Data Warehouse

A TPC-DS Benchmark Implementation Using Oracle

Aayush Paudel (000583200)

MD Kamrul Islam (000583487)

Shofiyyah Nadhiroh (000583951)

Sony Shrestha (000583110)

UNIVERSITÉ LIBRE DE BRUXELLES

ÉCOLE POLYTECHNIQUE DE BRUXELLES

Fall 2023

Table of Contents

Chapter 1 - Overview

1.1 Data Warehouse

Data warehouse is a collection of subject-oriented, integrated, nonvolatile, and time-varying data to support management decisions. **Subject oriented** data warehouse means that a data warehouse focuses on one or more areas of analysis based on the analytical needs of managers at different stages of the decision-making process. **Integrated** refers to the fact that the contents of a data warehouse are the result of data integration from numerous operational and external systems. **Nonvolatile** means that a data warehouse stores data from operating systems for an extended length of time. Thus, in data warehouses, data modification and removal are not permitted, and the only activity permitted is the purging of old data that is no longer required. Finally, **time varying** emphasizes that a data warehouse maintains track of how its data has changed over time (Vaisman and Zimányi).

The primary purpose of a data warehouse is to provide a central repository for data that can be used for analysis, reporting, and business intelligence purposes. A data warehouse is a critical component that supports the data needs of decision support systems. It provides a centralized, well-organized, and historical source of data that enables Decision Support System tools to extract insights and facilitate informed decision-making within an organization.

1.2 Decision Support System

In the ever-changing data-driven business landscape, decision-support systems, or DSS, have become more and more important. Interactive information systems, or DSSs, are skilled at supporting complex decision-making activities with a blend of raw data, models, and analytical tools. Large volumes of unstructured data are transformed into useful insights that guide tactical, operational, and strategic choices. DSSs are essential in situations where businesses struggle with big datasets, as is common with data warehousing projects like TPC-DS. These technologies enable firms to foresee and prepare for future circumstances by providing forecasting capabilities in addition to historical information. When based on dependable, performance-driven DBMSs like Oracle, the resilience of DSSs is further increased. The complex relationship between data warehousing, DSSs, and performance benchmarking emphasizes the need to explore technologies like TPC-DS in modern business landscapes (Grey & Watson, 1998).

1.3 Data Warehouse using Oracle Database

The concept of a Data Warehouse based on the Oracle Database arose from the need to efficiently handle enormous amounts of data while providing relevant insights to organizations. Oracle's database solutions have been at the forefront of this domain, providing robust scalability, performance, and integrated data management features that are appropriate for both old and novel data types. Oracle's Data Warehouse strategy is built on the capabilities of high-performance query processing, simple data integration, and adaptive optimization to manage a wide range of workloads. Oracle Database's powerful features ensure that businesses can gain quick, accurate, and relevant insights from their data. Furthermore, Oracle continues to improve its data warehousing capabilities, including in-memory processing, advanced analytics, and autonomous operations, ensuring that organizations can adapt and thrive in an increasingly data-driven world [Oracle, 2021].

Chapter 2 - The TPC-DS Benchmarking

2.1 Overview

The TPC Benchmark™DS (TPC-DS) is a benchmark specifically designed to simulate and evaluate the performance of decision support systems. It encompasses several essential components of such systems, such as queries and data management. The benchmark offers a comprehensive assessment of the performance of the System Under Test (SUT) as a decision support system with broad applicability.

This benchmark demonstrates decision support systems that:

- Analyze substantial amounts of data.
- Provide solutions to practical business inquiries.
- Perform queries of diverse operational needs and intricacies (e.g., spontaneous, reporting, iterative OLAP, data mining).
- Exhibit high levels of CPU and IO utilization.
- Undergo periodic synchronization with source OLTP databases using database maintenance functions.
- Operates on "Big Data" solutions, including both RDBMS and Hadoop/Spark based systems.

TPC benchmarks serve the purpose of delivering pertinent and unbiased performance statistics to users in the industry. In order to fulfill this objective, the TPC benchmark specifications mandate that benchmark tests must be conducted using systems, products, technologies, and pricing that meet the following criteria:

- They must be readily accessible to users in general.
- They must be applicable to the specific market segment that the TPC benchmark represents. For example, the TPC-DS benchmark represents complex, high data volume, decision support environments.
- They must be realistically implementable by a substantial number of users within the market segment that the benchmark models or represents.

A benchmark result quantifies the time it takes to respond to queries in single user mode, the rate at which queries may be processed in multi user mode, and the performance of data maintenance tasks. This measurement is done for a specific combination of hardware, operating system, and data processing system configuration, under a controlled and sophisticated workload that simulates multiple users making decisions based on the data.

2.2 Business and Benchmark Model

TPC-DS simulates the decision support functions of a retail product supplier. The supporting schema includes crucial business data, such as customer, order, and product information. The benchmark represents the two most crucial elements of any mature decision support system:

1. User queries, which translate operational fact into business intelligence.
2. Data maintenance, which synchronizes the management analysis process with the operational external data source on which it relies.

TPC-DS's business environment modeling is divided into three primary categories:

- Data Model and Data Access Assumptions

    1. TPC-DS simulates a system for handling long and complex queries, assuming that the data processing system remains quiet for queries at any given time.
    2. TPC-DS data periodically tracks the state of an operational database through maintenance functions, which can modify parts of the decision support system with some delay.
    3. TPC-DS uses a snowflake schema, which includes multiple dimension and fact tables. Each dimension has a unique surrogate key, and fact tables connect to dimensions using these surrogate keys.
    4. To balance performance and consistency, the system administrator can establish locking levels and scheduling rules for queries and data maintenance functions once and for all.
    5. The size of a DSS system, specifically the data it contains, can vary between companies and over different time periods. As a result, the TPC-DS benchmark models various DSS sizes, referred to as benchmark scaling or scale factors.

- Query and User Model Assumptions

    The benchmark's modeled users and queries exhibit the following characteristics:

    1. They solve complex business issues.
    2. They employ diverse access patterns, query formulations, operators, and answer set constraints.
    3. They employ variable query parameters across query executions.

    TPC-DS employs a generalized query model to address the vast variety of query types and user behaviors encountered by a decision support system. This model enables the benchmark to capture essential characteristics of the interactive, iterative nature of on-line analytical processing (OLAP) queries, the longer-running, complex queries of data mining and knowledge discovery, and the more predictable behavior of well-known report queries.

- Data Maintenance Assumptions

  A data warehouse is only as reliable and up-to-date as the operational data upon which it is built. As a result, it is critical to move data from operational OLTP systems to analytical DSS systems. The migration varies significantly from business to business and application to application. Previous benchmarks only included the data analysis portion of decision support systems, leaving out a practical data refresh procedure. TPC-DS provides a more fair perspective.

  Three different and significant steps are typically included in decision support database refresh processes:

  1. During the data extraction stage, accurate data extraction from OLTP databases in use and other important data sources is done.
  2. Data transformation refers to the process of cleansing and manipulating retrieved data into a standardized format that is appropriate for integration into the decision support database.
  3. Data Load refers to the process of directly inserting, modifying, or deleting data within the decision support database.

2.3 Logical Database Design and Scaling and Database Population

The majority of modern decision support systems (DSS) use a star schema data model. A star schema consists of a large fact table and a number of small dimension (lookup) tables. TPC-DS' data model consists of multiple snowflake schemas interconnected by shared dimensions. In a snowflake schema, dimensions can have relationships with other dimensions in addition to their relationship with the fact table.

The TPC-DS schema represents the sales and sales returns process of an organization that utilizes three main sales channels: physical stores, catalogs, and online platforms. The schema comprises seven fact tables:

- A pair of fact tables focused on the product sales and returns for each of the three channels
- A single fact table that models inventory for the catalog and internet sales channels.

Furthermore, the schema includes 17 dimension tables that are linked to all sales channels. The following sentences define the logical structure of each table:

- The name of the table, along with its abbreviation (listed parenthetically)
- A logical diagram of each fact table and its related dimension tables

- The high-level definitions for each table and its relationship to other tables
- The scaling and cardinality information for each column

The TPC-DS benchmark specifies a set of discrete scaling points ("scale factors") based on the size of the raw data produced by dsdgen. Depending on the hardware and software platforms, the actual number of bytes may vary. 1TB, 3TB, 10TB, 30TB, and 100TB are the scale factors specified for TPC-DS. However, due to storage and computing limitations, we chose to conduct our experiments with much smaller scale factors: 1GB, 3GB, 5GB, and 10GB.

2.4 Execution Model

2.4.1 Load Test

The Load Test is defined as all activities necessary to bring the System Under Test to the configuration immediately preceding the start of the Performance Test. The Load Test must not execute any of the queries from the Power Test, Throughput Test, or similar queries. Generation and loading of the data can be accomplished in two ways:

1. **Load from flat files:** dsdgen is used to generate flat files that are stored in or copied to a location on the SUT or external storage that is distinct from the Database Location, implying that this data is a copy of the TPC-DS data. These files' records can optionally be permuted and transferred to the SUT or external storage. Data from these flat files must be loaded into the Database Location prior to benchmark execution. Only the loading into the Database Location contributes to the database load time in this situation.
2. **In-line load:** dsdgen is used to generate data that is loaded directly into the Database Location via a "inline" load facility. In this situation, both generation and loading occur concurrently, adding to the database load time.

The Load Test starts when the first character is read from any of the flat files. It involves creating tables, loading data, creating auxiliary structures, defining and validating constraints, gathering statistics, and configuring the system in order to bring it to the state immediately preceding the first query execution.

2.4.2 Power Test

The Power Test is performed immediately after the load test. The Power Test assesses the system's capacity to execute a series of queries in the shortest amount of time in a single stream format. The Power Test elapsed time is the difference between:

- **Power Test Start Time**, which is the timestamp that must be captured before the first character of the executable query text of the first query of Stream 0 is sent to the SUT by the driver; and
- **Power Test End Time**, which is the timestamp that must be captured after the last character of output data from the last query of Stream 0 is received by the driver from the SUT.

2.4.3 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. Throughput Test 1 immediately follows the Power Test. For a given query template t, used to produce the i th query within query stream s, the query elapsed time, QD(s, i, t), is the difference between:

- The timestamp when the first character of the executable query text is submitted to the SUT by the driver;
- The timestamp when the last character of the output is returned from the SUT to the driver and a success message is sent to the driver.

2.5 Data Maintenance

As part of the benchmark execution, data maintenance operations are performed. These operations consist of refresh run processing.The total number of refresh iterations in the benchmark is equal to the number of query threads in a single Throughput Test. Insertion and deletion operations are specified in pseudo code by data maintenance functions. There are three methods based on the operation they execute and the type of table on which they operate. The following are:

- Fact insert data maintenance
- Fact delete data maintenance
- Inventory delete data maintenance

This test has not been implemented for our project.

Chapter 3 - Implementation Details

3.1 Setting up Oracle Database

Some members of our group were using Windows Operating System while others were using Mac. Since the Oracle database could directly be installed on the Windows Operating System, results obtained on a Windows-based computer with an AMD Ryzen 5 5600H with Radeon Graphics and 16 GB of RAM were used. As for the database, we utilized Oracle Database 21c Enterprise Edition as our tool.

3.2 Generating and Loading the Data

TPC-DS featured a program known as dsdgen, enabling users to generate data. Data was generated for the scale factor of 1, 3, 5 and 10 using the following command.

```
.\dsdgen.exe /SCALE 1 /FORCE Y /DIR ../output/generated_data/sf1 /VERBOSE Y
```

3.3 Generating queries

TPC-DS featured a program known as dsdgen, enabling users to generate queries. Queries were generated using scale factor 1 using the following command.

```
.\dsqgen.exe /DIRECTORY ../query_templates /INPUT ../query_templates/templates.lst /VERBOSE Y
/QUALIFY Y /SCALE 1 /DIALECT oracle /OUTPUT_DIR ../generated_query/sf1
```

While generating the query template, an error was observed mentioning that the variable _END was being used before substitution. In order to resolve this issue, we added define _END = ""; line at the end of query_templates/oracle.tpl file.

```
ERROR:
Substitution'_END' is used before being initialized at line 63 in ../query_templates/query1.tpl
```

3.4 Running queries

We prepared a python script in order to execute all 99 queries one after another in a single execution. Run time was dumped into the excel file for further analysis.

