TPC-DS benchmark implementation on MySQL

INFO-H-419: Data Warehouses
Professor Esteban Zimanyi
Université Libre de Bruxelles (ULB)

Ali AbuSaleh
Liliia Aliakberova
Muhammad Rizwan Khalid
Mariana Mayorga Llano

Table of Contents

1 Introduction

The report is devoted to the use of MySQL for the implementation of Benchmark in accordance with the TPC-DS specification version 3.2.0. The Benchmark was performed using Ubuntu 18.04.6 LTS (64-bit), with specifications of Memory 16GB DDR4, 11th Gen Intel Core i5-11300H (3.80GHz * 8) and 105 GB SSD storage. The results of the TPC-DS are published https://github.com/aliabusaleh/dbma-data-warehouse.

1.1 TPC-DS Benchmark Abstract

Transaction Processing Performance Council (TPC) is a non-profit cooperation of companies that generates, provides the details, supports, and supervises benchmarking. Regarding data warehouses, the most relevant TPC benchmarks are the Decision Support Benchmark (TPC-DS) and the Data Integration Support Benchmark (TPC-DI).

The TPC-Decision Support benchmark that models several generally applicable aspects of a decision support system, including queries and data maintenance (TPC. 2021). Considering that the purpose of TPC-DS benchmarks is to provide relevant, objective performance data to industry users, it provides a representative evaluation of the System Under Test's (SUT) performance as a general-purpose decision support system.

1.2 MySQL Abstract

MySQL is an open-source relational database. The MySQL Database Software is a client/server system that consists of a multithreaded SQL server that supports different back ends, several different client programs and libraries, administrative tools, and a wide range of application programming interfaces (APIs) [1].

MySQL Server is intended for mission-critical, heavy-load production systems as well as for embedding into mass-deployed software [2]. Additionally, MySQL is characterized by high flexibility and scalability due to the ability to integrate data from various types of tables.

2 Methodology

The main concept in TPC-DS benchmarking is to provide a fair measure to test different database vendors, and in this project, we are testing MySQL, since it's one of the most common relational databases used around the globe [3]. This section will demonstrate the procedures, technology, and will walk through the process of doing this project.

To assure consistency in the results given during this benchmark, all queries in all scale factors were executed on the same device under the most similar conditions possible.

2.1 Tools Used

The benchmark was conducted utilizing the following tools:

1. TPC materials and programs;

2. MySQL Community Server 8.0.30;
3. MySQL Workbench 8.0.30;
4. Command-line interface tools;
5. Python.

Data generation for benchmark was performed using TPC-DS data generator at 4 scale factors (SF):
1. SF 1;
2. SF 2;
3. SF 5;
4. SF 10.

The data population was conducted using the Python programming language in PyCharm Integrated Development Environment.

Two scientific articles were used as a scientific basis for the TPC-DS benchmark and the report preparation, namely:
1. The Making of TPC-DS (https://www.researchgate.net/publication/221311196_The_Making_of_TPC-DS).
2. An overview of decision support benchmarks: TPC-DS, TPC-H and SSB (https://www.researchgate.net/publication/283797188_An_overview_of_decision_support_benchmarks_TPC-DS_TPC-H_and_SSB).

During the implementation of TPC-DS benchmark there were used two git repositories, specifically:
1. https://github.com/gregrahn/tpcds-kit
2. https://github.com/damjad/bdma-datawarehouse/tree/master/tpc-ds-postgres

2.2 Benchmark Implementation Stages

There are several stages of Benchmark Implementation:
1. Data Warehouse Schema Generation;
2. Data Generation;
3. Data Loading (Load Test);
4. MySQL Query Creation and Adaptation;
5. Power Test for 4 Scale Factors.

2.2.1 Data Warehouse Schema Generation Stage

To create the schema of the data warehouse the “tpcds.sql” file must be used. The file consists of SQL query that generates the data warehouse tables. The tpcds.sql file is provided as part of TPC materials and programs [4].

