jmeter to perform scalability testing on the DuckDB database by increasing the number of users (called thread) and then performing the 99 queries and also running the whole stream query with their stream unique identification (query_0,query_1,query_2,query_3) . It was able to perform well on scale factor 1 as we increased the user from 1 to 20 but when we tried to increase it to 50 users, the system became unresponsive. We did the same for scale factor 3, it became unresponsive for 10 users running the whole queries at once. Same thing applied for Scale factor 5 and 7, they were not able to run with 10 users. We were able to identify the slowest execution queries by filtering the time, they include query 14b, 23b, 23,67, 95 and 14. The following figures below shows the time for the test and the response time for the different scale factor and users:



(a) Scale 1

(b) Scale 1 with 10 users

Figure 4.28: Time to perform the queries



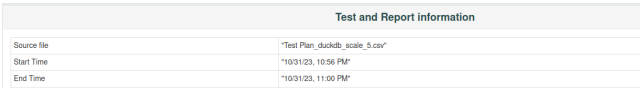
Test and Report information	
Source file	"Test Plan_duckdb_Scale_1_20.csv"
Start Time	"10/31/23, 5:33 PM"
End Time	"10/31/23, 6:02 PM"

(a) Scale 1 with 20 users

Test and Report information	
Source file	"Test Plan_duckdb_scale_3_1.csv"
Start Time	"10/31/23, 6:07 PM"
End Time	"10/31/23, 6:11 PM"

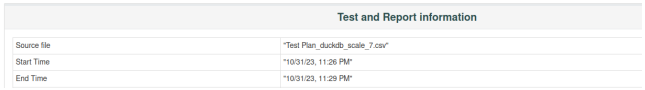
(b) Scale 3

Figure 4.29: Time to perform the queries



Test and Report information	
Source file	"Test Plan_duckdb_scale_5.csv"
Start Time	"10/31/23, 10:56 PM"
End Time	"10/31/23, 11:00 PM"

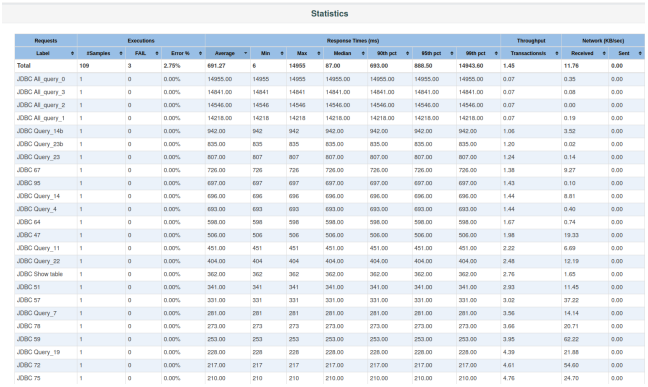
(a) Scale 5



Test and Report information	
Source file	"Test Plan_duckdb_scale_7.csv"
Start Time	"10/31/23, 11:26 PM"
End Time	"10/31/23, 11:29 PM"

(b) Scale 7

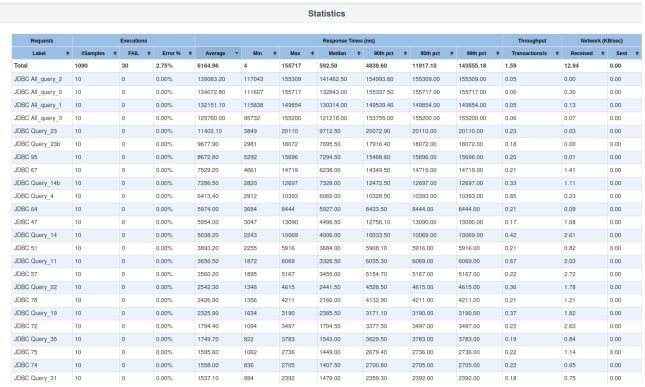
Figure 4.30: Time to perform the queries



Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	109	3	2.75%	691.27	6	14955	87.00	693.00	888.50	14943.60	1.45	11.76	0.00
JDBC All_query_0	1	0	0.00%	14955.00	14955	14955	14955.00	14955.00	14955.00	14955.00	0.07	0.35	0.00
JDBC All_query_3	1	0	0.00%	14841.00	14841	14841	14841.00	14841.00	14841.00	14841.00	0.07	0.08	0.00
JDBC All_query_2	1	0	0.00%	14546.00	14546	14546	14546.00	14546.00	14546.00	14546.00	0.07	0.00	0.00
JDBC All_query_1	1	0	0.00%	14218.00	14218	14218	14218.00	14218.00	14218.00	14218.00	0.07	0.19	0.00
JDBC Query_14b	1	0	0.00%	942.00	942	942	942.00	942.00	942.00	942.00	1.06	3.52	0.00
JDBC Query_23b	1	0	0.00%	835.00	835	835	835.00	835.00	835.00	835.00	1.20	0.02	0.00
JDBC Query_23	1	0	0.00%	807.00	807	807	807.00	807.00	807.00	807.00	1.24	0.14	0.00
JDBC 67	1	0	0.00%	726.00	726	726	726.00	726.00	726.00	726.00	1.38	9.27	0.00
JDBC 95	1	0	0.00%	697.00	697	697	697.00	697.00	697.00	697.00	1.43	0.10	0.00
JDBC Query_14	1	0	0.00%	696.00	696	696	696.00	696.00	696.00	696.00	1.44	8.81	0.00
JDBC Query_4	1	0	0.00%	693.00	693	693	693.00	693.00	693.00	693.00	1.44	0.40	0.00
JDBC 64	1	0	0.00%	598.00	598	598	598.00	598.00	598.00	598.00	1.67	0.74	0.00
JDBC 47	1	0	0.00%	506.00	506	506	506.00	506.00	506.00	506.00	1.98	19.33	0.00
JDBC Query_11	1	0	0.00%	451.00	451	451	451.00	451.00	451.00	451.00	2.22	6.69	0.00
JDBC Query_22	1	0	0.00%	404.00	404	404	404.00	404.00	404.00	404.00	2.48	12.19	0.00
JDBC Query_3	1	0	0.00%	362.00	362	362	362.00	362.00	362.00	362.00	2.76	1.65	0.00
JDBC 51	1	0	0.00%	341.00	341	341	341.00	341.00	341.00	341.00	2.93	11.45	0.00
JDBC 57	1	0	0.00%	331.00	331	331	331.00	331.00	331.00	331.00	3.02	37.22	0.00
JDBC Query_7	1	0	0.00%	281.00	281	281	281.00	281.00	281.00	281.00	3.56	14.14	0.00
JDBC 78	1	0	0.00%	273.00	273	273	273.00	273.00	273.00	273.00	3.66	20.71	0.00
JDBC 59	1	0	0.00%	253.00	253	253	253.00	253.00	253.00	253.00	3.95	62.22	0.00
JDBC Query_19	1	0	0.00%	228.00	228	228	228.00	228.00	228.00	228.00	4.39	21.88	0.00
JDBC 72	1	0	0.00%	217.00	217	217	217.00	217.00	217.00	217.00	4.61	54.60	0.00
JDBC 75	1	0	0.00%	210.00	210	210	210.00	210.00	210.00	210.00	4.76	24.70	0.00

(a) Scale 1

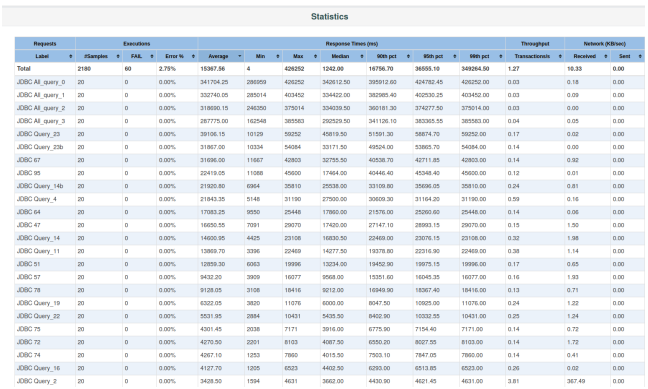


Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	1090	30	2.75%	6164.96	4	155717	592.50	4839.60	11917.10	143555.18	1.59	12.94	0.00
JDBC All_query_2	10	0	0.00%	139083.20	117043	155309	141462.50	154993.80	155309.00	155309.00	0.05	0.00	0.00
JDBC All_query_0	10	0	0.00%	134072.80	111607	155717	132843.00	155337.50	155717.00	155717.00	0.06	0.30	0.00
JDBC All_query_1	10	0	0.00%	132151.10	115838	149854	130314.00	149539.40	149854.00	149854.00	0.05	0.13	0.00
JDBC All_query_3	10	0	0.00%	125760.00	95732	155200	121218.00	153755.00	155200.00	155200.00	0.06	0.07	0.00
JDBC Query_23	10	0	0.00%	11402.10	3849	20110	9712.50	20072.90	20110.00	20110.00	0.23	0.03	0.00
JDBC Query_23b	10	0	0.00%	9677.90	2981	18072	7695.50	17916.40	18072.00	18072.00	0.18	0.00	0.00
JDBC 95	10	0	0.00%	8672.80	5292	15696	7294.50	15468.60	15696.00	15696.00	0.20	0.01	0.00
JDBC 67	10	0	0.00%	7529.20	4661	14719	6238.00	14346.50	14719.00	14719.00	0.21	1.41	0.00
JDBC Query_14b	10	0	0.00%	7286.50	2820	12697	7328.00	12472.50	12697.00	12697.00	0.33	1.11	0.00
JDBC Query_4	10	0	0.00%	6413.40	2912	10393	6069.00	10328.50	10393.00	10393.00	0.85	0.23	0.00
JDBC 64	10	0	0.00%	5974.00	3684	8444	5927.00	8433.50	8444.00	8444.00	0.21	0.09	0.00
JDBC 47	10	0	0.00%	5954.00	3047	13090	4496.50	12756.10	13090.00	13090.00	0.17	1.68	0.00
JDBC Query_14	10	0	0.00%	5038.20	2243	10069	4006.00	10033.50	10069.00	10069.00	0.42	2.61	0.00
JDBC 51	10	0	0.00%	3693.20	2255	5916	3684.00	5906.10	5916.00	5916.00	0.21	0.82	0.00
JDBC Query_11	10	0	0.00%	3656.50	1872	6069	3326.50	6035.30	6069.00	6069.00	0.67	2.03	0.00
JDBC 57	10	0	0.00%	3560.20	1895	5167	3455.00	5154.70	5167.00	5167.00	0.22	2.72	0.00
JDBC Query_22	10	0	0.00%	2542.30	1346	4615	2441.50	4528.50	4615.00	4615.00	0.36	1.78	0.00
JDBC 78	10	0	0.00%	2426.90	1356	4211	2166.00	4132.90	4211.00	4211.00	0.21	1.21	0.00
JDBC Query_19	10	0	0.00%	2325.90	1634	3190	2385.50	3171.10	3190.00	3190.00	0.37	1.82	0.00
JDBC 72	10	0	0.00%	1794.40	1094	3497	1704.50	3377.50	3497.00	3497.00	0.22	2.63	0.00
JDBC Query_35	10	0	0.00%	1749.70	922	3783	1543.00	3629.50	3783.00	3783.00	0.19	0.84	0.00
JDBC 75	10	0	0.00%	1595.60	1062	2736	1448.00	2679.40	2736.00	2736.00	0.22	1.14	0.00
JDBC 74	10	0	0.00%	1558.00	836	2705	1407.50	2700.80	2705.00	2705.00	0.22	0.65	0.00
JDBC Query_31	10	0	0.00%	1537.10	994	2392	1479.00	2359.30	2392.00	2392.00	0.18	0.75	0.00