During execution of queries, following issues were identified and were resolved as mentioned below.

1. Missing fields were being used for performing aggregation. In order to resolve this issue, unwanted fields were removed prior to query execution.

2. **Cast** operation being performed was not compatible with the Oracle version being used. So, it was replaced with **to_date** inorder to resolve this error.

   (**cast**('1998-08-04' **as date**) + 14 days) was changed to

   (**to_date**('1998-08-04','YYYY-MM-DD') + 14)

3. The **EXCEPT** operator was not working, which was replaced with the **MINUS**.

During execution of queries, following observations were made and corresponding changes were done:

1. Query 14, 23, 24 ,39 were composed of two subqueries.We have considered both of them as a single query.

2. Original query templates fetched 100 rows. For the purpose of optimization, we have fetched all rows.

   Before execution of each query, shared_pool was flushed for precise tracking of execution time.

```
alter system flush shared_pool;
```

3.5 Load Test

Python script was prepared in order to load generated data into respective schema (say. DW_TPCDS_SF10). Before loading data into the tables, a tpcds.sql script was executed which contains DDL for creating tables. Once the data was loaded into the respective schema, script present in the tpcds_ri.sql file was executed containing DDL statement for adding referential integrity. While executing a query present in tpcds_ri.sql file, few errors were identified related to

1. Duplicate index name being used

2. Adding foreign key on field which was not present in table

For resolving these issues 1, we renamed the index name, while issue 2 was simply ignored.

The bar chart showing load time of data for scale factor of 1, 3, 5 and 10 is provided.

3.6 Power Test

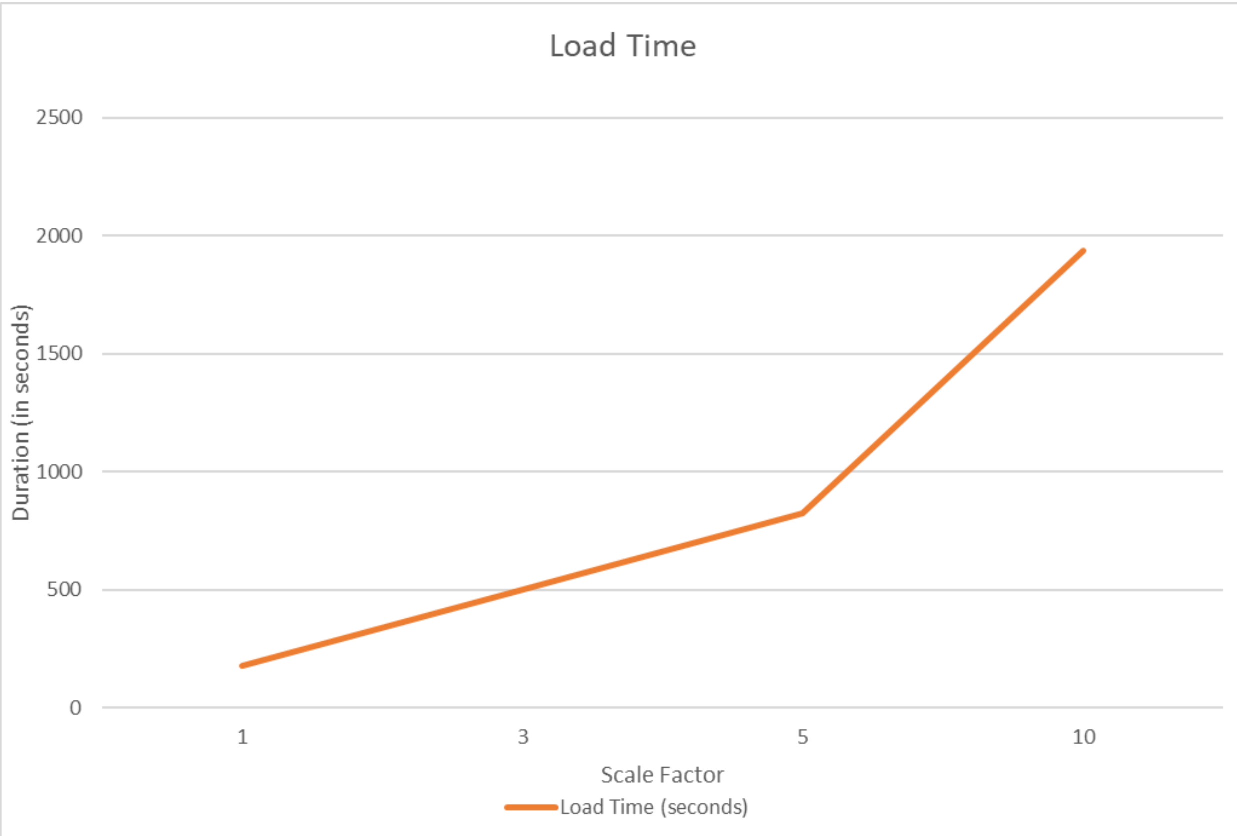
We prepared a python script in order to execute all 99 queries one after another in a single execution. Run time was dumped into the excel file for further analysis.

3.7 Throughput Test

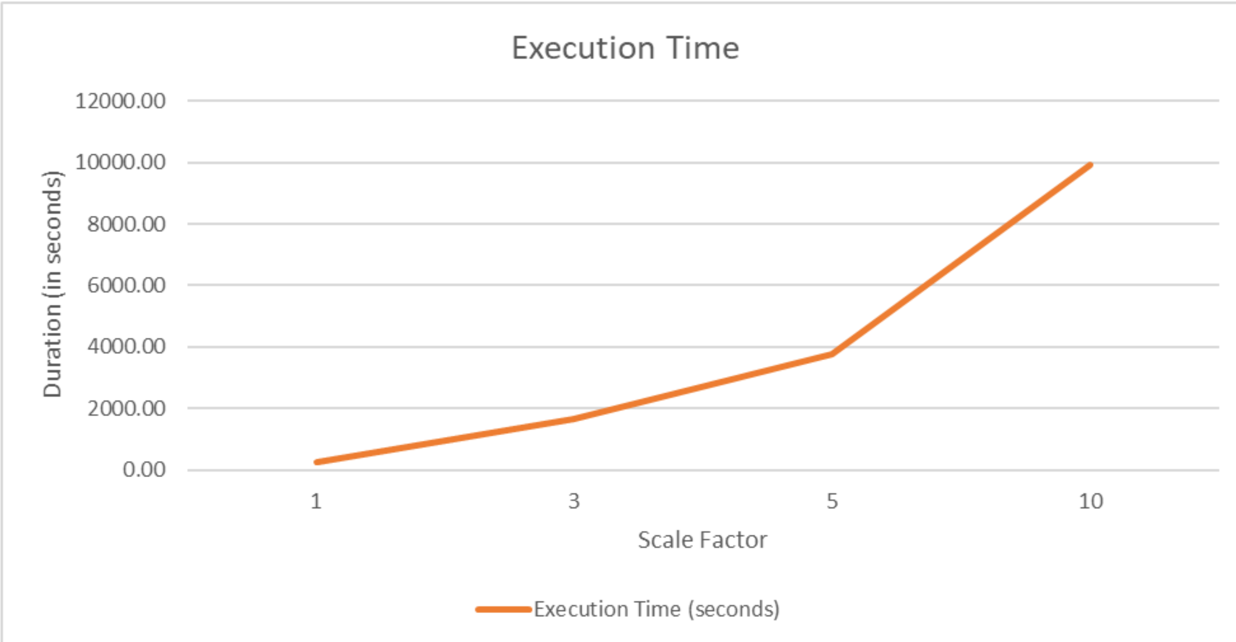
We prepared a python script for the execution of 99 queries in parallel using the multiprocessing module. Four processes were created, each executing 99 queries in random order. The start and end time for each process were tracked to calculate **Throughput Test Time**. This test could only be performed for the scale factor of 1, 3 and 5 since executing 99 queries in parallel by four processes for the scale factor of 10 was consuming a lot of resources causing the execution process to stop abruptly.

Chapter 4 - Results and Discussion

4.1 Load Test Result

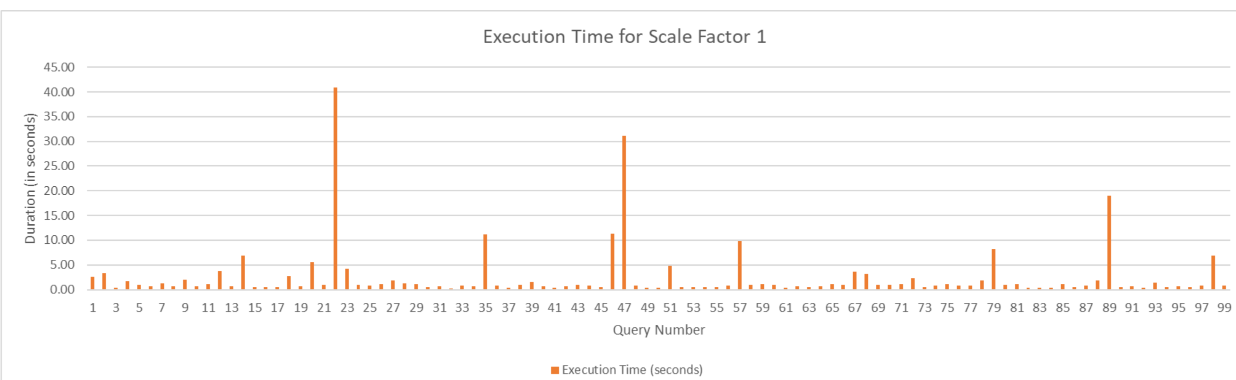


4.2 Power Test Result



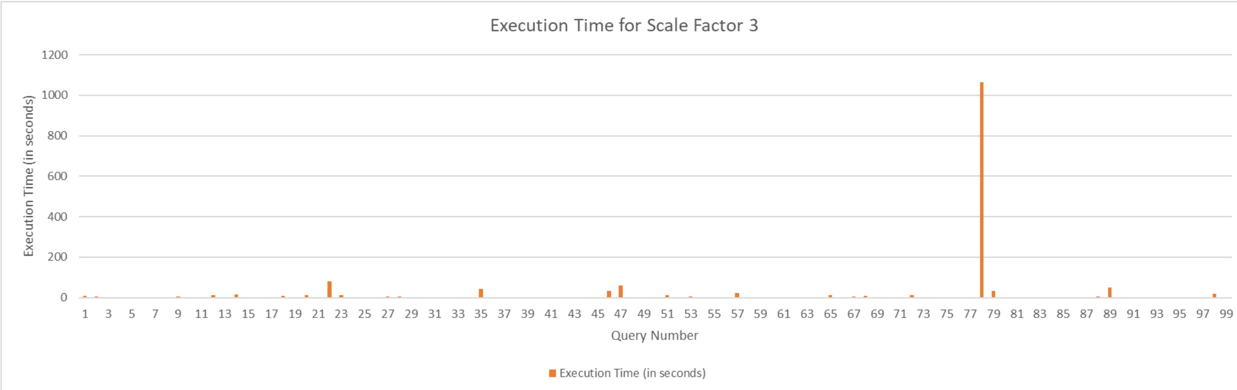
Execution time for 99 queries on scale factor of 1, 5 and 10 was analyzed. The bar graph showing execution time of each queries is provided. We then selected time consuming queries and tried to optimize them.