Data Warehouse Schema models the sales and returns process for a company. There are three main sales channels for the company namely catalogs, stores and the Internet. The scheme includes 25 tables including 6 data warehouse fact tables. Below are the seven fact tables for the TPC data warehouse:
1. Store Sales;
2. Store Returns;

3. Catalog Sales;
4. Catalog Returns;
5. Web Sales;
6. Web Returns.

2.2.2 Data Generation, Data Load and Load Test Stages

Data generation stage is performed using the “dsdgen” utility that generates input data. The work with the utility occurs in command-line interface tools. The name of the utility, the volume of the factor and the specific directory must be written in a command-line to execute the data generation.

Load Test entails time to create database schema, generate data followed by loading it to the System Under Test (SUT) i.e. MySQL. A bash script was written to generate data for a given scale factor. The data generated by the “dsdgen” utility is delimited by the pipe (“|”) delimiter by default. A null is represented by two or more consecutive delimiters. MySQL does not identify this as null and returns a warning and may cause the data loss. “\N” is recognized by MySQL as null. In order to generate mysql compatible data, “dsdgen” utility was used. Additionally, the used Linux “sed” command was utilized to replace two or more empty delimiters with “\N”. Linux Command:

```
#For Ubuntu Linux
LC_CTYPE=C && sed -i -e 's_^|_\\N|_g' -e 's_||_|\\N|_g' -e 's_||_|\\N|_g' $DATA_DIR/*.dat
```

LC_CTYPE is set to old standard C to avoid any type of encoding issues followed by a combination of regex expressions to replace empty delimiters with “\N”. Following sample line is taken from “call_center” data file:

```
1|AAAAAAAABAAAAAAA|1998-01-01|||2450952|NY Metro|large|135|76815|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|Pleasant
Hill|Williamson County|TN|33604|United States|-5|0.11|
```

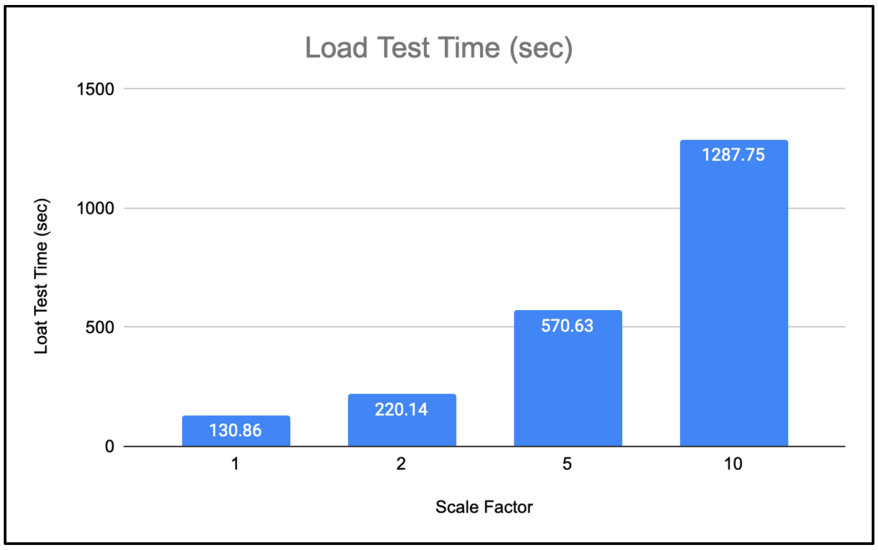
Two consecutive empty delimiters can be seen after the date, this will cause MySQL to return a warning and after applying “sed” operation:

```
1|AAAAAAAABAAAAAAA|1998-01-01|\N|\N|2450952|NY Metro|large|135|76815|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|Pleasant
Hill|Williamson County|TN|33604|United States|-5|0.11|
```