(b) Scale 1 with 10 users

Figure 4.31: Statistics of Queries



Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	2180	60	2.75%	15367.56	4	426252	1242.00	16756.70	36555.10	349264.50	1.27	10.33	0.00
JDBC All_query_0	20	0	0.00%	341704.25	286959	426252	342612.50	395912.60	424782.45	426252.00	0.03	0.18	0.00
JDBC All_query_1	20	0	0.00%	332740.05	285014	403452	334422.00	382985.40	402530.25	403452.00	0.03	0.09	0.00
JDBC All_query_2	20	0	0.00%	318690.15	246350	375014	334039.50	360181.30	374277.50	375014.00	0.03	0.00	0.00
JDBC All_query_3	20	0	0.00%	287775.00	162548	385583	292529.50	341126.10	383365.55	385583.00	0.04	0.05	0.00
JDBC Query_23	20	0	0.00%	39106.15	10129	59252	45819.50	51591.30	56874.70	59252.00	0.17	0.02	0.00
JDBC Query_23b	20	0	0.00%	31867.00	10334	54084	33171.50	49524.00	53865.70	54084.00	0.14	0.00	0.00
JDBC 67	20	0	0.00%	31696.00	11667	42803	32755.50	40538.70	42711.85	42803.00	0.14	0.92	0.00
JDBC 95	20	0	0.00%	22419.05	11088	45600	17464.00	40446.40	45348.40	45600.00	0.12	0.01	0.00
JDBC Query_14b	20	0	0.00%	21920.80	6964	35810	25538.00	33109.80	35696.05	35810.00	0.24	0.81	0.00
JDBC Query_4	20	0	0.00%	21843.35	5146	31190	27900.00	30609.30	31164.20	31190.00	0.59	0.16	0.00
JDBC 64	20	0	0.00%	17083.25	9550	25446	17860.00	21576.00	25260.60	25446.00	0.14	0.06	0.00
JDBC 47	20	0	0.00%	16650.55	7091	29070	17420.00	27147.10	28993.15	29070.00	0.15	1.50	0.00
JDBC Query_14	20	0	0.00%	14600.95	4425	23108	16830.50	22469.00	23076.15	23108.00	0.32	1.98	0.00
JDBC Query_11	20	0	0.00%	13869.70	3396	22469	14277.50	19378.80	22316.90	22469.00	0.38	1.14	0.00
JDBC 51	20	0	0.00%	12859.30	6063	19996	13234.00	19452.90	19975.15	19996.00	0.17	0.65	0.00
JDBC 57	20	0	0.00%	9432.20	3909	16077	9568.00	15351.60	16045.35	16077.00	0.16	1.93	0.00
JDBC 78	20	0	0.00%	9128.05	3108	18416	9212.00	16949.90	18367.40	18416.00	0.13	0.71	0.00
JDBC Query_19	20	0	0.00%	6322.05	3820	11076	6000.00	8047.50	10925.00	11076.00	0.24	1.22	0.00
JDBC Query_22	20	0	0.00%	5531.95	2884	10431	5435.50	8402.90	10032.55	10431.00	0.25	1.24	0.00
JDBC 75	20	0	0.00%	4301.45	2038	7171	3916.00	6775.90	7154.40	7171.00	0.14	0.72	0.00
JDBC 72	20	0	0.00%	4270.50	2201	8103	4087.50	6550.20	8027.55	8103.00	0.14	1.72	0.00
JDBC 74	20	0	0.00%	4267.10	1253	7860	4015.50	7503.10	7847.05	7860.00	0.14	0.41	0.00
JDBC Query_16	20	0	0.00%	4127.70	1205	6523	4402.50	6293.00	6513.85	6523.00	0.26	0.02	0.00
JDBC Query_2	20	0	0.00%	3428.50	1594	4631	3662.00	4430.90	4621.45	4631.00	3.81	367.49	0.00

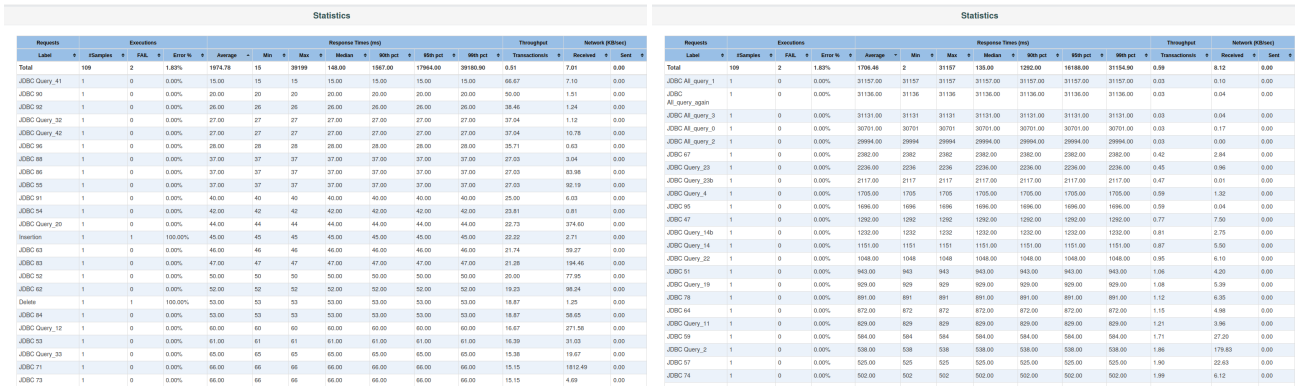
(a) Scale 1 with 20 users

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	109	2	1.83%	2225.72	10	46192	169.00	1961.00	19740.50	45870.60	0.45	6.22	0.00
JDBC All_query_0	1	0	0.00%	46192.00	46192	46192	46192.00	46192.00	46192.00	46192.00	0.02	0.11	0.00
JDBC All_query_again	1	0	0.00%	42978.00	42978	42978	42978.00	42978.00	42978.00	42978.00	0.02	0.03	0.00
JDBC All_query_2	1	0	0.00%	38815.00	38815	38815	38815.00	38815.00	38815.00	38815.00	0.03	0.00	0.00
JDBC All_query_3	1	0	0.00%	36926.00	36926	36926	36926.00	36926.00	36926.00	36926.00	0.03	0.03	0.00
JDBC All_query_1	1	0	0.00%	36399.00	36399	36399	36399.00	36399.00	36399.00	36399.00	0.03	0.08	0.00
JDBC 67	1	0	0.00%	3082.00	3082	3082	3082.00	3082.00	3082.00	3082.00	0.32	2.20	0.00
JDBC Query_23b	1	0	0.00%	3001.00	3001	3001	3001.00	3001.00	3001.00	3001.00	0.33	0.01	0.00
JDBC Query_23	1	0	0.00%	2641.00	2641	2641	2641.00	2641.00	2641.00	2641.00	0.38	0.81	0.00
JDBC 95	1	0	0.00%	2547.00	2547	2547	2547.00	2547.00	2547.00	2547.00	0.39	0.03	0.00
JDBC 47	1	0	0.00%	1730.00	1730	1730	1730.00	1730.00	1730.00	1730.00	0.58	5.72	0.00
JDBC Query_14b	1	0	0.00%	1661.00	1661	1661	1661.00	1661.00	1661.00	1661.00	0.60	2.04	0.00
JDBC Query_4	1	0	0.00%	1637.00	1637	1637	1637.00	1637.00	1637.00	1637.00	0.61	1.38	0.00
JDBC Query_19	1	0	0.00%	1372.00	1372	1372	1372.00	1372.00	1372.00	1372.00	0.73	3.65	0.00
JDBC 78	1	0	0.00%	1277.00	1277	1277	1277.00	1277.00	1277.00	1277.00	0.78	4.43	0.00
JDBC Query_22	1	0	0.00%	1271.00	1271	1271	1271.00	1271.00	1271.00	1271.00	0.79	5.03	0.00
JDBC Query_14	1	0	0.00%	1237.00	1237	1237	1237.00	1237.00	1237.00	1237.00	0.81	5.11	0.00
JDBC 64	1	0	0.00%	1194.00	1194	1194	1194.00	1194.00	1194.00	1194.00	0.84	3.64	0.00
JDBC 51	1	0	0.00%	1183.00	1183	1183	1183.00	1183.00	1183.00	1183.00	0.85	3.35	0.00
JDBC Query_11	1	0	0.00%	1072.00	1072	1072	1072.00	1072.00	1072.00	1072.00	0.93	3.06	0.00
JDBC 74	1	0	0.00%	656.00	656	656	656.00	656.00	656.00	656.00	1.52	4.68	0.00
JDBC 59	1	0	0.00%	644.00	644	644	644.00	644.00	644.00	644.00	1.55	24.67	0.00
JDBC 57	1	0	0.00%	614.00	614	614	614.00	614.00	614.00	614.00	1.63	19.05	0.00
JDBC 75	1	0	0.00%	588.00	588	588	588.00	588.00	588.00	588.00	1.70	8.90	0.00

(b) Scale 3

Figure 4.32: Statistics of Queries



Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	109	2	1.83%	1974.78	15	39199	148.00	1567.00	17964.00	39180.90	0.51	7.01	0.00
JDBC Query_41	1	0	0.00%	15.00	15	15	15.00	15.00	15.00	15.00	66.67	7.10	0.00
JDBC 90	1	0	0.00%	20.00	20	20	20.00	20.00	20.00	20.00	50.00	1.51	0.00
JDBC 92	1	0	0.00%	26.00	26	26	26.00	26.00	26.00	26.00	38.46	1.24	0.00
JDBC Query_32	1	0	0.00%	27.00	27	27	27.00	27.00	27.00	27.00	37.04	1.12	0.00
JDBC Query_42	1	0	0.00%	27.00	27	27	27.00	27.00	27.00	27.00	37.04	10.78	0.00
JDBC 96	1	0	0.00%	28.00	28	28	28.00	28.00	28.00	28.00	35.71	0.63	0.00
JDBC 88	1	0	0.00%	37.00	37	37	37.00	37.00	37.00	37.00	27.03	3.04	0.00
JDBC 86	1	0	0.00%	37.00	37	37	37.00	37.00	37.00	37.00	27.03	83.98	0.00
JDBC 55	1	0	0.00%	37.00	37	37	37.00	37.00	37.00	37.00	27.03	92.19	0.00
JDBC 91	1	0	0.00%	40.00	40	40	40.00	40.00	40.00	40.00	25.00	6.03	0.00
JDBC 54	1	0	0.00%	42.00	42	42	42.00	42.00	42.00	42.00	23.81	0.81	0.00
JDBC Query_20	1	0	0.00%	44.00	44	44	44.00	44.00	44.00	44.00	22.73	374.60	0.00
Insertion	1	1	100.00%	45.00	45	45	45.00	45.00	45.00	45.00	22.22	2.71	0.00
JDBC 63	1	0	0.00%	46.00	46	46	46.00	46.00	46.00	46.00	21.74	59.27	0.00
JDBC 83	1	0	0.00%	47.00	47	47	47.00	47.00	47.00	47.00	21.28	194.46	0.00
JDBC 52	1	0	0.00%	50.00	50	50	50.00	50.00	50.00	50.00	20.00	77.95	0.00
JDBC 62	1	0	0.00%	52.00	52	52	52.00	52.00	52.00	52.00	19.23	98.24	0.00
Delete	1	1	100.00%	53.00	53	53	53.00	53.00	53.00	53.00	18.87	1.25	0.00
JDBC 84	1	0	0.00%	53.00	53	53	53.00	53.00	53.00	53.00	18.87	58.65	0.00
JDBC Query_12	1	0	0.00%	60.00	60	60	60.00	60.00	60.00	60.00	16.67	271.58	0.00
JDBC 53	1	0	0.00%	61.00	61	61	61.00	61.00	61.00	61.00	16.39	31.03	0.00
JDBC Query_33	1	0	0.00%	65.00	65	65	65.00	65.00	65.00	65.00	15.38	19.67	0.00
JDBC 71	1	0	0.00%	66.00	66	66	66.00	66.00	66.00	66.00	15.15	1812.49	0.00
JDBC 73	1	0	0.00%	66.00	66	66	66.00	66.00	66.00	66.00	15.15	4.69	0.00