4.2.1 Scale Factor 1



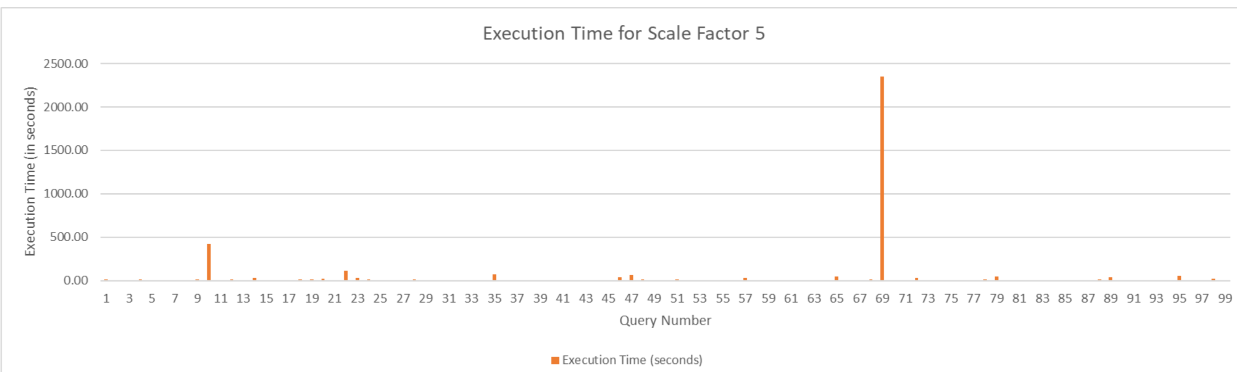
It took 248.24 sec (~4 min) for execution of 99 queries for the scale factor of 1, with minimum execution time of 0.3 sec taken by Query 32 and maximum execution time of 40.83 sec taken by Query 22.

4.2.2 Scale Factor 3



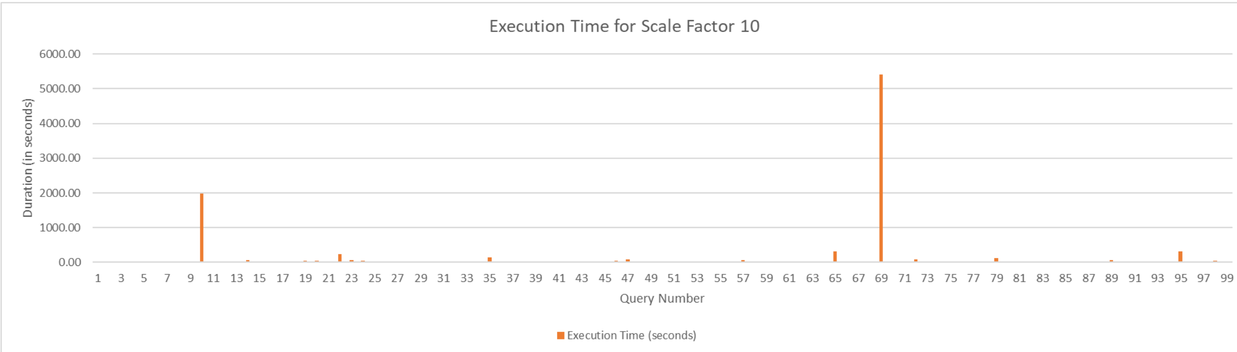
It took 1657.6 sec (~27 min) for execution of 99 queries for the scale factor of 3, with minimum execution time of 0.32 sec taken by Query 92 and maximum execution time of 1063.18 sec (~17 min) taken by Query 78.

4.2.3 Scale Factor 5



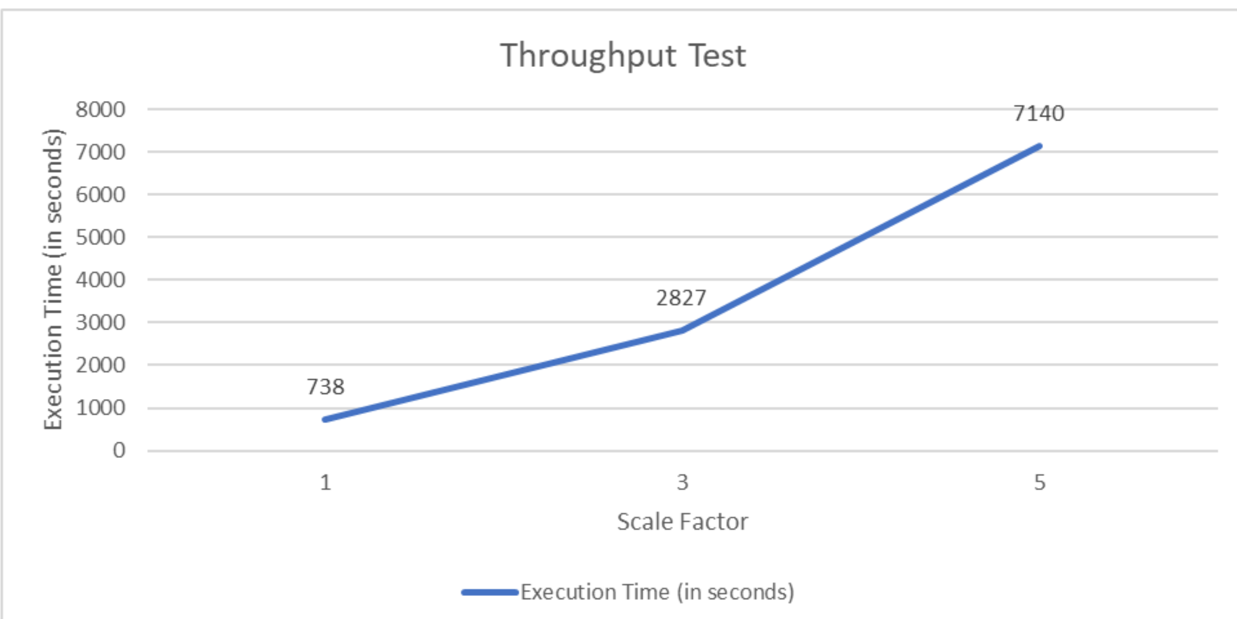
It took 3784.98 sec (~63 min) for execution of 99 queries for the scale factor of 5, with minimum execution time of 0.34 sec taken by Query 32 and maximum execution time of 2350.01 sec (~39 min) taken by Query 69.

4.2.4 Scale Factor 10

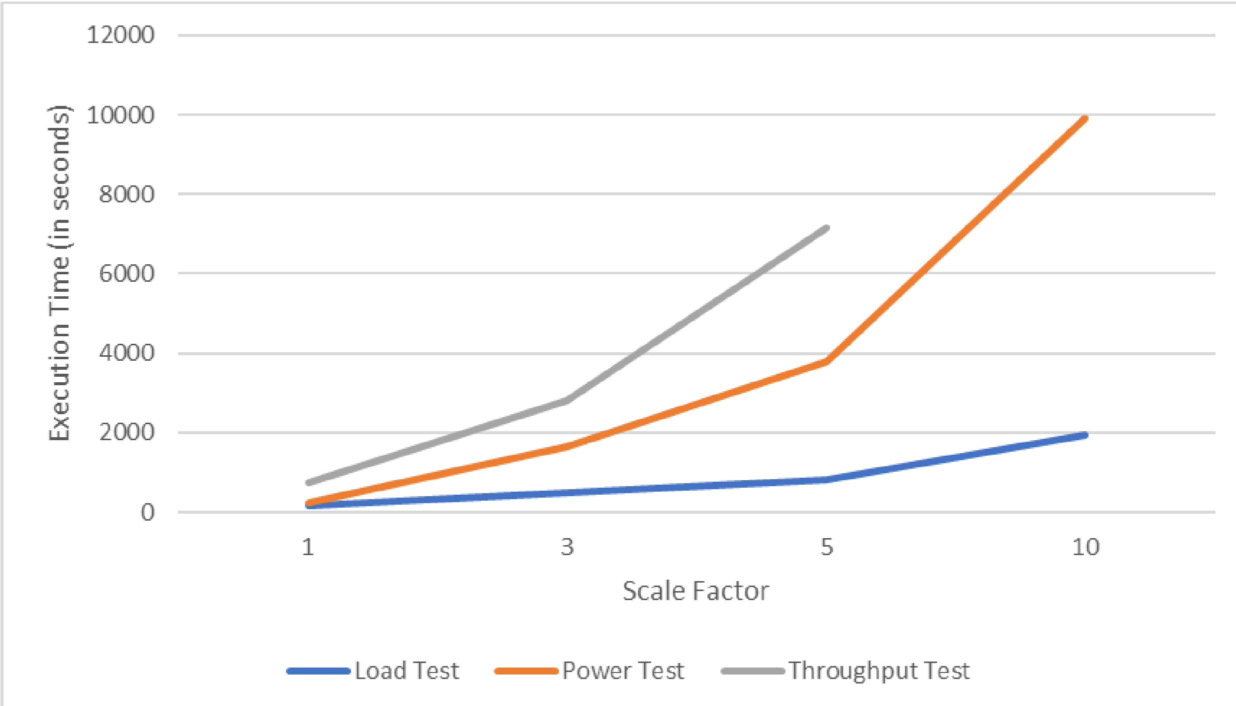


It took 9926.33 sec (~1 hr 45 min) for execution of 99 queries for the scale factor of 1, with minimum execution time of 0.41 sec taken by Query 61 and maximum execution time of 5400.17 sec (~1 hr 30 min) taken by Query 69.

4.3 Throughput Test Result



It took ~12 mins for each one of the 4 processes to execute all 99 queries in random sequence for the scale factor of 1, while the execution time for scale factor of 3 and 5 being ~47 mins and ~1 hour 49 mins respectively.