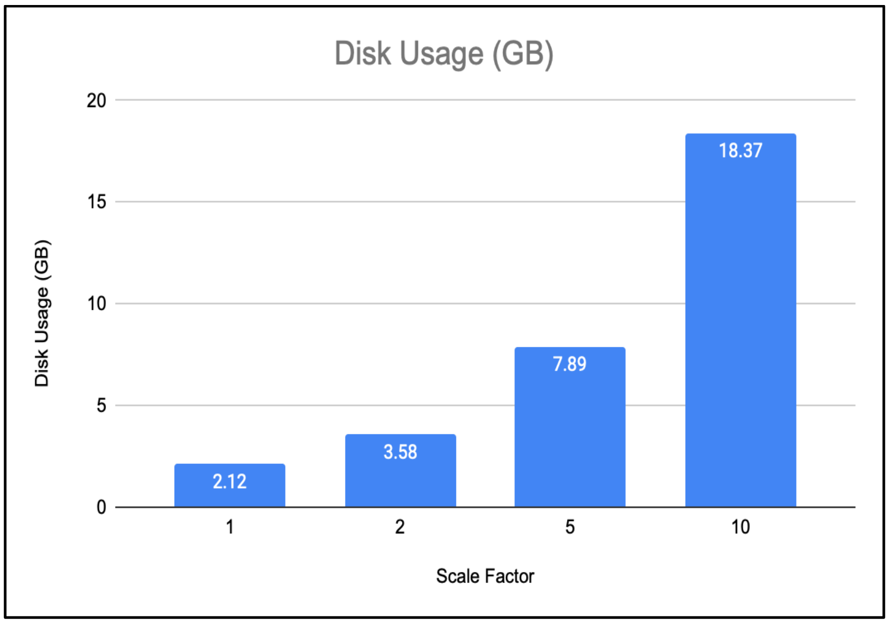
Data Generation and Data Loading operations were combined in the bash file. For loading the data into MySQL from the local system files generated by “dsdgen” utility, local_infile should be enabled otherwise an error message is returned by MySQL that loading data from the local file is disabled by default. This can be achieved by adding the below property to MySQL configuration file:

```
[mysqld]
# Only allow connections from localhost
bind-address = 127.0.0.1
mysqlx-bind-address = 127.0.0.1
local_infile = 1
```

The below charts depict the time taken for the load testing and disk usage against mentioned 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 130,86 sec. for SF 1 factor up to 1287,75 sec. for SF 10. The Load Test indicator raised by almost 10 times.



On the diagram shown Disc Usage in GB for each scale factor. There is a rapid growth of Disc Usage from 1 SF to 10 SF. The indicator increased from 2 GB up to 18 GB. It is important to mention that both charts prove the significant influence in terms of Test Load Time and Disc Usage according to the selected scale factors.

The Load Test was done based on the instructions from the TCP-DS documentation Version 2.10.0. This test measures the Database Load Time, known as TLOAD, which is the elapsed time taken by the system to create and prepare the database for the execution of the performance test. In this case, our Load Test included:

1. Database creation;
2. Data generation / data loading.

The Load Test does not include the execution of any of the queries in the Power Test. This is the difference between Load Start Time and Load End Time. Conditions:

1. There cannot be any manual intervention during the Database Load.
2. The system under test (SUT) or any component of it must not be restarted after the start of the Load Test and before the start of the Performance Test.

2.2.3 Validation of Loaded Data

A manual row count validation was performed for the scale factor 1. The rows in the db matched the number of records given in TPC-DS table 3.2. The row count was not performed for scale factor 2, 5, and 10 it was not known about the number of records for these scale factors. In specification document row count for scale factor 1 was provided along with row counts of bigger scale factor (1 TB and more). This step was performed to validate if the row count matches with TPC-DS specification.