(a) Scale 5

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	109	2	1.83%	1706.46	2	31157	135.00	1292.00	16188.00	31154.90	0.59	8.12	0.00
JDBC All_query_1	1	0	0.00%	31157.00	31157	31157	31157.00	31157.00	31157.00	31157.00	0.03	0.10	0.00
JDBC All_query_again	1	0	0.00%	31136.00	31136	31136	31136.00	31136.00	31136.00	31136.00	0.03	0.04	0.00
JDBC All_query_3	1	0	0.00%	31131.00	31131	31131	31131.00	31131.00	31131.00	31131.00	0.03	0.04	0.00
JDBC All_query_0	1	0	0.00%	30701.00	30701	30701	30701.00	30701.00	30701.00	30701.00	0.03	0.17	0.00
JDBC All_query_2	1	0	0.00%	29994.00	29994	29994	29994.00	29994.00	29994.00	29994.00	0.03	0.00	0.00
JDBC 67	1	0	0.00%	2382.00	2382	2382	2382.00	2382.00	2382.00	2382.00	0.42	2.84	0.00
JDBC Query_23	1	0	0.00%	2236.00	2236	2236	2236.00	2236.00	2236.00	2236.00	0.45	0.96	0.00
JDBC Query_23b	1	0	0.00%	2117.00	2117	2117	2117.00	2117.00	2117.00	2117.00	0.47	0.01	0.00
JDBC Query_4	1	0	0.00%	1705.00	1705	1705	1705.00	1705.00	1705.00	1705.00	0.59	1.32	0.00
JDBC 95	1	0	0.00%	1696.00	1696	1696	1696.00	1696.00	1696.00	1696.00	0.59	0.04	0.00
JDBC 47	1	0	0.00%	1292.00	1292	1292	1292.00	1292.00	1292.00	1292.00	0.77	7.50	0.00
JDBC Query_14b	1	0	0.00%	1232.00	1232	1232	1232.00	1232.00	1232.00	1232.00	0.81	2.75	0.00
JDBC Query_14	1	0	0.00%	1151.00	1151	1151	1151.00	1151.00	1151.00	1151.00	0.87	5.50	0.00
JDBC Query_22	1	0	0.00%	1048.00	1048	1048	1048.00	1048.00	1048.00	1048.00	0.95	6.10	0.00
JDBC 51	1	0	0.00%	943.00	943	943	943.00	943.00	943.00	943.00	1.06	4.20	0.00
JDBC Query_19	1	0	0.00%	929.00	929	929	929.00	929.00	929.00	929.00	1.08	5.39	0.00
JDBC 78	1	0	0.00%	891.00	891	891	891.00	891.00	891.00	891.00	1.12	6.35	0.00
JDBC 64	1	0	0.00%	872.00	872	872	872.00	872.00	872.00	872.00	1.15	4.98	0.00
JDBC Query_11	1	0	0.00%	829.00	829	829	829.00	829.00	829.00	829.00	1.21	3.96	0.00
JDBC 59	1	0	0.00%	584.00	584	584	584.00	584.00	584.00	584.00	1.71	27.20	0.00
JDBC Query_2	1	0	0.00%	538.00	538	538	538.00	538.00	538.00	538.00	1.86	179.83	0.00
JDBC 57	1	0	0.00%	525.00	525	525	525.00	525.00	525.00	525.00	1.90	22.63	0.00
JDBC 74	1	0	0.00%	502.00	502	502	502.00	502.00	502.00	502.00	1.99	6.12	0.00

(b) Scale 7

Figure 4.33: Statistics of Queries

4.7 Optimization

Considering the results from the low performance queries, optimization were made. The optimized queries are detailed in Appendix A. The optimization were done using the following:

- The Query plans were studied to analyze possible points of optimization.
- Indexes were added for fast retrieval of table tuples.
- Creating of TEMPORARY table to store and manipulate results

Query 95

Query 95 that has the question, *Produce a count of web sales and total shipping cost and net profit in a given 60 day period to customers in a given state from a named web site for returned orders shipped from more than one warehouse.*
Qualification Substitution Parameters:

- STATE.01=IL
- MONTH.01=2
- YEAR.01=1999

The original Query was:

```
    with ws_wh as
(select ws1.ws_order_number,ws1.ws_warehouse_sk wh1,ws2.ws_warehouse_sk wh2
from web_sales ws1,web_sales ws2
where ws1.ws_order_number = ws2.ws_order_number
  and ws1.ws_warehouse_sk <> ws2.ws_warehouse_sk)
select
  count(distinct ws_order_number) as "order count"
 ,sum(ws_ext_ship_cost) as "total shipping cost"
 ,sum(ws_net_profit) as "total net profit"
from
  web_sales ws1
 ,date_dim
 ,customer_address
 ,web_site
where
   d_date between '1999-02-01' and
          (cast('1999-02-01' as date) + interval 60 days)
and ws1.ws_ship_date_sk = d_date_sk
and ws1.ws_ship_addr_sk = ca_address_sk
and ca_state = 'IL'
and ws1.ws_web_site_sk = web_site_sk
and web_company_name = 'pri'
and ws1.ws_order_number in (select ws_order_number
                            from ws_wh)
and ws1.ws_order_number in (select wr_order_number
                            from web_returns,ws_wh
                            where wr_order_number = ws_wh.ws_order_number)
order by count(distinct ws_order_number)
limit 100;
```

Listing 4.1: Query 95.

This query involves joins on several tables (web_sales, date_dim, customer_address and web_site), and filtering conditions to retrieve count order of web sales and total shipping cost and net profit. The following was done to optimize this query

- creating indexes,
- using CTEs, and

- optimizing the structure of the SQL statement

```
    CREATE INDEX if not exists idx_web_sales_order_number ON web_sales(ws_order_number);

-- Create an index on web_returns to improve join performance
CREATE INDEX if not exists idx_web_returns_order_number ON web_returns(wr_order_number);

-- Use CTEs for better readability and performance
WITH OrderWarehouses AS (
    SELECT ws1.ws_order_number, ws1.ws_warehouse_sk AS wh1, ws2.ws_warehouse_sk AS wh2
    FROM web_sales ws1
    JOIN web_sales ws2 ON ws1.ws_order_number = ws2.ws_order_number
    WHERE ws1.ws_warehouse_sk <> ws2.ws_warehouse_sk
),
FilteredOrders AS (
    SELECT DISTINCT ws_order_number
    FROM web_sales
    WHERE ws_order_number IN (SELECT ws_order_number FROM OrderWarehouses)
),
FilteredReturns AS (
    SELECT DISTINCT wr_order_number
    FROM web_returns wr
    JOIN OrderWarehouses ow ON wr.wr_order_number = ow.ws_order_number
),
DateRange AS (
    SELECT d_date_sk
    FROM date_dim
    WHERE d_date BETWEEN '1999-02-01' AND ('1999-02-01'::DATE + INTERVAL '60 days')
)
-- Retrieve results
SELECT
    COUNT(DISTINCT ws.ws_order_number) AS "order count",
    SUM(ws.ws_ext_ship_cost) AS "total shipping cost",
    SUM(ws.ws_net_profit) AS "total net profit"
FROM web_sales ws
JOIN DateRange dr ON ws.ws_ship_date_sk = dr.d_date_sk
JOIN customer_address ca ON ws.ws_ship_addr_sk = ca.ca_address_sk
```

```
36 JOIN web_site ws2 ON ws.ws_web_site_sk = ws2.web_site_sk
37 WHERE ca.ca_state = 'IL'
38     AND ws2.web_company_name = 'pri'
39     AND ws.ws_order_number IN (SELECT * FROM FilteredOrders)
40     AND ws.ws_order_number IN (SELECT * FROM FilteredReturns)
41 GROUP BY dr.d_date_sk
42 ORDER BY "order count" DESC
43 LIMIT 100;
```

Listing 4.2: Optimized Query 95.

As a result, testing these changes in SF1, the Query 95 went from 0.1328871250152588 seconds to 0.0005030632019042969 seconds, SF-3 from 0.48744726181030273 seconds to 0.0005521774291992188 seconds, SF-5 from 0.7921462059020996 seconds to 0.0007989406585693359 seconds and SF-7 from 1.0165119171142578 seconds to 0.0007801055908203125 seconds. The figure below shows the result. They were so far apart by orders of magnitude that we have to use a logarithmic scale on the y-axis.

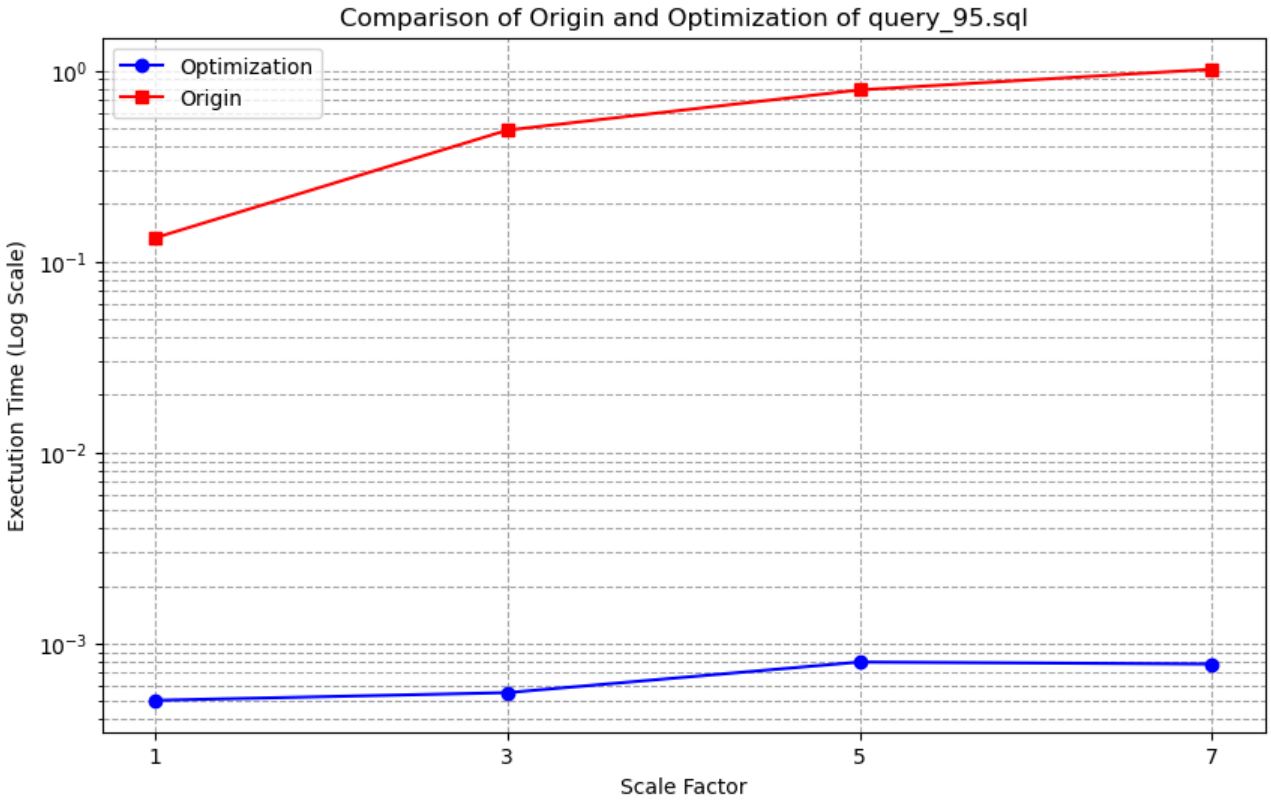


Figure 4.34: Optimization/Comparison of Origin and Optimization of query 95

Query 67

Query 67 has the question *Find top stores for each category based on store sales in a specific year.* Qualification Substitution Parameters:

- DMS.01 = 1200

The original Query was:

```
    select  *
from (select i_category
           ,i_class
           ,i_brand
           ,i_product_name
           ,d_year
           ,d_qoy
           ,d_moy
           ,s_store_id
           ,sumsales
           ,rank() over (partition by i_category order by sumsales desc) rk
     from (select i_category
                 ,i_class
                 ,i_brand
                 ,i_product_name
                 ,d_year
                 ,d_qoy
                 ,d_moy
                 ,s_store_id
                 ,sum(coalesce(ss_sales_price*ss_quantity,0)) sumsales
           from store_sales
               ,date_dim
               ,store
               ,item
      where  ss_sold_date_sk=d_date_sk
         and ss_item_sk=i_item_sk
         and ss_store_sk = s_store_sk
         and d_month_seq between 1200 and 1200+11
      group by  rollup(i_category, i_class, i_brand, i_product_name, d_year,
    d_qoy, d_moy,s_store_id))dw1) dw2
where rk <= 100
```

```
order by i_category
       ,i_class
       ,i_brand
       ,i_product_name
       ,d_year
       ,d_qoy
       ,d_moy
       ,s_store_id
       ,sumsales
       ,rk
limit 100;
```

Listing 4.3: Query 67.

This query was optimized by the use of indexing. Index were added on items, date_dim, store_sales and store.

```
--Indexes for store_sales
CREATE INDEX idx_store_sales_sold_date ON store_sales(ss_sold_date_sk);
CREATE INDEX idx_store_sales_item ON store_sales(ss_item_sk);
CREATE INDEX idx_store_sales_store ON store_sales(ss_store_sk);

--Indexes for item
CREATE INDEX idx_item_item_sk ON item(i_item_sk);

--Indexes for store
CREATE INDEX idx_store_store_sk ON store(s_store_sk);

--Indexes for date_dim
CREATE INDEX idx_date_dim_date_sk ON date_dim(d_date_sk);
```

Listing 4.4: Optimized Query 67.

The result is shown below:

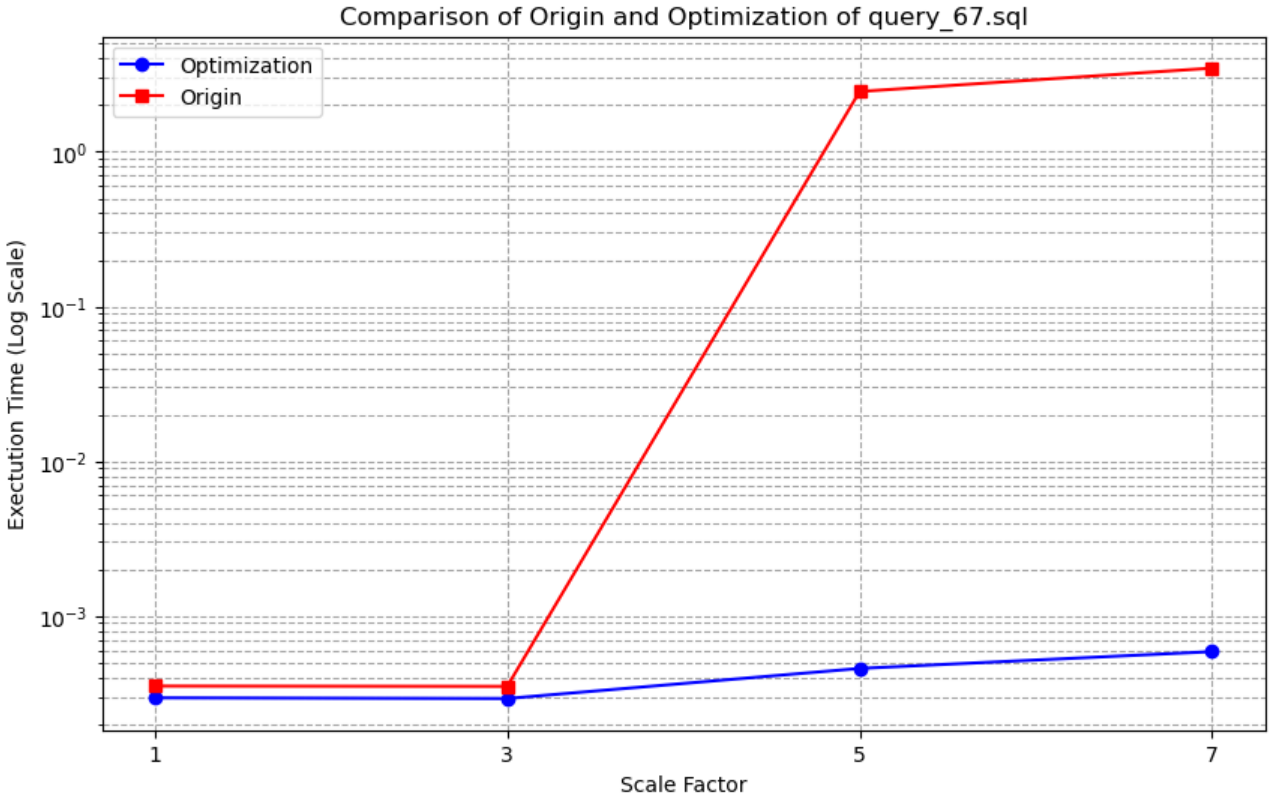


Figure 4.35: Optimization/Comparison of Origin and Optimization of query 67

Query 23b

Query 23 was divided into two part because of the question it has to answer. This query contains multiple, related iterations: has the question *Find frequently sold items that were sold more than 4 times on any day during four consecutive years through the store sales channel. Compute the maximum store sales made by any given customer in a period of four consecutive years (same as above). Compute the best store customers that are in the 5th percentile of sales. Finally, compute the total web and catalog sales in a particular month made by our best store customers buying our most frequent store items.*
Qualification Substitution Parameters:

- MONTH.01 = 2
- YEAR.01 = 2000
- TOPPERCENT=50

Original Query for 23b

```
with frequent_ss_items as
(select substr(i_item_desc,1,30) itemdesc,i_item_sk item_sk,d_date solddate,
   count(*) cnt
 from store_sales
     ,date_dim
     ,item
 where ss_sold_date_sk = d_date_sk
   and ss_item_sk = i_item_sk
   and d_year in (2000,2000+1,2000+2,2000+3)
 group by substr(i_item_desc,1,30),i_item_sk,d_date
 having count(*) >4),
max_store_sales as
(select max(csales) tpcds_cmax
 from (select c_customer_sk,sum(ss_quantity*ss_sales_price) csales
       from store_sales
           ,customer
           ,date_dim
       where ss_customer_sk = c_customer_sk
        and ss_sold_date_sk = d_date_sk
        and d_year in (2000,2000+1,2000+2,2000+3)
       group by c_customer_sk)),
best_ss_customer as
(select c_customer_sk,sum(ss_quantity*ss_sales_price) ssales
 from store_sales
     ,customer
 where ss_customer_sk = c_customer_sk
 group by c_customer_sk
 having sum(ss_quantity*ss_sales_price) > (50/100.0) * (select
 *
from
max_store_sales))
 select  sum(sales)
from (select cs_quantity*cs_list_price sales
      from catalog_sales
          ,date_dim
      where d_year = 2000
        and d_moy = 2
        and cs_sold_date_sk = d_date_sk
        and cs_item_sk in (select item_sk from frequent_ss_items)
        and cs_bill_customer_sk in (select c_customer_sk from
```

```
   best_ss_customer)
40   union all
41   select ws_quantity*ws_list_price sales
42    from web_sales
43        ,date_dim
44    where d_year = 2000
45      and d_moy = 2
46      and ws_sold_date_sk = d_date_sk
47      and ws_item_sk in (select item_sk from frequent_ss_items)
48      and ws_bill_customer_sk in (select c_customer_sk from
   best_ss_customer))
49 limit 100;
```

Listing 4.5: Query 23b.

The query was optimized by the use a TEMPORARY table instead of using the Common Table Expressions and then index was added on the temporary table, as seen below:

```
1  CREATE TEMPORARY TABLE frequent_ss_items AS
2  SELECT
3      SUBSTR(i_item_desc, 1, 30) AS itemdesc,
4      i_item_sk AS item_sk,
5      d_date AS solddate,
6      COUNT(*) AS cnt
7  FROM
8      store_sales
9  JOIN
10     date_dim ON ss_sold_date_sk = d_date_sk
11 JOIN
12     item ON ss_item_sk = i_item_sk
13 WHERE
14     d_year IN (2000, 2001, 2002, 2003)
15 GROUP BY
16     SUBSTR(i_item_desc, 1, 30), i_item_sk, d_date
17 HAVING COUNT(*) > 4;
18
19 -- Create an index on the temporary table
20 CREATE INDEX idx_frequent_items ON frequent_ss_items (item_sk);
21
22 -- Create a temporary table for maximum store sales
23 CREATE TEMPORARY TABLE max_store_sales AS
```

```
SELECT MAX(csales) AS tpcds_cmax
FROM (
    SELECT
        c_customer_sk,
        SUM(ss_quantity * ss_sales_price) AS csales
    FROM
        store_sales
    JOIN
        customer ON ss_customer_sk = c_customer_sk
    JOIN
        date_dim ON ss_sold_date_sk = d_date_sk
    WHERE
        d_year IN (2000, 2001, 2002, 2003)
    GROUP BY
        c_customer_sk
);

-- Create a temporary table for the best store sales customers
CREATE TEMPORARY TABLE best_ss_customer AS
SELECT
    c_customer_sk,
    SUM(ss_quantity * ss_sales_price) AS ssales
FROM
    store_sales
JOIN
    customer ON ss_customer_sk = c_customer_sk
GROUP BY
    c_customer_sk
HAVING
    SUM(ss_quantity * ss_sales_price) > (50/100.0) * (SELECT tpcds_cmax FROM
    max_store_sales);

-- Finally, run your main query using the temporary tables
SELECT
    c_last_name,
    c_first_name,
    sales
FROM (
    SELECT
        c_last_name,
```

```
63          c_first_name,
64          SUM(cs_quantity * cs_list_price) AS sales
65      FROM
66          catalog_sales
67      JOIN
68          customer ON cs_bill_customer_sk = c_customer_sk
69      JOIN
70          date_dim ON cs_sold_date_sk = d_date_sk
71      WHERE
72          d_year = 2000
73      AND
74          d_moy = 2
75      AND
76          cs_item_sk IN (SELECT item_sk FROM frequent_ss_items)
77      AND
78          cs_bill_customer_sk IN (SELECT c_customer_sk FROM best_ss_customer)
79      AND
80          cs_bill_customer_sk = c_customer_sk
81      GROUP BY
82          c_last_name, c_first_name
83      UNION ALL
84      SELECT
85          c_last_name,
86          c_first_name,
87          SUM(ws_quantity * ws_list_price) AS sales
88      FROM
89          web_sales
90      JOIN
91          customer ON ws_bill_customer_sk = c_customer_sk
92      JOIN
93          date_dim ON ws_sold_date_sk = d_date_sk
94      WHERE
95          d_year = 2000
96      AND
97          d_moy = 2
98      AND
99          ws_item_sk IN (SELECT item_sk FROM frequent_ss_items)
100     AND
101         ws_bill_customer_sk IN (SELECT c_customer_sk FROM best_ss_customer)
102     AND
```

```
        ws_bill_customer_sk = c_customer_sk
    GROUP BY
        c_last_name, c_first_name
) AS final_results
ORDER BY
    c_last_name, c_first_name, sales
LIMIT 100;
```

Listing 4.6: Optimized Query 23b.