4.4 Optimizing Queries

4.4.1 Query 88

4.4.1.1 Original Query

```
SELECT
    *
FROM (
    SELECT
        COUNT(*) h8_30_to_9
    FROM store_sales, household_demographics , time_dim, store
    WHERE ss_sold_time_sk = time_dim.t_time_sk
        AND ss_hdemo_sk = household_demographics.hd_demo_sk
        AND ss_store_sk = s_store_sk
        AND time_dim.t_hour = 8
        AND time_dim.t_minute >= 30
        AND ((household_demographics.hd_dep_count = 3 AND household_demographics.hd_vehicle_count<=3+2)
OR
            (household_demographics.hd_dep_count = 0 AND household_demographics.hd_vehicle_count<=0+2) OR
            (household_demographics.hd_dep_count = 1 AND household_demographics.hd_vehicle_count<=1+2))
        AND store.s_store_name = 'ese'
) s1,
```

```
(
    SELECT
        COUNT(*) h9_to_9_30
    FROM store_sales, household_demographics , time_dim, store
    WHERE ss_sold_time_sk = time_dim.t_time_sk
        AND ss_hdemo_sk = household_demographics.hd_demo_sk
        AND ss_store_sk = s_store_sk
        AND time_dim.t_hour = 9
        AND time_dim.t_minute < 30
        AND ((household_demographics.hd_dep_count = 3 AND household_demographics.hd_vehicle_count<=3+2)
OR
            (household_demographics.hd_dep_count = 0 AND household_demographics.hd_vehicle_count<=0+2) OR
            (household_demographics.hd_dep_count = 1 AND household_demographics.hd_vehicle_count<=1+2))
        AND store.s_store_name = 'ese'
) s2,
(
    SELECT
        COUNT(*) h9_30_to_10
    FROM store_sales, household_demographics , time_dim, store
    WHERE ss_sold_time_sk = time_dim.t_time_sk
        AND ss_hdemo_sk = household_demographics.hd_demo_sk
        AND ss_store_sk = s_store_sk
        AND time_dim.t_hour = 9
        AND time_dim.t_minute >= 30
        AND ((household_demographics.hd_dep_count = 3 AND household_demographics.hd_vehicle_count<=3+2)
OR
            (household_demographics.hd_dep_count = 0 AND household_demographics.hd_vehicle_count<=0+2) OR
            (household_demographics.hd_dep_count = 1 AND household_demographics.hd_vehicle_count<=1+2))
        AND store.s_store_name = 'ese'
) s3,
(
    SELECT
        COUNT(*) h10_to_10_30
    FROM store_sales, household_demographics , time_dim, store
    WHERE ss_sold_time_sk = time_dim.t_time_sk
        AND ss_hdemo_sk = household_demographics.hd_demo_sk
        AND ss_store_sk = s_store_sk
        AND time_dim.t_hour = 10
        AND time_dim.t_minute < 30
        AND ((household_demographics.hd_dep_count = 3 AND household_demographics.hd_vehicle_count<=3+2)
OR
            (household_demographics.hd_dep_count = 0 AND household_demographics.hd_vehicle_count<=0+2) OR
            (household_demographics.hd_dep_count = 1 AND household_demographics.hd_vehicle_count<=1+2))
        AND store.s_store_name = 'ese'
) s4,
(
    SELECT
        COUNT(*) h10_30_to_11
```

```
    FROM store_sales, household_demographics , time_dim, store
    WHERE ss_sold_time_sk = time_dim.t_time_sk
    AND ss_hdemo_sk = household_demographics.hd_demo_sk
    AND ss_store_sk = s_store_sk
    AND time_dim.t_hour = 10
    AND time_dim.t_minute >= 30
    AND ((household_demographics.hd_dep_count = 3 AND household_demographics.hd_vehicle_count<=3+2) OR
        (household_demographics.hd_dep_count = 0 AND household_demographics.hd_vehicle_count<=0+2) OR
        (household_demographics.hd_dep_count = 1 AND household_demographics.hd_vehicle_count<=1+2))
    AND store.s_store_name = 'ese'
) s5,
(
    SELECT
        COUNT(*) h10_30_to_11
    FROM store_sales, household_demographics , time_dim, store
    WHERE ss_sold_time_sk = time_dim.t_time_sk
    AND ss_hdemo_sk = household_demographics.hd_demo_sk
    AND ss_store_sk = s_store_sk
    AND time_dim.t_hour = 11
    AND time_dim.t_minute < 30
    AND ((household_demographics.hd_dep_count = 3 AND household_demographics.hd_vehicle_count<=3+2) OR
        (household_demographics.hd_dep_count = 0 AND household_demographics.hd_vehicle_count<=0+2) OR
        (household_demographics.hd_dep_count = 1 AND household_demographics.hd_vehicle_count<=1+2))
    AND store.s_store_name = 'ese'
) s6,
(
    SELECT COUNT(*) h11_30_to_12
    FROM store_sales, household_demographics , time_dim, store
    WHERE ss_sold_time_sk = time_dim.t_time_sk
    AND ss_hdemo_sk = household_demographics.hd_demo_sk
    AND ss_store_sk = s_store_sk
    AND time_dim.t_hour = 11
    AND time_dim.t_minute >= 30
    AND ((household_demographics.hd_dep_count = 3 AND household_demographics.hd_vehicle_count<=3+2) OR
        (household_demographics.hd_dep_count = 0 AND household_demographics.hd_vehicle_count<=0+2) OR
        (household_demographics.hd_dep_count = 1 AND household_demographics.hd_vehicle_count<=1+2))
    AND store.s_store_name = 'ese'
) s7,
(
    SELECT COUNT(*) h12_to_12_30
    FROM store_sales, household_demographics , time_dim, store
    WHERE ss_sold_time_sk = time_dim.t_time_sk
    AND ss_hdemo_sk = household_demographics.hd_demo_sk
    AND ss_store_sk = s_store_sk
    AND time_dim.t_hour = 12
    AND time_dim.t_minute < 30
    AND ((household_demographics.hd_dep_count = 3 AND household_demographics.hd_vehicle_count<=3+2) OR
        (household_demographics.hd_dep_count = 0 AND household_demographics.hd_vehicle_count<=0+2) OR
```

```
        (household_demographics.hd_dep_count = 1 AND household_demographics.hd_vehicle_count<=1+2))
    AND store.s_store_name = 'ese'
) s8;
```

4.4.1.2 Changes made

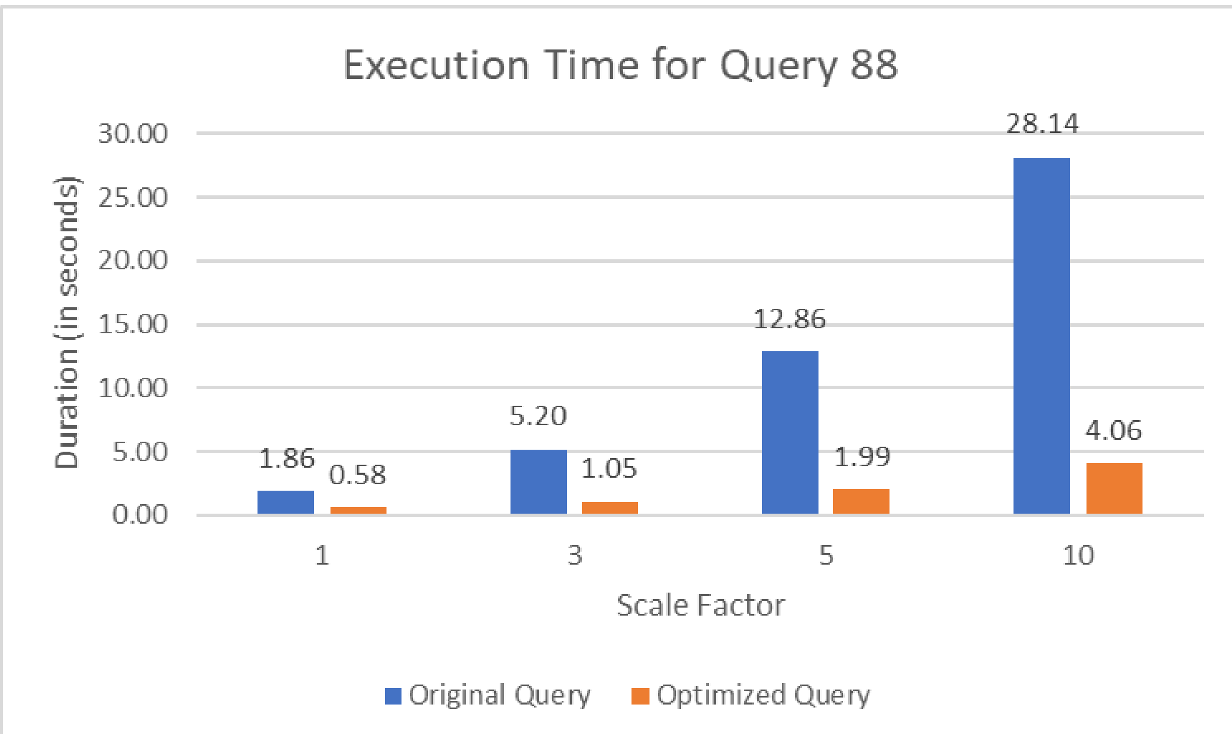
1. Same tables are joined multiple times (8 times) with different filtering criteria (hour and minute). Instead, all the required calculations have been done using a single join which has later been pivoted to display the result in the original format.
2. Dynamic run time computation has been replaced with static computation.

4.4.1.3 Optimized Query

```
SELECT
  *
FROM (
 SELECT
   flag,
   count(*) AS value
 FROM (
   SELECT
   CASE
     WHEN time_dim.t_hour = 8 AND time_dim.t_minute >= 30 THEN 1
     WHEN time_dim.t_hour = 9 AND time_dim.t_minute < 30 THEN 2
     WHEN time_dim.t_hour = 9 AND time_dim.t_minute >= 30 THEN 3
     WHEN time_dim.t_hour = 10 AND time_dim.t_minute < 30 THEN 4
     WHEN  time_dim.t_hour = 10 AND time_dim.t_minute >= 30 THEN 5
     WHEN time_dim.t_hour = 11 AND time_dim.t_minute < 30 THEN 6
     WHEN time_dim.t_hour = 11 AND time_dim.t_minute >= 30 THEN 7
     WHEN time_dim.t_hour = 12 AND time_dim.t_minute < 30 THEN 8
   END AS flag
   FROM store_sales, household_demographics , time_dim, store
   WHERE
       ss_sold_time_sk = time_dim.t_time_sk
       AND ss_hdemo_sk = household_demographics.hd_demo_sk
       AND ss_store_sk = s_store_sk
       AND (
           (household_demographics.hd_dep_count = 3 AND household_demographics.hd_vehicle_count<=5) OR
           (household_demographics.hd_dep_count = 0 AND household_demographics.hd_vehicle_count<=2) OR
           (household_demographics.hd_dep_count = 1 AND household_demographics.hd_vehicle_count<=3)
       )
       AND store.s_store_name = 'ese'
    )
```

```
    GROUP BY flag
)
PIVOT (
  SUM (value) AS sum_value FOR (flag) IN (
    1 AS Bucket1,
    2 AS Bucket2,
    3 AS Bucket3,
    4 AS Bucket4,
    5 AS Bucket5,
    6 AS Bucket6,
    7 AS Bucket7,
    8 AS Bucket8
  )
);
```

4.4.1.4 Execution Time before and after optimization



After Query Optimization, the execution time for Query 88 decreased by 85.57% for the scale factor of 10.

4.4.2 Query 9

4.4.2.1 Original Query

```
SELECT
CASE WHEN (SELECT
            COUNT(*)
            FROM store_sales
            WHERE ss_quantity BETWEEN 1 AND 20) > 25437
    THEN (SELECT AVG(ss_ext_discount_amt)
            FROM store_sales
            WHERE ss_quantity BETWEEN 1 AND 20)
        ELSE (SELECT AVG(ss_net_profit)
            FROM store_sales
            WHERE ss_quantity BETWEEN 1 AND 20) END bucket1 ,
    CASE WHEN (SELECT COUNT(*)
            FROM store_sales
            WHERE ss_quantity BETWEEN 21 AND 40) > 22746
        THEN (SELECT AVG(ss_ext_discount_amt)
            FROM store_sales
            WHERE ss_quantity BETWEEN 21 AND 40)
        ELSE (SELECT AVG(ss_net_profit)
            FROM store_sales
            WHERE ss_quantity BETWEEN 21 AND 40) END bucket2,
    CASE WHEN (SELECT COUNT(*)
            FROM store_sales
            WHERE ss_quantity BETWEEN 41 AND 60) > 9387
        THEN (SELECT AVG(ss_ext_discount_amt)
            FROM store_sales
            WHERE ss_quantity BETWEEN 41 AND 60)
        ELSE (SELECT AVG(ss_net_profit)
            FROM store_sales
            WHERE ss_quantity BETWEEN 41 AND 60) END bucket3,
    CASE WHEN (SELECT COUNT(*)
            FROM store_sales
            WHERE ss_quantity BETWEEN 61 AND 80) > 10098
        THEN (SELECT AVG(ss_ext_discount_amt)
            FROM store_sales
            WHERE ss_quantity BETWEEN 61 AND 80)
        ELSE (SELECT AVG(ss_net_profit)
            FROM store_sales
            WHERE ss_quantity BETWEEN 61 AND 80) END bucket4,
    CASE WHEN (SELECT COUNT(*)
            FROM store_sales
            WHERE ss_quantity BETWEEN 81 AND 100) > 18213
        THEN (SELECT AVG(ss_ext_discount_amt)
            FROM store_sales
```

```
            WHERE ss_quantity BETWEEN 81 AND 100)
        ELSE (SELECT AVG(ss_net_profit)
            FROM store_sales
            WHERE ss_quantity BETWEEN 81 AND 100) END bucket5
FROM reason
WHERE r_reason_sk = 1;
```

4.4.2.2 Changes made

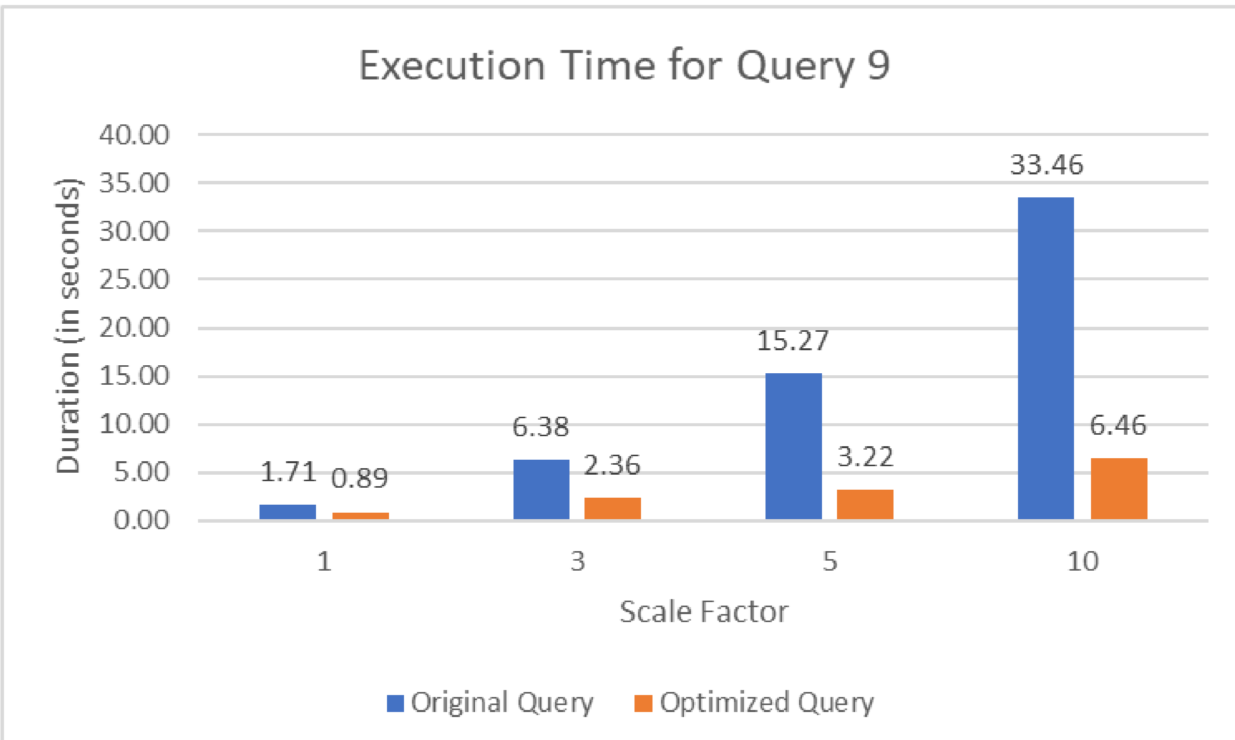
1. Similar to query 88, Same table is being scanned multiple times (5 times) with different filtering criteria (quantity). Instead, all the required calculations have been done using a single table scan which has later been pivoted to display the result in the original format.

4.4.2.3 Optimized Query

```
SELECT
   *
FROM (
   SELECT
      bucket,
   CASE
      WHEN bucket=1 AND cnt>25437 THEN ss_ext_discount_amt
      WHEN bucket=2 AND cnt>22746 THEN ss_ext_discount_amt
      WHEN bucket=3 AND cnt>9387 THEN ss_ext_discount_amt
      WHEN bucket=4 AND cnt>10098 THEN ss_ext_discount_amt
      WHEN bucket=5 AND cnt>18213 THEN ss_ext_discount_amt
      ELSE ss_net_profit END AS value
   FROM (
      SELECT
         bucket,
         count(*) cnt,
         avg(ss_ext_discount_amt) ss_ext_discount_amt,
         avg(ss_net_profit) ss_net_profit FROM (
            SELECT
               ss_ext_discount_amt,
               ss_net_profit,
               CASE
                  WHEN  ss_quantity BETWEEN 1 AND 20 THEN 1
                  WHEN ss_quantity BETWEEN 21 AND 40 THEN 2
                  WHEN  ss_quantity BETWEEN 41 AND 60 THEN 3
                  WHEN ss_quantity BETWEEN 61 AND 80 THEN 4
                  WHEN ss_quantity BETWEEN 81 AND 100 THEN 5
               END AS bucket
```

```
            FROM  store_sales
         ) GROUP BY bucket
      )
)
PIVOT (SUM (value) AS sum_value FOR(bucket) IN (
   1 AS Bucket1,
   2 AS Bucket2,
   3 AS Bucket3,
   4 AS Bucket4,
   5 AS Bucket5)
);
```

4.4.2.4 Execution Time before and after optimization



After Query Optimization, the execution time for Query 9 decreased by 80.69% for the scale factor of 10.

4.4.3 Query 28

4.4.3.1 Original Query

```
SELECT * FROM (SELECT  *
FROM (SELECT AVG(ss_list_price) B1_LP
          ,COUNT(ss_list_price) B1_CNT
          ,COUNT(distinct ss_list_price) B1_CNTD
     FROM store_sales
     WHERE ss_quantity BETWEEN 0 AND 5
       AND (ss_list_price BETWEEN 11 AND 11+10
           OR ss_coupon_amt BETWEEN 460 AND 460+1000
           OR ss_wholesale_cost BETWEEN 14 AND 14+20)) B1,
    (SELECT AVG(ss_list_price) B2_LP
          ,COUNT(ss_list_price) B2_CNT
          ,COUNT(distinct ss_list_price) B2_CNTD
     FROM store_sales
     WHERE ss_quantity BETWEEN 6 AND 10
       AND (ss_list_price BETWEEN 91 AND 91+10
         OR ss_coupon_amt BETWEEN 1430 AND 1430+1000
         OR ss_wholesale_cost BETWEEN 32 AND 32+20)) B2,
    (SELECT AVG(ss_list_price) B3_LP
          ,COUNT(ss_list_price) B3_CNT
          ,COUNT(distinct ss_list_price) B3_CNTD
     FROM store_sales
     WHERE ss_quantity BETWEEN 11 AND 15
       AND (ss_list_price BETWEEN 66 AND 66+10
         OR ss_coupon_amt BETWEEN 920 AND 920+1000
         OR ss_wholesale_cost BETWEEN 4 AND 4+20)) B3,
    (SELECT AVG(ss_list_price) B4_LP
          ,COUNT(ss_list_price) B4_CNT
          ,COUNT(distinct ss_list_price) B4_CNTD
     FROM store_sales
     WHERE ss_quantity BETWEEN 16 AND 20
       AND (ss_list_price BETWEEN 142 AND 142+10
         OR ss_coupon_amt BETWEEN 3054 AND 3054+1000
         OR ss_wholesale_cost BETWEEN 80 AND 80+20)) B4,
    (SELECT AVG(ss_list_price) B5_LP
          ,COUNT(ss_list_price) B5_CNT
          ,COUNT(distinct ss_list_price) B5_CNTD
     FROM store_sales
     WHERE ss_quantity BETWEEN 21 AND 25
       AND (ss_list_price BETWEEN 135 AND 135+10
         OR ss_coupon_amt BETWEEN 14180 AND 14180+1000
         OR ss_wholesale_cost BETWEEN 38 AND 38+20)) B5,
    (SELECT AVG(ss_list_price) B6_LP
          ,COUNT(ss_list_price) B6_CNT
```

```
      ,COUNT(distinct ss_list_price) B6_CNTD
    FROM store_sales
    WHERE ss_quantity BETWEEN 26 AND 30
     AND (ss_list_price BETWEEN 28 AND 28+10
      OR ss_coupon_amt BETWEEN 2513 AND 2513+1000
      OR ss_wholesale_cost BETWEEN 42 AND 42+20)) B6
 ) ;
```

4.4.3.2 Changes made

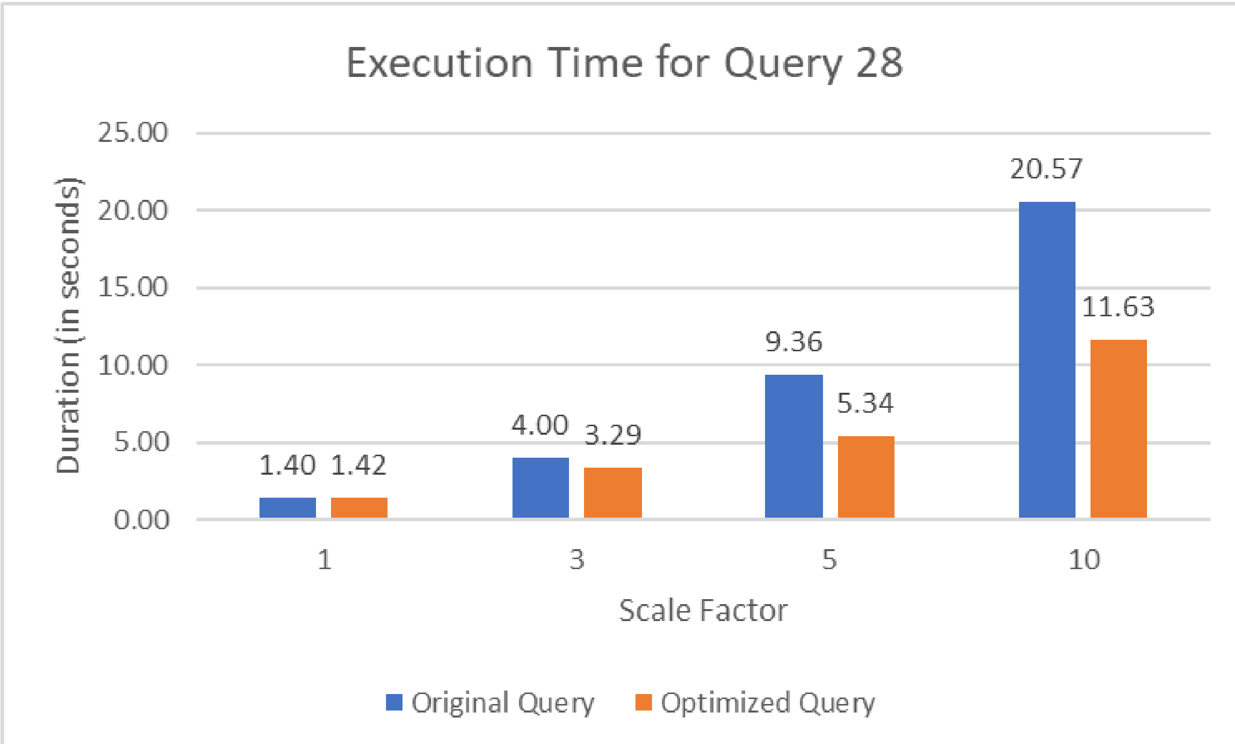
1. Similar to query 88 and 9, Same table is being scanned multiple times (6 times) with different filtering criteria (ss_list_price, ss_coupon_amt, ss_wholesale_cost). Instead, all the required calculations have been done using a single table scan which has later been pivoted to display the result in the original format.
2. Dynamic run computation has been replaced with static computation.