The result is visualized below:

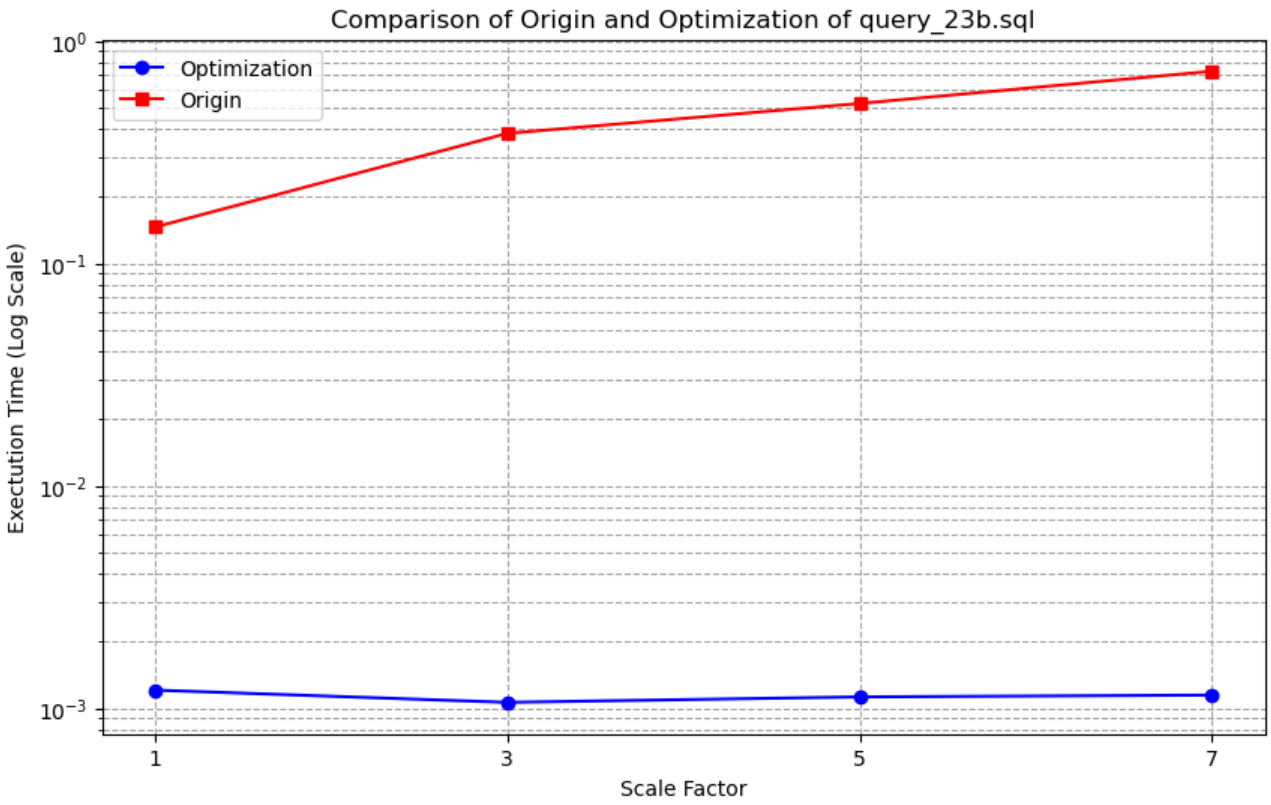


Figure 4.36: Optimization/Comparison of Origin and Optimization of query 23b

Query 4

Query 4 answer the question, *Find customers who spend more money via catalog than in stores. Identify preferred customers and their country of origin.*

Qualification Substitution Parameters:

- • YEAR.01=2001

- • SELECTONE.01= t_s_secyear.customer_preferred_cust_flag

The Original Query is :

```
   with year_total as (
select c_customer_id customer_id
      ,c_first_name customer_first_name
      ,c_last_name customer_last_name
      ,c_preferred_cust_flag customer_preferred_cust_flag
      ,c_birth_country customer_birth_country
      ,c_login customer_login
      ,c_email_address customer_email_address
      ,d_year dyear
      ,sum(((ss_ext_list_price-ss_ext_wholesale_cost-ss_ext_discount_amt)+
   ss_ext_sales_price)/2) year_total
      ,'s' sale_type
from customer
    ,store_sales
    ,date_dim
where c_customer_sk = ss_customer_sk
  and ss_sold_date_sk = d_date_sk
group by c_customer_id
        ,c_first_name
        ,c_last_name
        ,c_preferred_cust_flag
        ,c_birth_country
        ,c_login
        ,c_email_address
        ,d_year
union all
select c_customer_id customer_id
      ,c_first_name customer_first_name
      ,c_last_name customer_last_name
      ,c_preferred_cust_flag customer_preferred_cust_flag
      ,c_birth_country customer_birth_country
      ,c_login customer_login
      ,c_email_address customer_email_address
      ,d_year dyear
      ,sum((((cs_ext_list_price-cs_ext_wholesale_cost-cs_ext_discount_amt)+
   cs_ext_sales_price)/2) ) year_total
      ,'c' sale_type
```

```
from customer
    ,catalog_sales
    ,date_dim
where c_customer_sk = cs_bill_customer_sk
  and cs_sold_date_sk = d_date_sk
group by c_customer_id
        ,c_first_name
        ,c_last_name
        ,c_preferred_cust_flag
        ,c_birth_country
        ,c_login
        ,c_email_address
        ,d_year
union all
select c_customer_id customer_id
      ,c_first_name customer_first_name
      ,c_last_name customer_last_name
      ,c_preferred_cust_flag customer_preferred_cust_flag
      ,c_birth_country customer_birth_country
      ,c_login customer_login
      ,c_email_address customer_email_address
      ,d_year dyear
      ,sum((((ws_ext_list_price-ws_ext_wholesale_cost-ws_ext_discount_amt)+
   ws_ext_sales_price)/2) ) year_total
      ,'w' sale_type
from customer
    ,web_sales
    ,date_dim
where c_customer_sk = ws_bill_customer_sk
  and ws_sold_date_sk = d_date_sk
group by c_customer_id
        ,c_first_name
        ,c_last_name
        ,c_preferred_cust_flag
        ,c_birth_country
        ,c_login
        ,c_email_address
        ,d_year
        )
 select
```

```
                t_s_secyear.customer_id
               ,t_s_secyear.customer_first_name
               ,t_s_secyear.customer_last_name
               ,t_s_secyear.customer_preferred_cust_flag
from year_total t_s_firstyear
    ,year_total t_s_secyear
    ,year_total t_c_firstyear
    ,year_total t_c_secyear
    ,year_total t_w_firstyear
    ,year_total t_w_secyear
where t_s_secyear.customer_id = t_s_firstyear.customer_id
  and t_s_firstyear.customer_id = t_c_secyear.customer_id
  and t_s_firstyear.customer_id = t_c_firstyear.customer_id
  and t_s_firstyear.customer_id = t_w_firstyear.customer_id
  and t_s_firstyear.customer_id = t_w_secyear.customer_id
  and t_s_firstyear.sale_type = 's'
  and t_c_firstyear.sale_type = 'c'
  and t_w_firstyear.sale_type = 'w'
  and t_s_secyear.sale_type = 's'
  and t_c_secyear.sale_type = 'c'
  and t_w_secyear.sale_type = 'w'
  and t_s_firstyear.dyear =  2001
  and t_s_secyear.dyear = 2001+1
  and t_c_firstyear.dyear =  2001
  and t_c_secyear.dyear =  2001+1
  and t_w_firstyear.dyear = 2001
  and t_w_secyear.dyear = 2001+1
  and t_s_firstyear.year_total > 0
  and t_c_firstyear.year_total > 0
  and t_w_firstyear.year_total > 0
  and case when t_c_firstyear.year_total > 0 then t_c_secyear.year_total /
   t_c_firstyear.year_total else null end
          > case when t_s_firstyear.year_total > 0 then t_s_secyear.
   year_total / t_s_firstyear.year_total else null end
  and case when t_c_firstyear.year_total > 0 then t_c_secyear.year_total /
   t_c_firstyear.year_total else null end
          > case when t_w_firstyear.year_total > 0 then t_w_secyear.
   year_total / t_w_firstyear.year_total else null end
order by t_s_secyear.customer_id
        ,t_s_secyear.customer_first_name
```

```
111         ,t_s_secyear.customer_last_name
112         ,t_s_secyear.customer_preferred_cust_flag
113 limit 100;
```

Listing 4.7: Query 4.

It was Optimized by the use of Indexing as shown below:

```
1 CREATE INDEX idx_customer_c_customer_sk ON customer (c_customer_sk);
2 CREATE INDEX idx_customer_c_customer_id ON customer (c_customer_id);
3 CREATE INDEX idx_store_sales_ss_customer_sk ON store_sales (ss_customer_sk);
4 CREATE INDEX idx_store_sales_ss_sold_date_sk ON store_sales (ss_sold_date_sk
    );
5 CREATE INDEX idx_date_dim_d_date_sk ON date_dim (d_date_sk);
6 CREATE INDEX idx_catalog_sales_cs_bill_customer_sk ON catalog_sales (
    cs_bill_customer_sk);
7 CREATE INDEX idx_catalog_sales_cs_sold_date_sk ON catalog_sales (
    cs_sold_date_sk);
8 CREATE INDEX idx_web_sales_ws_bill_customer_sk ON web_sales (
    ws_bill_customer_sk);
9 CREATE INDEX idx_web_sales_ws_sold_date_sk ON web_sales (ws_sold_date_sk);
```

Listing 4.8: Optimized Query 14.