4.4.3.3 Optimized Query

```
SELECT
  SUM(CASE WHEN flag=1 THEN LP ELSE 0 END) AS B1_LP,
  SUM(CASE WHEN flag=1 THEN CNT ELSE 0 END) AS B1_CNT,
  SUM(CASE WHEN flag=1 THEN CNTD ELSE 0 END) AS B1_CNTD,
  SUM(CASE WHEN flag=2 THEN LP ELSE 0 END) AS B2_LP,
  SUM(CASE WHEN flag=2 THEN CNT ELSE 0 END) AS B2_CNT,
  SUM(CASE WHEN flag=2 THEN CNTD ELSE 0 END) AS B2_CNTD,
  SUM(CASE WHEN flag=3 THEN LP ELSE 0 END) AS B3_LP,
  SUM(CASE WHEN flag=3 THEN CNT ELSE 0 END) AS B3_CNT,
  SUM(CASE WHEN flag=3 THEN CNTD ELSE 0 END) AS B3_CNTD,
  SUM(CASE WHEN flag=4 THEN LP ELSE 0 END) AS B4_LP,
  SUM(CASE WHEN flag=4 THEN CNT ELSE 0 END) AS B4_CNT,
  SUM(CASE WHEN flag=4 THEN CNTD ELSE 0 END) AS B4_CNTD,
  SUM(CASE WHEN flag=5 THEN LP ELSE 0 END) AS B5_LP,
  SUM(CASE WHEN flag=5 THEN CNT ELSE 0 END) AS B5_CNT,
  SUM(CASE WHEN flag=5 THEN CNTD ELSE 0 END) AS B5_CNTD,
  SUM(CASE WHEN flag=6 THEN LP ELSE 0 END) AS B6_LP,
  SUM(CASE WHEN flag=6 THEN CNT ELSE 0 END) AS B6_CNT,
  SUM(CASE WHEN flag=6 THEN CNTD ELSE 0 END) AS B6_CNTD
FROM (
  SELECT
    flag,
    AVG(ss_list_price) LP,
    COUNT(ss_list_price) CNT,
```

```
    COUNT(DISTINCT ss_list_price) CNTD
  FROM (
    SELECT
      ss_list_price,
      CASE
        WHEN ss_quantity BETWEEN 0 AND 5 AND (ss_list_price BETWEEN 11 AND 21 OR ss_coupon_amt
BETWEEN 460 AND 1460 OR ss_wholesale_cost BETWEEN 14 AND 34) THEN 1
        WHEN ss_quantity BETWEEN 6 AND 10 AND (ss_list_price BETWEEN 91 AND 101 OR
ss_coupon_amt BETWEEN 1430 AND 2430 OR ss_wholesale_cost BETWEEN 32 AND 52) THEN 2
        WHEN ss_quantity BETWEEN 11 AND 15 AND (ss_list_price BETWEEN 66 AND 76 OR ss_coupon_amt
BETWEEN 920 AND 1920 OR ss_wholesale_cost BETWEEN 4 AND 24) THEN 3
        WHEN ss_quantity BETWEEN 16 AND 20 AND (ss_list_price BETWEEN 142 AND 152 OR
ss_coupon_amt BETWEEN 3054 AND 4054 OR ss_wholesale_cost BETWEEN 80 AND 100) THEN 4
        WHEN ss_quantity BETWEEN 21 AND 25 AND (ss_list_price BETWEEN 135 AND 145 OR
ss_coupon_amt BETWEEN 14180 AND 15180 OR ss_wholesale_cost BETWEEN 38 AND 58) THEN 5
        when ss_quantity BETWEEN 26 AND 30 AND (ss_list_price BETWEEN 28 AND 38 OR ss_coupon_amt
BETWEEN 2513 AND 3513 OR ss_wholesale_cost BETWEEN 42 AND 62) THEN 6
      END AS flag
    from store_sales
  )
  GROUP BY flag
);
```

4.4.3.4 Execution Time before and after optimization



After Query Optimization, the execution time for Query 28 decreased by 43.46% for the scale factor of 10 even though there is slight increase in execution time for the scale factor of 1.

4.4.4 Query 65

4.4.4.1 Original Query

```
SELECT * FROM (
  SELECT
    s_store_name,
    i_item_desc,
    sc.revenue,
    i_current_price,
    i_wholesale_cost,
    i_brand
  FROM store, item, (
    SELECT ss_store_sk, avg(revenue) AS ave
    FROM (
      SELECT
        ss_store_sk,
```

```
            ss_item_sk,
            SUM(ss_sales_price) AS revenue
          FROM store_sales, date_dim
          WHERE ss_sold_date_sk = d_date_sk AND d_month_seq between 1212 AND 1212+11
          GROUP BY ss_store_sk, ss_item_sk
        ) sa
        GROUP BY ss_store_sk
    ) sb, (
    SELECT
        ss_store_sk,
        ss_item_sk,
        SUM(ss_sales_price) AS revenue
    FROM store_sales, date_dim
    WHERE ss_sold_date_sk = d_date_sk AND d_month_seq between 1212 AND 1212+11
    GROUP BY ss_store_sk, ss_item_sk
    ) sc
    WHERE sb.ss_store_sk = sc.ss_store_sk AND
        sc.revenue <= 0.1 * sb.ave AND
        s_store_sk = sc.ss_store_sk AND
        i_item_sk = sc.ss_item_sk
    ORDER BY s_store_name, i_item_desc
  );
```

4.4.4.2 Changes made

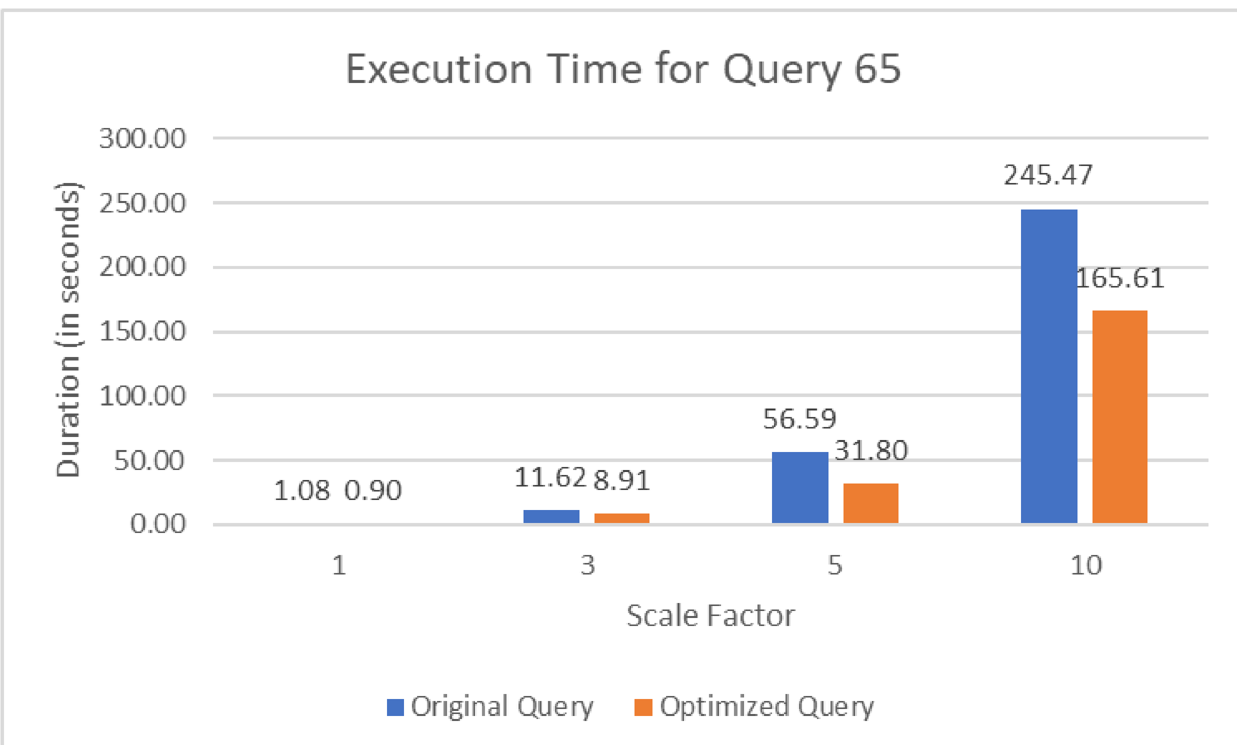
1. Two tables are being joined twice, once for the calculation of sum(ss_sales_price) grouped by ss_store_sk and ss_item_sk and second for the calculation of avg(ss_sales_price) grouped by ss_store_sk. Instead, a single join with partition window function has been used.
2. Dynamic run computation has been replaced with static computation.

4.4.4.3 Optimized Query

```
SELECT
  s_store_name,
  i_item_desc,
  revenue
FROM store, item, (
  SELECT
    ss_store_sk,
    ss_item_sk,
    revenue,
    AVG(revenue) OVER (PARTITION BY ss_store_sk) avgR
```

```
  FROM (
    SELECT
      ss_store_sk,
      ss_item_sk,
      sum(ss_sales_price) AS revenue
    FROM store_sales, date_dim
    WHERE ss_sold_date_sk = d_date_sk
    AND d_month_seq BETWEEN 1212 AND 1223
    GROUP BY ss_store_sk, ss_item_sk
  ) X
) Y
WHERE
  revenue <= 0.1 * avgR
  AND ss_store_sk = s_store_sk
  AND ss_item_sk = i_item_sk
ORDER BY s_store_name, i_item_desc;
```

4.4.4.4 Execution Time before and after optimization



After Query Optimization, the execution time for Query 65 decreased by 32.53% for the scale factor of 10.

4.4.5 Query 69

4.4.5.1 Original Query

```
SELECT * FROM (SELECT
  cd_gender,
  cd_marital_status,
  cd_education_status,
  COUNT(*) cnt1,
  cd_purchase_estimate,
  COUNT(*) cnt2,
  cd_credit_rating,
  COUNT(*) cnt3
 FROM
  customer c,customer_address ca,customer_demographics
 WHERE
  c.c_current_addr_sk = ca.ca_address_sk AND
  ca_state in ('CO','IL','MN') AND
  cd_demo_sk = c.c_current_cdemo_sk AND
  EXISTS (SELECT *
          FROM store_sales,date_dim
          WHERE c.c_customer_sk = ss_customer_sk AND
                ss_sold_date_sk = d_date_sk AND
                d_year = 1999 AND
                d_moy BETWEEN 1 AND 1+2) AND
   (NOT EXISTS (SELECT *
            FROM web_sales,date_dim
            WHERE c.c_customer_sk = ws_bill_customer_sk AND
                  ws_sold_date_sk = d_date_sk AND
                  d_year = 1999 AND
                  d_moy BETWEEN 1 AND 1+2) AND
    NOT EXISTS (SELECT *
            FROM catalog_sales,date_dim
            WHERE c.c_customer_sk = cs_ship_customer_sk AND
                  cs_sold_date_sk = d_date_sk AND
                  d_year = 1999 AND
                  d_moy BETWEEN 1 AND 1+2))
 GROUP BY cd_gender,
          cd_marital_status,
          cd_education_status,
          cd_purchase_estimate,
          cd_credit_rating
 ORDER BY cd_gender,
          cd_marital_status,
          cd_education_status,
          cd_purchase_estimate,
          cd_credit_rating
```

```
);
```

4.4.5.2 Changes made

1. Here, multiple correlated subqueries are being used in order to find the customers who have performed transactions in store but not in web and catalog for the given time frame. Instead, **MINUS** operator has been used to optimize the query.

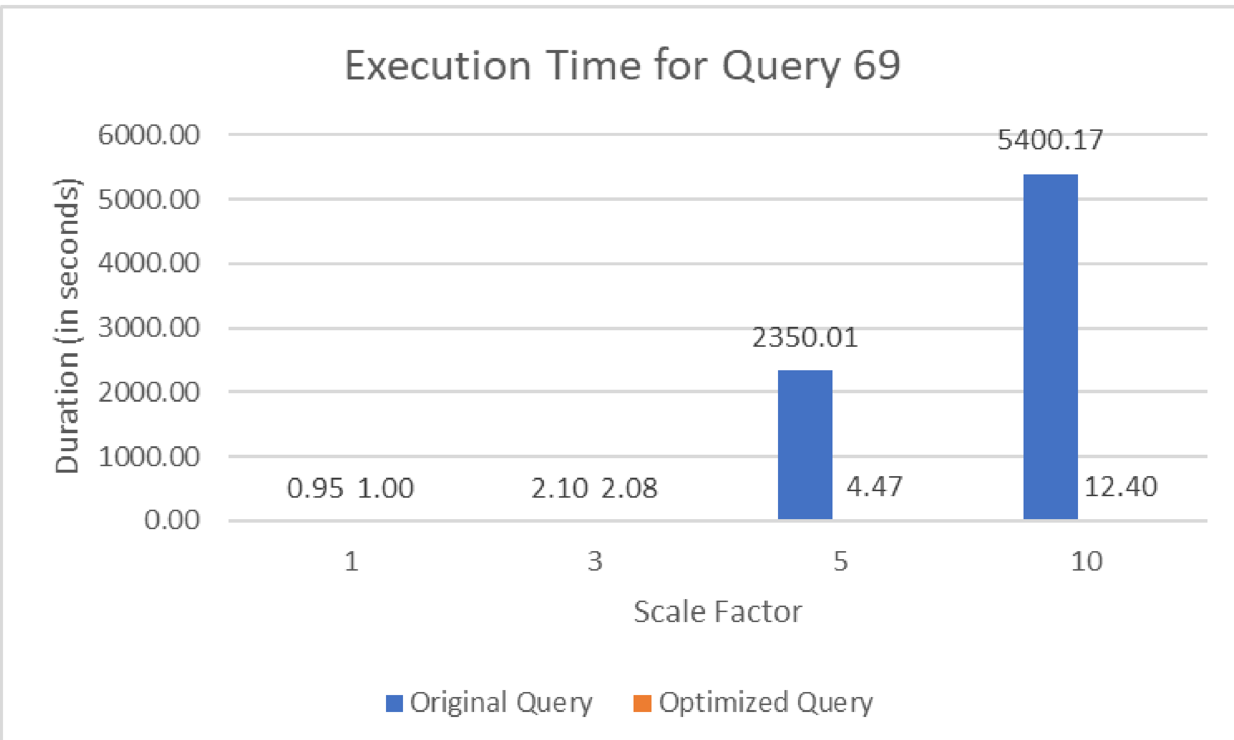
Dynamic run computation has been replaced with static computation.