The resulting visual is shown below:

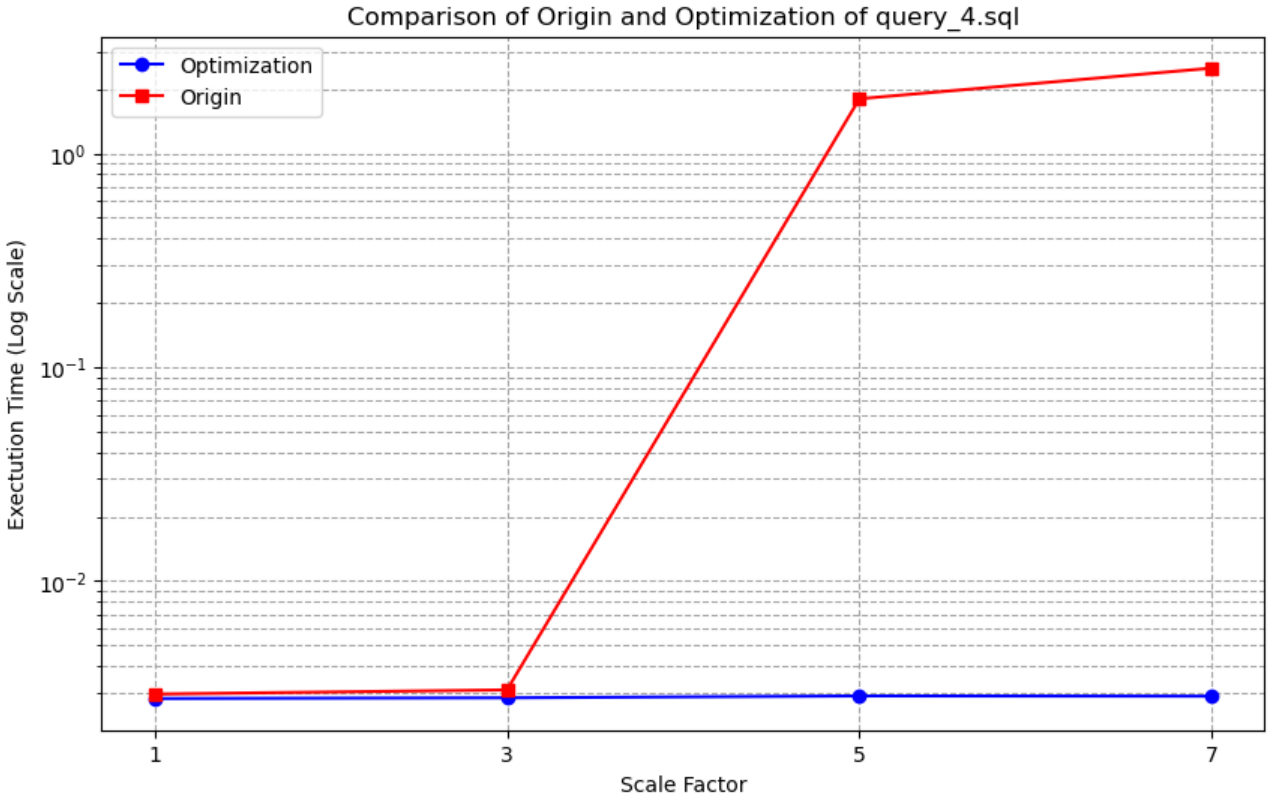


Figure 4.37: Optimization/Comparison of Origin and Optimization of query 4

The other optimized queries are in Appendix A section A

5 Conclusion

5.1 Conclusion

In this project, We have systematically learned the basic theories and concepts of TPC-DS benchmark testing and implemented it on DuckDB, which is an open-source and particularly fast database. We finished the data and query generation. We conducted experiments for four scale factors on eight tests: load test, power test, throughput test 1,throughput test 2, data maintenance test 1,data maintenance test 2, data accessibility test, scalability test. It is evident that the execution time grows linearly with the size of data. We also made performance optimizations on some queries, which reduced their execution runtime. Then we presented and analyzed the results of our experiments in figures and charts with data visualization.

All the results are almost in line with our intuition about the size of the data, i.e., the larger the scale factor, the more resources are needed on the system. Runtimes vary from query to query, depending on the query complexity and the backend processes of the system. Also, we found that the PC equipment has some impact on the runtime, as well. If you're an analyst with a recent Macbook Pro and a small data warehouse, the laptop you use to write SQL queries might be faster than the data warehouse you run them on [Fra23].

It's worth noting that while TPC-DS is a comprehensive benchmark, no benchmark can capture all the nuances of every real-world workload. Therefore, while TPC-DS can provide valuable insights, it should be used in conjunction with other evaluation methodologies when making decisions about system architectures or purchasing decisions.

References

[TPC21] (TPC), Transaction Processing Performance Council (2021). “TPC Benchmark DS Standard Specification”. In.

[Cod23] Code, Visual Studio (2023). “Visual Studio Code - Code Editing”. In: URL: https://code.visualstudio.com/docs.

[Com23] Commuity, Dbeaver (2023). “Dbeaver Community Tool”. In: URL: https://dbeaver.io/.

[Fra23] Fraser, George (2023). “Macbook Pro is impressive”. In: URL: https://www.fivetran.com/blog/how-fast-is-duckdb-really.

[Ham23] Hamilton, Thomas (2023). “What is JMeter? Introduction & Uses”. In: URL: https://www.guru99.com/introduction-to-jmeter.html.

[Koh+22] Kohn, André et al. (2022). “DuckDB-wasm: fast analytical processing for the web”. In: *Proceedings of the VLDB Endowment* 15.12, pp. 3574–3577.

[Mah21] Mahmud, Md Hedayet (2021). “What is benchmarking and how can it be used to measure our current performance against an ideal or best practice model?” In: URL: https://www.quora.com/What-is-benchmarking-and-how-can-it-be-used-to-measure-our-current-performance-against-an-ideal-or-best-practice-model.

[Pyt23] Python (2023). “General Python FAQ”. In: URL: https://https://docs.python.org/3/faq/general.html/.

[RM19] Raasveldt, Mark and Hannes Mühleisen (2019). “DuckDB: an Embeddable Analytical Database”. In: *International Conference on Management of Data (SIGMOD ’19)*. URL: https://dl.acm.org/doi/abs/10.1145/3299869.3320212.

[Sam21] Samy Kabangu, John Ebden (2021). “Benchmarking Databases”. In.

[VZ14] Vaisman, Alejandro and Esteban Zimányi (2014). “Data warehouse systems”. In: *Data-Centric Systems and Applications*.

Appendices

A Appendix A

A.1 Optimized Queries

```
Optimize Query 11

-- Index for store_sales table
CREATE INDEX idx_ss_customer_sk ON store_sales(ss_customer_sk);
CREATE INDEX idx_ss_sold_date_sk ON store_sales(ss_sold_date_sk);

-- Index for customer table
CREATE INDEX idx_c_customer_sk ON customer(c_customer_sk);

-- Index for web_sales table
CREATE INDEX idx_ws_bill_customer_sk ON web_sales(ws_bill_customer_sk);
CREATE INDEX idx_ws_sold_date_sk ON web_sales(ws_sold_date_sk);

-- Index for date_dim table
CREATE INDEX idx_d_date_sk ON date_dim(d_date_sk);
CREATE INDEX idx_d_year ON date_dim(d_year);


with year_total as (
select c_customer_id customer_id
     ,c_first_name customer_first_name
     ,c_last_name customer_last_name
```

```
     ,c_preferred_cust_flag customer_preferred_cust_flag
     ,c_birth_country customer_birth_country
     ,c_login customer_login
     ,c_email_address customer_email_address
     ,d_year dyear
     ,sum(ss_ext_list_price-ss_ext_discount_amt) year_total
     ,'s' sale_type
from customer
   ,store_sales
   ,date_dim
where c_customer_sk = ss_customer_sk
 and ss_sold_date_sk = d_date_sk
group by c_customer_id
       ,c_first_name
       ,c_last_name
       ,c_preferred_cust_flag
       ,c_birth_country
       ,c_login
       ,c_email_address
       ,d_year
union all
select c_customer_id customer_id
     ,c_first_name customer_first_name
     ,c_last_name customer_last_name
     ,c_preferred_cust_flag customer_preferred_cust_flag
     ,c_birth_country customer_birth_country
     ,c_login customer_login
     ,c_email_address customer_email_address
     ,d_year dyear
     ,sum(ws_ext_list_price-ws_ext_discount_amt) year_total
     ,'w' sale_type
from customer
   ,web_sales
   ,date_dim
where c_customer_sk = ws_bill_customer_sk
 and ws_sold_date_sk = d_date_sk
group by c_customer_id
       ,c_first_name
       ,c_last_name
       ,c_preferred_cust_flag
```

```
63        ,c_birth_country
64        ,c_login
65        ,c_email_address
66        ,d_year
67        )
68 select
69                 t_s_secyear.customer_id
70                ,t_s_secyear.customer_first_name
71                ,t_s_secyear.customer_last_name
72                ,t_s_secyear.customer_preferred_cust_flag
73 from year_total t_s_firstyear
74    ,year_total t_s_secyear
75    ,year_total t_w_firstyear
76    ,year_total t_w_secyear
77 where t_s_secyear.customer_id = t_s_firstyear.customer_id
78        and t_s_firstyear.customer_id = t_w_secyear.customer_id
79        and t_s_firstyear.customer_id = t_w_firstyear.customer_id
80        and t_s_firstyear.sale_type = 's'
81        and t_w_firstyear.sale_type = 'w'
82        and t_s_secyear.sale_type = 's'
83        and t_w_secyear.sale_type = 'w'
84        and t_s_firstyear.dyear = 2001
85        and t_s_secyear.dyear = 2001+1
86        and t_w_firstyear.dyear = 2001
87        and t_w_secyear.dyear = 2001+1
88        and t_s_firstyear.year_total > 0
89        and t_w_firstyear.year_total > 0
90        and case when t_w_firstyear.year_total > 0 then t_w_secyear.
    year_total / t_w_firstyear.year_total else 0.0 end
91            > case when t_s_firstyear.year_total > 0 then t_s_secyear.
    year_total / t_s_firstyear.year_total else 0.0 end
92 order by t_s_secyear.customer_id
93        ,t_s_secyear.customer_first_name
94        ,t_s_secyear.customer_last_name
95        ,t_s_secyear.customer_preferred_cust_flag
96 limit 100;
97
98 ---------------------------------------------------------------------------------
```