4.4.5.3 Optimized Query

```
SELECT
  cd_gender,
  cd_marital_status,
  cd_education_status,
  COUNT(*) cnt1,
  cd_purchase_estimate,
  COUNT(*) cnt2,
  cd_credit_rating,
  COUNT(*) cnt3
FROM customer c,customer_address ca,customer_demographics,(
    (
        SELECT
            ss_customer_sk customer_sk
        FROM store_sales,date_dim
        WHERE ss_sold_date_sk = d_date_sk AND
           d_year = 1999 AND
           d_moy BETWEEN 1 AND 3
    )
    MINUS
    (
        SELECT ws_bill_customer_sk customer_sk
        from web_sales,date_dim
        WHERE ws_sold_date_sk = d_date_sk AND
            d_year = 1999 AND
            d_moy BETWEEN 1 AND 3
    )
    MINUS
    (
        SELECT cs_ship_customer_sk customer_sk
        from catalog_sales,date_dim
        WHERE cs_sold_date_sk = d_date_sk AND
```

```
            d_year = 1999 AND
            d_moy BETWEEN 1 AND 3
    )
)
WHERE
    c.c_current_addr_sk = ca.ca_address_sk AND
    ca_state IN ('CO','IL','MN') AND
    cd_demo_sk = c.c_current_cdemo_sk AND
    c.c_customer_sk = customer_sk
GROUP BY cd_gender, cd_marital_status, cd_education_status, cd_purchase_estimate, cd_credit_rating
ORDER BY cd_gender, cd_marital_status, cd_education_status, cd_purchase_estimate, cd_credit_rating;
```

4.4.5.4 Execution Time before and after optimization



After Query Optimization, the execution time for Query 69 decreased by 99.77% for the scale factor of 10 even though there is slight increase in execution time for the scale factor of 1.

4.4.6 Query 10

4.4.6.1 Original Query

```
SELECT * FROM (SELECT
  cd_gender,
  cd_marital_status,
  cd_education_status,
  COUNT(*) cnt1,
  cd_purchase_estimate,
  COUNT(*) cnt2,
  cd_credit_rating,
  COUNT(*) cnt3,
  cd_dep_count,
  COUNT(*) cnt4,
  cd_dep_employed_count,
  COUNT(*) cnt5,
  cd_dep_college_count,
  COUNT(*) cnt6
 FROM
  customer c,customer_address ca,customer_demographics
 WHERE
  c.c_current_addr_sk = ca.ca_address_sk AND
  ca_county in ('Walker County','RichlAND County','Gaines County','Douglas County','Dona Ana County') AND
  cd_demo_sk = c.c_current_cdemo_sk AND
  EXISTS (SELECT *
          FROM store_sales,date_dim
          WHERE c.c_customer_sk = ss_customer_sk AND
                ss_sold_date_sk = d_date_sk AND
                d_year = 2002 AND
                d_moy between 4 AND 4+3) AND
   (EXISTS (SELECT *
            FROM web_sales,date_dim
            WHERE c.c_customer_sk = ws_bill_customer_sk AND
                  ws_sold_date_sk = d_date_sk AND
                  d_year = 2002 AND
                  d_moy between 4 AND 4+3) or
    EXISTS (SELECT *
            FROM catalog_sales,date_dim
            WHERE c.c_customer_sk = cs_ship_customer_sk AND
                  cs_sold_date_sk = d_date_sk AND
                  d_year = 2002 AND
                  d_moy between 4 AND 4+3))
 GROUP BY cd_gender,
          cd_marital_status,
          cd_education_status,
          cd_purchase_estimate,
```

```
        cd_credit_rating,
        cd_dep_count,
        cd_dep_employed_count,
        cd_dep_college_count
 ORDER BY cd_gender,
        cd_marital_status,
        cd_education_status,
        cd_purchase_estimate,
        cd_credit_rating,
        cd_dep_count,
        cd_dep_employed_count,
        cd_dep_college_count
 ) ;
```

4.4.6.2 Changes made

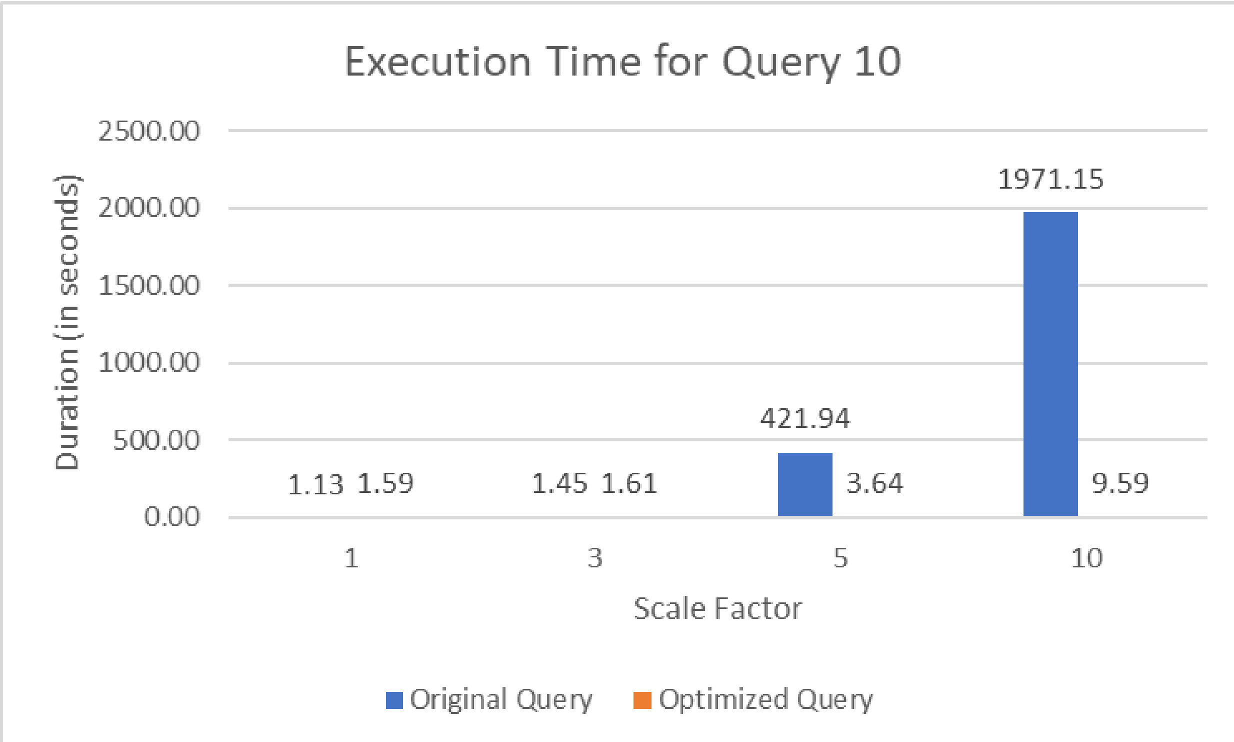
1. Here, multiple correlated subqueries are being used in order to find the customers who have performed transactions in store and in either web or catalog for the given time frame. Instead, **INTERSECT** and **UNION** operators have been used to optimize the query.
2. Dynamic run computation has been replaced with static computation.

4.4.6.3 Optimized Query

```
SELECT
  cd_gender,
  cd_marital_status,
  cd_education_status,
  count(*) cnt1,
  cd_purchase_estimate,
  count(*) cnt2,
  cd_credit_rating,
  count(*) cnt3,
  cd_dep_count,
  count(*) cnt4,
  cd_dep_employed_count,
  count(*) cnt5,
  cd_dep_college_count,
  count(*) cnt6
FROM customer c,customer_address ca,customer_demographics,(
     (
          SELECT
               ss_customer_sk customer_sk
```

```
        FROM store_sales,date_dim
        WHERE ss_sold_date_sk = d_date_sk AND
          d_year = 2002 AND
          d_moy BETWEEN 4 AND 7
    )
    INTERSECT ((
            SELECT ws_bill_customer_sk customer_sk
            FROM web_sales,date_dim
            WHERE ws_sold_date_sk = d_date_sk AND
                d_year = 2002 AND
                d_moy BETWEEN 4 AND 7
        )
    UNION (
        SELECT
            cs_ship_customer_sk customer_sk
        FROM catalog_sales,date_dim
        WHERE cs_sold_date_sk = d_date_sk AND
            d_year = 2002 AND
            d_moy BETWEEN 4 AND 7)
    )
)
WHERE c.c_current_addr_sk = ca.ca_address_sk AND
    ca_county in ('Walker County','RichlAND County','Gaines County','Douglas County','Dona Ana County')
AND
    cd_demo_sk = c.c_current_cdemo_sk AND
    c.c_customer_sk = customer_sk
GROUP BY cd_gender, cd_marital_status, cd_education_status, cd_purchase_estimate, cd_credit_rating,
cd_dep_count, cd_dep_employed_count, cd_dep_college_count
ORDER BY cd_gender, cd_marital_status, cd_education_status, cd_purchase_estimate, cd_credit_rating,
cd_dep_count, cd_dep_employed_count, cd_dep_college_count;
```

4.4.6.4 Execution time before and after optimization



After Query Optimization, the execution time for Query 10 decreased by 99.51% for the scale factor of 10 even though there is slight increase in execution time for the scale factor of 1 and 3.

4.5 Unsuccessful Attempts

Besides the above queries, we also tried optimizing a few more queries but changes we made did not have a significant impact on its execution time. We have added details of the way through which we tried optimizing these queries.