Listing 1: Optimized Query 11.

```
1
2  Optimized Query 14
3
4  CREATE INDEX idx_item_i_item_sk ON item (i_item_sk);
5  CREATE INDEX idx_date_dim_d_date_sk ON date_dim (d_date_sk);
6  CREATE INDEX idx_store_sales_ss_item_sk ON store_sales (ss_item_sk);
7  CREATE INDEX idx_catalog_sales_cs_item_sk ON catalog_sales (cs_item_sk);
8  CREATE INDEX idx_web_sales_ws_item_sk ON web_sales (ws_item_sk);
9
10
11 CREATE INDEX idx_item_i_brand_id ON item (i_brand_id);
12 CREATE INDEX idx_item_i_class_id ON item (i_class_id);
13 CREATE INDEX idx_item_i_category_id ON item (i_category_id);
14
15
16 with  cross_items as
17 (select i_item_sk ss_item_sk
18 from item,
19 (select iss.i_brand_id brand_id
20     ,iss.i_class_id class_id
21     ,iss.i_category_id category_id
22 from store_sales
23     ,item iss
24     ,date_dim d1
25 where ss_item_sk = iss.i_item_sk
26   and ss_sold_date_sk = d1.d_date_sk
27   and d1.d_year between 1999 AND 1999 + 2
28 intersect
29 select ics.i_brand_id
30     ,ics.i_class_id
31     ,ics.i_category_id
32 from catalog_sales
33     ,item ics
34     ,date_dim d2
35 where cs_item_sk = ics.i_item_sk
36   and cs_sold_date_sk = d2.d_date_sk
37   and d2.d_year between 1999 AND 1999 + 2
38 intersect
39 select iws.i_brand_id
40     ,iws.i_class_id
```

```
    ,iws.i_category_id
from web_sales
    ,item iws
    ,date_dim d3
where ws_item_sk = iws.i_item_sk
  and ws_sold_date_sk = d3.d_date_sk
  and d3.d_year between 1999 AND 1999 + 2) x
where i_brand_id = brand_id
     and i_class_id = class_id
     and i_category_id = category_id
),
avg_sales as
(select avg(quantity*list_price) average_sales
 from (select ss_quantity quantity
            ,ss_list_price list_price
      from store_sales
          ,date_dim
      where ss_sold_date_sk = d_date_sk
        and d_year between 1999 and 1999 + 2
      union all
      select cs_quantity quantity
            ,cs_list_price list_price
      from catalog_sales
          ,date_dim
      where cs_sold_date_sk = d_date_sk
        and d_year between 1999 and 1999 + 2
      union all
      select ws_quantity quantity
            ,ws_list_price list_price
      from web_sales
          ,date_dim
      where ws_sold_date_sk = d_date_sk
        and d_year between 1999 and 1999 + 2) x)
 select  this_year.channel ty_channel
                          ,this_year.i_brand_id ty_brand
                          ,this_year.i_class_id ty_class
                          ,this_year.i_category_id ty_category
                          ,this_year.sales ty_sales
                          ,this_year.number_sales ty_number_sales
                          ,last_year.channel ly_channel
```

```
                          ,last_year.i_brand_id ly_brand
                          ,last_year.i_class_id ly_class
                          ,last_year.i_category_id ly_category
                          ,last_year.sales ly_sales
                          ,last_year.number_sales ly_number_sales
from
(select 'store' channel, i_brand_id,i_class_id,i_category_id
       ,sum(ss_quantity*ss_list_price) sales, count(*) number_sales
from store_sales
    ,item
    ,date_dim
where ss_item_sk in (select ss_item_sk from cross_items)
  and ss_item_sk = i_item_sk
  and ss_sold_date_sk = d_date_sk
  and d_week_seq = (select d_week_seq
                    from date_dim
                    where d_year = 1999 + 1
                      and d_moy = 12
                      and d_dom = 3)
group by i_brand_id,i_class_id,i_category_id
having sum(ss_quantity*ss_list_price) > (select average_sales from avg_sales
   )) this_year,
(select 'store' channel, i_brand_id,i_class_id
       ,i_category_id, sum(ss_quantity*ss_list_price) sales, count(*)
   number_sales
from store_sales
    ,item
    ,date_dim
where ss_item_sk in (select ss_item_sk from cross_items)
  and ss_item_sk = i_item_sk
  and ss_sold_date_sk = d_date_sk
  and d_week_seq = (select d_week_seq
                    from date_dim
                    where d_year = 1999
                      and d_moy = 12
                      and d_dom = 3)
group by i_brand_id,i_class_id,i_category_id
having sum(ss_quantity*ss_list_price) > (select average_sales from avg_sales
   )) last_year
where this_year.i_brand_id= last_year.i_brand_id
```

```
  and this_year.i_class_id = last_year.i_class_id
  and this_year.i_category_id = last_year.i_category_id
order by this_year.channel, this_year.i_brand_id, this_year.i_class_id,
   this_year.i_category_id
limit 100;

-------------------------------------------------------------------------------
```

Listing 2: Optimized Query 14.

```
--Optimized Query 14b

CREATE INDEX idx_item_i_item_sk ON item (i_item_sk);
CREATE INDEX idx_date_dim_d_date_sk ON date_dim (d_date_sk);
CREATE INDEX idx_store_sales_ss_item_sk ON store_sales (ss_item_sk);
CREATE INDEX idx_catalog_sales_cs_item_sk ON catalog_sales (cs_item_sk);
CREATE INDEX idx_web_sales_ws_item_sk ON web_sales (ws_item_sk);


CREATE INDEX idx_item_i_brand_id ON item (i_brand_id);
CREATE INDEX idx_item_i_class_id ON item (i_class_id);
CREATE INDEX idx_item_i_category_id ON item (i_category_id);




with  cross_items as
(select i_item_sk ss_item_sk
from item,
(select iss.i_brand_id brand_id
    ,iss.i_class_id class_id
    ,iss.i_category_id category_id
from store_sales
    ,item iss
    ,date_dim d1
where ss_item_sk = iss.i_item_sk
  and ss_sold_date_sk = d1.d_date_sk
  and d1.d_year between 1999 AND 1999 + 2
intersect
select ics.i_brand_id
    ,ics.i_class_id
```

```
    ,ics.i_category_id
from catalog_sales
    ,item ics
    ,date_dim d2
where cs_item_sk = ics.i_item_sk
  and cs_sold_date_sk = d2.d_date_sk
  and d2.d_year between 1999 AND 1999 + 2
intersect
select iws.i_brand_id
    ,iws.i_class_id
    ,iws.i_category_id
from web_sales
    ,item iws
    ,date_dim d3
where ws_item_sk = iws.i_item_sk
  and ws_sold_date_sk = d3.d_date_sk
  and d3.d_year between 1999 AND 1999 + 2)
where i_brand_id = brand_id
     and i_class_id = class_id
     and i_category_id = category_id
),
avg_sales as
(select avg(quantity*list_price) average_sales
 from (select ss_quantity quantity
            ,ss_list_price list_price
      from store_sales
          ,date_dim
      where ss_sold_date_sk = d_date_sk
        and d_year between 1999 and 1999 + 2
      union all
      select cs_quantity quantity
            ,cs_list_price list_price
      from catalog_sales
          ,date_dim
      where cs_sold_date_sk = d_date_sk
        and d_year between 1999 and 1999 + 2
      union all
      select ws_quantity quantity
            ,ws_list_price list_price
      from web_sales
```

```
71          ,date_dim
72      where ws_sold_date_sk = d_date_sk
73        and d_year between 1999 and 1999 + 2) x)
74 select  channel, i_brand_id,i_class_id,i_category_id,sum(sales), sum(
   number_sales)
75 from(
76      select 'store' channel, i_brand_id,i_class_id
77            ,i_category_id,sum(ss_quantity*ss_list_price) sales
78            , count(*) number_sales
79      from store_sales
80          ,item
81          ,date_dim
82      where ss_item_sk in (select ss_item_sk from cross_items)
83        and ss_item_sk = i_item_sk
84        and ss_sold_date_sk = d_date_sk
85        and d_year = 1999+2
86        and d_moy = 11
87      group by i_brand_id,i_class_id,i_category_id
88      having sum(ss_quantity*ss_list_price) > (select average_sales from
   avg_sales)
89      union all
90      select 'catalog' channel, i_brand_id,i_class_id,i_category_id, sum(
   cs_quantity*cs_list_price) sales, count(*) number_sales
91      from catalog_sales
92          ,item
93          ,date_dim
94      where cs_item_sk in (select ss_item_sk from cross_items)
95        and cs_item_sk = i_item_sk
96        and cs_sold_date_sk = d_date_sk
97        and d_year = 1999+2
98        and d_moy = 11
99      group by i_brand_id,i_class_id,i_category_id
100     having sum(cs_quantity*cs_list_price) > (select average_sales from
   avg_sales)
101     union all
102     select 'web' channel, i_brand_id,i_class_id,i_category_id, sum(
   ws_quantity*ws_list_price) sales , count(*) number_sales
103     from web_sales
104         ,item
105         ,date_dim
```

```
      where ws_item_sk in (select ss_item_sk from cross_items)
        and ws_item_sk = i_item_sk
        and ws_sold_date_sk = d_date_sk
        and d_year = 1999+2
        and d_moy = 11
      group by i_brand_id,i_class_id,i_category_id
      having sum(ws_quantity*ws_list_price) > (select average_sales from
   avg_sales)
) y
group by rollup (channel, i_brand_id,i_class_id,i_category_id)
--order by channel,i_brand_id,i_class_id,i_category_id
ORDER BY channel NULLS FIRST,
        i_brand_id NULLS FIRST,
        i_class_id NULLS FIRST,
        i_category_id NULLS FIRST
limit 100;

-------------------------------------------------------------------------------
```

Listing 3: Optimized Query 14b.

```
CREATE TEMP TABLE tmp_avg_qoh AS
SELECT i_product_name, i_brand, i_class, i_category, AVG(
   inv_quantity_on_hand) AS qoh
FROM inventory
JOIN date_dim ON inv_date_sk = d_date_sk
JOIN item ON inv_item_sk = i_item_sk
WHERE d_month_seq BETWEEN 1200 AND 1200 + 11
GROUP BY i_product_name, i_brand, i_class, i_category;

SELECT i_product_name, i_brand, i_class, i_category, qoh
FROM tmp_avg_qoh
ORDER BY roqoh, i_product_name, i_brand, i_class, i_category
LIMIT 100;

OR THIS

CREATE INDEX idx_inv_date_sk ON inventory (inv_date_sk);
CREATE INDEX idx_inv_item_sk ON inventory (inv_item_sk);

```

```
select  i_product_name
            ,i_brand
            ,i_class
            ,i_category
            ,avg(inv_quantity_on_hand) qoh
      from inventory
          ,date_dim
          ,item
      where inv_date_sk=d_date_sk
             and inv_item_sk=i_item_sk
             and d_month_seq between 1200 and 1200 + 11
      group by rollup(i_product_name
                      ,i_brand
                      ,i_class
                      ,i_category)
order by qoh, i_product_name, i_brand, i_class, i_category
limit 100;
```

Listing 4: Optimized Query 22.

```
CREATE TEMPORARY TABLE frequent_ss_items AS
SELECT
    SUBSTR(i_item_desc, 1, 30) AS itemdesc,
    i_item_sk AS item_sk,
    d_date AS solddate,
    COUNT(*) AS cnt
FROM
    store_sales
JOIN
    date_dim ON ss_sold_date_sk = d_date_sk
JOIN
    item ON ss_item_sk = i_item_sk
WHERE
    d_year IN (2000, 2001, 2002, 2003)
GROUP BY
    SUBSTR(i_item_desc, 1, 30), i_item_sk, d_date
HAVING COUNT(*) > 4;

-- Create an index on the temporary table
CREATE INDEX idx_frequent_items ON frequent_ss_items (item_sk);

```

```
-- Create a temporary table for maximum store sales
CREATE TEMPORARY TABLE max_store_sales AS
SELECT MAX(csales) AS tpcds_cmax
FROM (
    SELECT
        c_customer_sk,
        SUM(ss_quantity * ss_sales_price) AS csales
    FROM
        store_sales
    JOIN
        customer ON ss_customer_sk = c_customer_sk
    JOIN
        date_dim ON ss_sold_date_sk = d_date_sk
    WHERE
        d_year IN (2000, 2001, 2002, 2003)
    GROUP BY
        c_customer_sk
);

-- Create a temporary table for the best store sales customers
CREATE TEMPORARY TABLE best_ss_customer AS
SELECT
    c_customer_sk,
    SUM(ss_quantity * ss_sales_price) AS ssales
FROM
    store_sales
JOIN
    customer ON ss_customer_sk = c_customer_sk
GROUP BY
    c_customer_sk
HAVING
    SUM(ss_quantity * ss_sales_price) > (50/100.0) * (SELECT tpcds_cmax FROM
    max_store_sales);

SELECT
    c_last_name,
    c_first_name,
    sales
FROM (
    SELECT
```

```
61          c_last_name,
62          c_first_name,
63          SUM(cs_quantity * cs_list_price) AS sales
64      FROM
65          catalog_sales
66      JOIN
67          customer ON cs_bill_customer_sk = c_customer_sk
68      JOIN
69          date_dim ON cs_sold_date_sk = d_date_sk
70      WHERE
71          d_year = 2000
72      AND
73          d_moy = 2
74      AND
75          cs_item_sk IN (SELECT item_sk FROM frequent_ss_items)
76      AND
77          cs_bill_customer_sk IN (SELECT c_customer_sk FROM best_ss_customer)
78      AND
79          cs_bill_customer_sk = c_customer_sk
80      GROUP BY
81          c_last_name, c_first_name
82      UNION ALL
83      SELECT
84          c_last_name,
85          c_first_name,
86          SUM(ws_quantity * ws_list_price) AS sales
87      FROM
88          web_sales
89      JOIN
90          customer ON ws_bill_customer_sk = c_customer_sk
91      JOIN
92          date_dim ON ws_sold_date_sk = d_date_sk
93      WHERE
94          d_year = 2000
95      AND
96          d_moy = 2
97      AND
98          ws_item_sk IN (SELECT item_sk FROM frequent_ss_items)
99      AND
100         ws_bill_customer_sk IN (SELECT c_customer_sk FROM best_ss_customer)
```

```
    AND
        ws_bill_customer_sk = c_customer_sk
    GROUP BY
        c_last_name, c_first_name
) AS final_results
ORDER BY
    c_last_name, c_first_name, sales
LIMIT 100;
-------------------------------------------------------------------------------
```