4.5.1 Query 46

4.5.1.1 Original Query

```
SELECT * FROM (
    SELECT
        c_last_name,
        c_first_name,
        ca_city,
        bought_city,
        ss_ticket_number,
```

```
        amt,
        profit
    FROM(
        SELECT
            ss_ticket_number,
            ss_customer_sk,
            ca_city bought_city,
            SUM(ss_coupon_amt) amt,
            SUM(ss_net_profit) profit
        FROM store_sales,date_dim,store,household_demographics,customer_address
        WHERE store_sales.ss_sold_date_sk = date_dim.d_date_sk
            AND store_sales.ss_store_sk = store.s_store_sk
            AND store_sales.ss_hdemo_sk = household_demographics.hd_demo_sk
            AND store_sales.ss_addr_sk = customer_address.ca_address_sk
            AND (household_demographics.hd_dep_count = 5 OR household_demographics.hd_vehicle_count=
3)
            AND date_dim.d_dow IN (6,0)
            AND date_dim.d_year IN (1999,1999+1,1999+2)
            AND store.s_city IN ('Midway','Fairview','Fairview','Midway','Fairview')
        GROUP BY ss_ticket_number,ss_customer_sk,ss_addr_sk,ca_city
    ) dn,customer,customer_address current_addr
    WHERE ss_customer_sk = c_customer_sk
        AND customer.c_current_addr_sk = current_addr.ca_address_sk
        AND current_addr.ca_city <> bought_city
    ORDER BY c_last_name, c_first_name, ca_city, bought_city, ss_ticket_number
) ;
```

4.5.1.2 Changes made

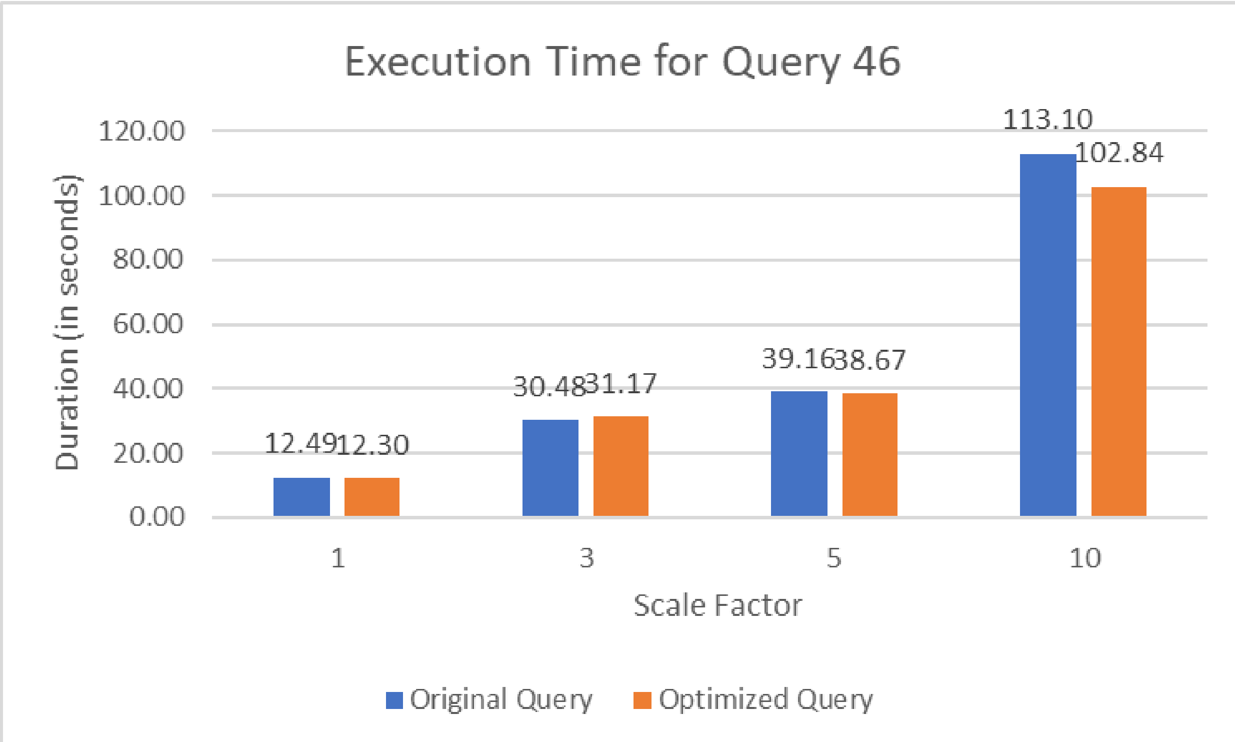
1. Duplicate values being used in **IN** condition were replaced by distinct values.
2. Dynamic run computation has been replaced with static computation.
3. Use of subquery was avoided by using join operation

4.5.1.3 Optimized Query

```
SELECT
  c_last_name,
  c_first_name,
  ca_city,
  bought_city,
  ss_ticket_number,
  amt,
  profit
```

```
FROM (
   SELECT
      ss_ticket_number,
      ss_customer_sk,
      customer_address.ca_city bought_city,
      current_addr.ca_city,c_last_name,
      c_first_name,
      sum(ss_coupon_amt) amt,
      sum(ss_net_profit) profit
   FROM store_sales,date_dim,store,household_demographics,customer_address, customer,customer_address
current_addr
   WHERE store_sales.ss_sold_date_sk = date_dim.d_date_sk
      AND store_sales.ss_store_sk = store.s_store_sk
      AND store_sales.ss_hdemo_sk = household_demographics.hd_demo_sk
      AND store_sales.ss_addr_sk = customer_address.ca_address_sk
      AND (household_demographics.hd_dep_count = 5 OR household_demographics.hd_vehicle_count= 3)
      AND date_dim.d_dow in (6,0)
      AND date_dim.d_year in (1999,2000,2001)
      AND store.s_city in ('Midway','Fairview')
      AND ss_customer_sk = c_customer_sk
      AND customer.c_current_addr_sk = current_addr.ca_address_sk
      AND current_addr.ca_city <> customer_address.ca_city
   GROUP BY
ss_ticket_number,ss_customer_sk,ss_addr_sk,customer_address.ca_city,current_addr.ca_city,c_last_name,c_first_
name
)
ORDER BY c_last_name, c_first_name, ca_city, bought_city, ss_ticket_number;
```

4.5.1.4 Execution Time before and after query changes



4.5.2 Query 89

4.5.2.1 Original Query

```
SELECT * FROM (
  SELECT  * FROM (
    SELECT
      i_category,
      i_class,
      i_brand,
      s_store_name, s_company_name,
      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)
avg_monthly_sales
    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 IN (2000) AND ((
```

```
        i_category IN ('Home','Books','Electronics') AND
        i_class IN ('wallpaper','parenting','musical')
       ) OR (
        i_category IN ('Shoes','Jewelry','Men') AND
        i_class IN ('womens','birdal','pants')
      ))
   GROUP BY i_category, i_class, i_brand, s_store_name, s_company_name, d_moy
   ) tmp1
 WHERE CASE WHEN (avg_monthly_sales <> 0) THEN (abs(sum_sales - avg_monthly_sales) /
avg_monthly_sales) ELSE NULL END > 0.1
 ORDER BY sum_sales - avg_monthly_sales, s_store_name
) ;
```

4.5.2.2 Changes made

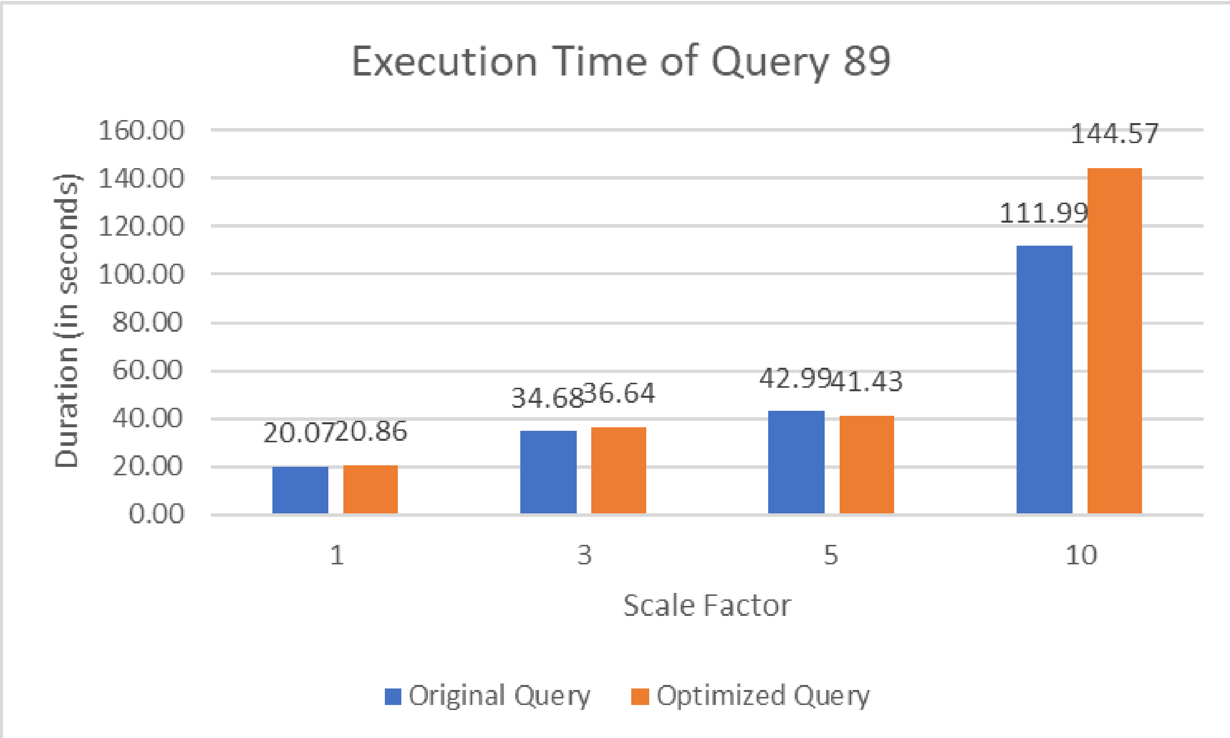
1. Instead of calculating complex logic in **WHERE** predicate, it is precomputed, which is used later.
2. Calculation done in **ORDER BY** clause is also precomputed.
3. **OR** operation is modified by using **UNION ALL.**

4.5.2.3 Optimized Query

```
SELECT
        i_category,
        i_class,
        i_brAND,
        s_store_name,
        s_company_name,
        d_moy,
        sum_sales,
        avg_monthly_sales
FROM (
        SELECT
                i_category,
                i_class,
                i_brAND,
        s_store_name,
                s_company_name,
        d_moy,
                sum_sales,
                avg_monthly_sales,
sum_sales - avg_monthly_sales AS diff,
                CASE WHEN (avg_monthly_sales <> 0) THEN (ABS(sum_sales - avg_monthly_sales) /
avg_monthly_sales) ELSE NULL END AS calc_val FROM (
```

```
                    SELECT
                            i_category,
                            i_class,
                            i_brAND,
                            s_store_name,
                            s_company_name,
                            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) avg_monthly_sales FROM ((
                                    SELECT
                                            * 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=2000 AND i_category IN
('Home','Books','Electronics') AND i_class IN ('wallpaper','parenting','musical')
                            )
                            UNION ALL
                            SELECT
                                    *
                            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=2000 AND         (i_category in ('Shoes','Jewelry','Men')
AND i_class in ('womens','birdal','pants'))
                )
        group by i_category, i_class, i_brAND, s_store_name, s_company_name, d_moy
        ) tmp1
)
WHERE calc_val > 0.1
ORDER BY diff, s_store_name;
```

4.5.2.4 Execution time before and after query changes



Chapter 5 - Conclusion

In conclusion, the TPC-DS benchmarking project has provided us with valuable insights into the complexities and challenges of evaluating a system's performance. Throughout this endeavor, we have encountered numerous considerations and critical decisions that have a significant impact on the results obtained.

One of the key takeaways from this project is the importance of meticulousness when assessing a system. Benchmarking is a crucial step in making informed decisions when choosing between different systems. It serves as a critical tool for understanding how a system performs under specific workloads and conditions.

Ultimately, this project has highlighted that rigorous benchmarking is essential for making informed decisions about system selection and performance evaluation. It underscores the need for careful consideration and analysis to ensure that the results are reliable and relevant to the specific requirements and objectives of the organization.

References

[1] Vaisman, Alejandro, and Esteban Zimányi (2022). *Data Warehouse Systems: Design and Implementation*. Springer Berlin Heidelberg.

[2] "DS Specification V 1.0.0L." *TPC-DS Specification V 1.0.0L*, 12 November 2015, https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-DS_v3.2.0.pdf.

[3] Gray, P., & Watson, H.J. (1998). *Decision Support in the Data Warehouse*. Prentice Hall PTR.

[4] Oracle. (2021). Introduction to Data Warehouse Concepts. Oracle Database 21c Documentation. Retrieved from https://docs.oracle.com/en/database/oracle/oracle-database/21/dwhsg/introduction-data-warehouse-concepts.html#GUID-C3AD27A3-970A-442D-8B18-86B79D643F25