Listing 5: Optimized Query 23.

```
--Optimized Query 47

-- Create a temporary table for CTE v1
CREATE TEMP TABLE temp_v1 AS
SELECT i_category, i_brand,
       s_store_name, s_company_name,
       d_year, d_moy,
       SUM(ss_sales_price) sum_sales,
       AVG(SUM(ss_sales_price)) OVER
         (PARTITION BY i_category, i_brand,
                     s_store_name, s_company_name, d_year)
         avg_monthly_sales,
       RANK() OVER/
         (PARTITION BY i_category, i_brand,
                     s_store_name, s_company_name
          ORDER BY d_year, d_moy) rn
FROM item, store_sales, date_dim, store
WHERE ss_item_sk = i_item_sk AND
      ss_sold_date_sk = d_date_sk AND
      ss_store_sk = s_store_sk AND
      (
        d_year = 1999 or
        (d_year = 1999 - 1 and d_moy = 12) or
        (d_year = 1999 + 1 and d_moy = 1)
      )
GROUP BY i_category, i_brand,
         s_store_name, s_company_name,
         d_year, d_moy;
```

```

-- Create a temporary table for CTE v2
CREATE TEMP TABLE temp_v2 AS
SELECT v1.i_category, v1.i_brand, v1.s_store_name, v1.s_company_name, v1.
   s_store_name, v1.s_company_name
       ,v1.d_year
       ,v1.avg_monthly_sales
       ,v1.sum_sales, v1_lag.sum_sales psum, v1_lead.sum_sales nsum,v1.
   d_year, v1.d_moy
FROM temp_v1 v1
JOIN temp_v1 v1_lag ON v1.i_category = v1_lag.i_category and
                       v1.i_brand = v1_lag.i_brand and
                       v1.s_store_name = v1_lag.s_store_name and
                       v1.s_company_name = v1_lag.s_company_name and
                       v1.rn = v1_lag.rn + 1
JOIN temp_v1 v1_lead ON v1.i_category = v1_lead.i_category and
                        v1.i_brand = v1_lead.i_brand and
                        v1.s_store_name = v1_lead.s_store_name and
                        v1.s_company_name = v1_lead.s_company_name and
                        v1.rn = v1_lead.rn - 1;

-- Retrieve the final result
SELECT *
FROM temp_v2
WHERE d_year = 1999 and
      avg_monthly_sales > 0 and
      CASE WHEN avg_monthly_sales > 0 THEN ABS(sum_sales - avg_monthly_sales
   ) / avg_monthly_sales ELSE NULL END > 0.1
ORDER BY s_store_name
LIMIT 100;


--------------------------------------------------------------------------------

```

Listing 6: Optimized Query 47.

```
--Optimized Query 57


-- Create a temporary table for CTE v1
```

```
CREATE TEMP TABLE temp_v11 AS
SELECT i_category, i_brand, cc_name, d_year, d_moy,
      SUM(cs_sales_price) sum_sales,
      AVG(SUM(cs_sales_price)) OVER
        (PARTITION BY i_category, i_brand, cc_name, d_year)
        avg_monthly_sales,
      RANK() OVER
        (PARTITION BY i_category, i_brand, cc_name
         ORDER BY d_year, d_moy) rn
FROM item, catalog_sales, date_dim, call_center
WHERE cs_item_sk = i_item_sk AND
     cs_sold_date_sk = d_date_sk AND
     cc_call_center_sk = cs_call_center_sk AND
     (
       d_year = 1999 or
       (d_year = 1999 - 1 and d_moy = 12) or
       (d_year = 1999 + 1 and d_moy = 1)
     )
GROUP BY i_category, i_brand, cc_name, d_year, d_moy;

--Create a temporary table for CTE v2
CREATE TEMP TABLE temp_v22 AS
SELECT v11.i_category, v11.i_brand, v11.cc_name
      ,v11.d_year
      ,v11.avg_monthly_sales
      ,v11.sum_sales, v1_lag.sum_sales psum, v1_lead.sum_sales nsum,
      v11.i_category, v11.i_brand, v11.cc_name, v11.d_year, v11.d_moy
FROM temp_v11 v11
JOIN temp_v11 v1_lag ON v11.i_category = v1_lag.i_category and
                    v11.i_brand = v1_lag.i_brand and
                    v11.cc_name = v1_lag.cc_name and
                    v11.rn = v1_lag.rn + 1
JOIN temp_v11 v1_lead ON v11.i_category = v1_lead.i_category and
                     v11.i_brand = v1_lead.i_brand and
                     v11.cc_name = v1_lead.cc_name and
                     v11.rn = v1_lead.rn - 1;

--Retrieve the final result
SELECT *
FROM temp_v22
```

```
WHERE d_year = 1999 and
     avg_monthly_sales > 0 and
     CASE WHEN avg_monthly_sales > 0 THEN ABS(sum_sales - avg_monthly_sales)
    / avg_monthly_sales ELSE NULL END > 0.1
ORDER BY cc_name
LIMIT 100;

-------------------------------------------------------------------------------
```

Listing 7: Optimized Query 57.

```
--OPtimized Query 64

Indexes for Join Conditions
CREATE INDEX idx_cs_item_sk ON catalog_sales (cs_item_sk);
CREATE INDEX idx_cr_item_sk ON catalog_returns (cr_item_sk);

CREATE INDEX idx_customer_join ON customer (c_customer_sk,
   c_current_cdemo_sk, c_current_hdemo_sk, c_current_addr_sk);
CREATE INDEX idx_cd_join ON customer_demographics (cd_demo_sk,
   cd_marital_status);

--CREATE INDEX idx_ca_address_sk ON customer_address (ca_address_sk);

CREATE INDEX idx_promotion_promo_sk ON promotion (p_promo_sk);

CREATE INDEX idx_hd_income_band_sk ON household_demographics (
   hd_income_band_sk);

CREATE INDEX idx_ib_income_band_sk ON income_band (ib_income_band_sk);

CREATE INDEX idx_item_join ON item (i_item_sk, i_color, i_current_price);

-- Indexes for Aggregation and Filtering
CREATE INDEX idx_cs_ui_item_sk ON cs_ui (cs_item_sk);



with cs_ui as
(select cs_item_sk
```

```
      ,sum(cs_ext_list_price) as sale,sum(cr_refunded_cash+
   cr_reversed_charge+cr_store_credit) as refund
 from catalog_sales
     ,catalog_returns
 where cs_item_sk = cr_item_sk
   and cs_order_number = cr_order_number
 group by cs_item_sk
 having sum(cs_ext_list_price)>2*sum(cr_refunded_cash+cr_reversed_charge+
   cr_store_credit)),
cross_sales as
(select i_product_name product_name
    ,i_item_sk item_sk
    ,s_store_name store_name
    ,s_zip store_zip
    ,ad1.ca_street_number b_street_number
    ,ad1.ca_street_name b_street_name
    ,ad1.ca_city b_city
    ,ad1.ca_zip b_zip
    ,ad2.ca_street_number c_street_number
    ,ad2.ca_street_name c_street_name
    ,ad2.ca_city c_city
    ,ad2.ca_zip c_zip
    ,d1.d_year as syear
    ,d2.d_year as fsyear
    ,d3.d_year s2year
    ,count(*) cnt
    ,sum(ss_wholesale_cost) s1
    ,sum(ss_list_price) s2
    ,sum(ss_coupon_amt) s3
 FROM   store_sales
       ,store_returns
       ,cs_ui
       ,date_dim d1
       ,date_dim d2
       ,date_dim d3
       ,store
       ,customer
       ,customer_demographics cd1
       ,customer_demographics cd2
       ,promotion
```

```
      ,household_demographics hd1
      ,household_demographics hd2
      ,customer_address ad1
      ,customer_address ad2
      ,income_band ib1
      ,income_band ib2
      ,item
 WHERE  ss_store_sk = s_store_sk AND
        ss_sold_date_sk = d1.d_date_sk AND
        ss_customer_sk = c_customer_sk AND
        ss_cdemo_sk= cd1.cd_demo_sk AND
        ss_hdemo_sk = hd1.hd_demo_sk AND
        ss_addr_sk = ad1.ca_address_sk and
        ss_item_sk = i_item_sk and
        ss_item_sk = sr_item_sk and
        ss_ticket_number = sr_ticket_number and
        ss_item_sk = cs_ui.cs_item_sk and
        c_current_cdemo_sk = cd2.cd_demo_sk AND
        c_current_hdemo_sk = hd2.hd_demo_sk AND
        c_current_addr_sk = ad2.ca_address_sk and
        c_first_sales_date_sk = d2.d_date_sk and
        c_first_shipto_date_sk = d3.d_date_sk and
        ss_promo_sk = p_promo_sk and
        hd1.hd_income_band_sk = ib1.ib_income_band_sk and
        hd2.hd_income_band_sk = ib2.ib_income_band_sk and
        cd1.cd_marital_status <> cd2.cd_marital_status and
        i_color in ('purple','burlywood','indian','spring','floral','meduim'
   ) and
        i_current_price between 64 and 64 + 10 and
        i_current_price between 64 + 1 and 64 + 15
group by i_product_name
      ,i_item_sk
      ,s_store_name
      ,s_zip
      ,ad1.ca_street_number
      ,ad1.ca_street_name
      ,ad1.ca_city
      ,ad1.ca_zip
      ,ad2.ca_street_number
      ,ad2.ca_street_name
```

```
      ,ad2.ca_city
      ,ad2.ca_zip
      ,d1.d_year
      ,d2.d_year
      ,d3.d_year
)
select cs1.product_name
    ,cs1.store_name
    ,cs1.store_zip
    ,cs1.b_street_number
    ,cs1.b_street_name
    ,cs1.b_city
    ,cs1.b_zip
    ,cs1.c_street_number
    ,cs1.c_street_name
    ,cs1.c_city
    ,cs1.c_zip
    ,cs1.syear
    ,cs1.cnt
    ,cs1.s1 as s11
    ,cs1.s2 as s21
    ,cs1.s3 as s31
    ,cs2.s1 as s12
    ,cs2.s2 as s22
    ,cs2.s3 as s32
    ,cs2.syear
    ,cs2.cnt
from cross_sales cs1,cross_sales cs2
where cs1.item_sk=cs2.item_sk and
    cs1.syear = 1999 and
    cs2.syear = 1999 + 1 and
    cs2.cnt <= cs1.cnt and
    cs1.store_name = cs2.store_name and
    cs1.store_zip = cs2.store_zip
order by cs1.product_name
      ,cs1.store_name
      ,cs2.cnt
      ,cs1.s1
      ,cs2.s1;

```

144

Listing 8: Optimized Query 64.