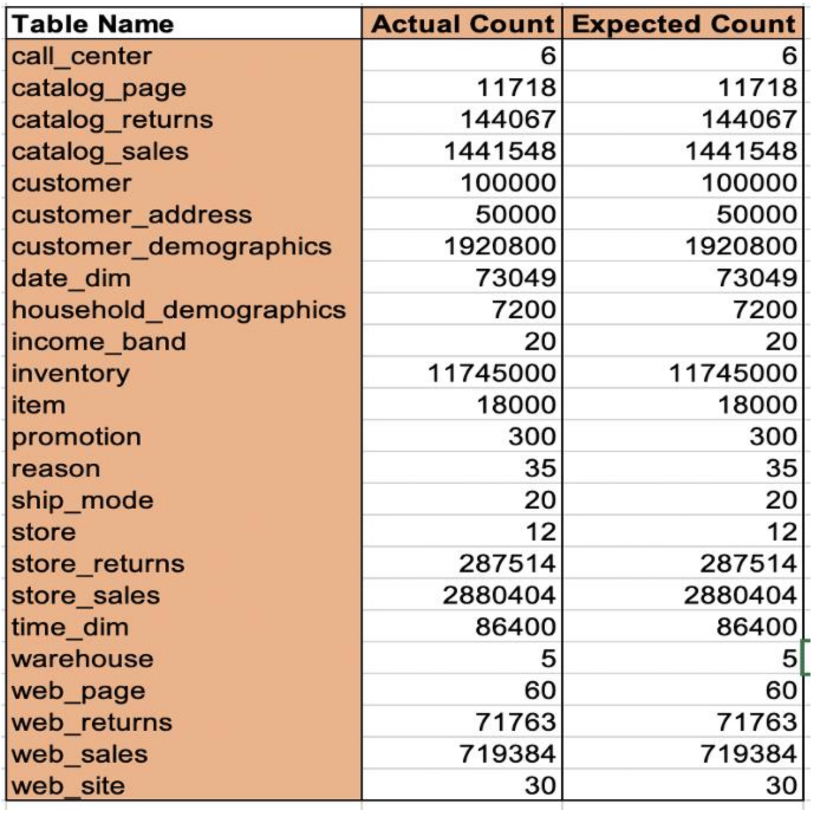


Table Name	Actual Count	Expected Count
call_center	6	6
catalog_page	11718	11718
catalog_returns	144067	144067
catalog_sales	1441548	1441548
customer	100000	100000
customer_address	50000	50000
customer_demographics	1920800	1920800
date_dim	73049	73049
household_demographics	7200	7200
income_band	20	20
inventory	11745000	11745000
item	18000	18000
promotion	300	300
reason	35	35
ship_mode	20	20
store	12	12
store_returns	287514	287514
store_sales	2880404	2880404
time_dim	86400	86400
warehouse	5	5
web_page	60	60
web_returns	71763	71763
web_sales	719384	719384
web_site	30	30

2.2.4 Query Generation

To generate queries, “dsqgen” utility provided by TPC was used. By default, this utility generates 1 stream and parses all 99 queries into a single file. So a python script was used to generate the queries into separate sql files. The “dsqgen” utility also has a dialect option but no mysql dialect was provided by the TPC. MySQL dialect was written to envelop some basic functionalities, manual modifications were also made after parsing the 99 queries. Below the dialect written to be used by “dsqgen” utility:

```
define __LIMITA = "";
define __LIMITB = "";
define __LIMITC = "limit %d";
define _BEGIN = "-- start query " + [_QUERY] + " in stream " + [_STREAM] + " using
template " + [_TEMPLATE];
define _END = "-- end query " + [_QUERY] + " in stream " + [_STREAM] + " using
template " + [_TEMPLATE];
define _CONCATA = "Concat(";
define _CONCATB = " , ifnull(";
define _CONCATC = ", ''))";
define _DATEADDA = "DATE_ADD(";
define _DATEADDB = ", INTERVAL";
define _DATEADDC = ")";
```

2.2.5 Syntactic Changes in Queries

Out of 99 queries, a total of 34 queries were not able to be executed as generated since they were not mysql compatible syntactically. Therefore, syntatic changes were required. In this section, the syntactical issues in the queries generated by the “dsqgen” utility are provided, as well as an explanation of how they were resolved:

1. Alias for every derived table. In MySQL, every derived table must have its own alias. The sample is provided below:

```
# No alias for inner sub query
SELECT ID FROM ( SELECT ID, msisdn FROM ( SELECT * FROM TT2 ) );

# alias added
SELECT ID FROM ( SELECT ID, msisdn FROM ( SELECT * FROM TT2 ) AS
T ) AS T;
```

2. Syntax issue for date interval. Queries generated by “dsqgen” were using incompatible syntax to add date intervals in the where clause. The sample is provided below:

```
# no interval keyword used
select * from employees where join_date between between cast('1998-08-04' as
date) and (cast('1998-08-04' as date) +  14 days
# mysql interval keyword and day keyword instead of days
select * from employees where join_date between between cast('1998-08-04' as
date) and (cast('1998-08-04' as date) +  interval 14 day
```

3. Syntax issue for group by rollup. There is no group by rollup in MySQL. Equivalent is group by *attributes* with rollup. The sample is provided below:

```
# group by rollup not compatible with MySQL
Select channel, id, count(*) from store_sales group by rollup (channel,id)

# group with rollup in MySQL
Select channel, id, count(*) from store_sales group by channel,id with rollup
```

4. No intersect operator in mysql. Some of the dsqgen queries were using the intersect operator which is not available in MySQL. The sample is provided below:

```
# intersect operator not available in mysql
SELECT ClientId FROM Commandes
INTERSECT
SELECT ClientId FROM Clients;

# IN operator was used
SELECT Commandes.ClientId
FROM Commandes
WHERE Commandes.ClientId IN (SELECT Clients.ClientId FROM Clients);
```

5. No full outer join in MySQL. Few of the dsdgen queries were using full outer join and this functionality is not available in MySQL. The sample is provided below:

```
# full outer join not available in mysql
SELECT * FROM t1 FULL OUTER JOIN t2 ON t1.Name = t2.Name;

# performing union on left join and right join
SELECT * FROM t1 LEFT JOIN t2 ON t1.id = t2.id
UNION
SELECT * FROM t1 RIGHT JOIN t2 ON t1.id = t2.id
```

6. No Except operator in MySQL. There is no except operator in MySQL so an alternate method should be adopted in mysql. The sample is provided below:

```
# No except operator in MySQL
```

```
SELECT * FROM Tab1
EXCEPT
SELECT * FROM Tab1 WHERE int_attribute_of_Tab1>0

# Not operator was used to perform except operation
SELECT * FROM Tab1
WHERE id NOT IN ( SELECT id FROM Tab1 WHERE int_attribute_of_Tab1>0 )
```

2.2.6 Query Execution

Bash scripts were implemented to execute queries using parallel cores. However, in parallel core scenarios, connectivity issues with the MySQL database connection appeared, so the execution was implemented with 1 core instead.

The Linux time command was used to log the elapsed time it took to execute each query. A bash script was formed that produces 3 output files. Log file containing the time of execution of query, result file containing the output of the query, and the error file containing the error thrown by the MySQL. The sample snippet to run one query is provided below:

```
/usr/bin/time -o "$file_path/`basename ${file_name%.*}`.log" mysql -uuser -ppassword -
Ddatabase < $file_path > "$file_path/`basename ${file_name%.*}`.res" 2>
"$file_path/`basename ${file_name%.*}`.err"
```

2.2.7 Result Accumulation

In order to accumulate all the execution time into a csv file from previously formed query log files, an additional bash script. Subsequently, the csv file was used for results visualization.

```
for f in `ls $RESULTS_DIR/*.log`
   do TIME=`head -1 $f |  awk '{print $3}' | awk '{sub(/\(.*/, "", $(NF-1)); print $1}'`
   echo "`basename ${f%.*>}`,$TIME" >> $RESULTS_DIR/$CSV_NAME
done
```

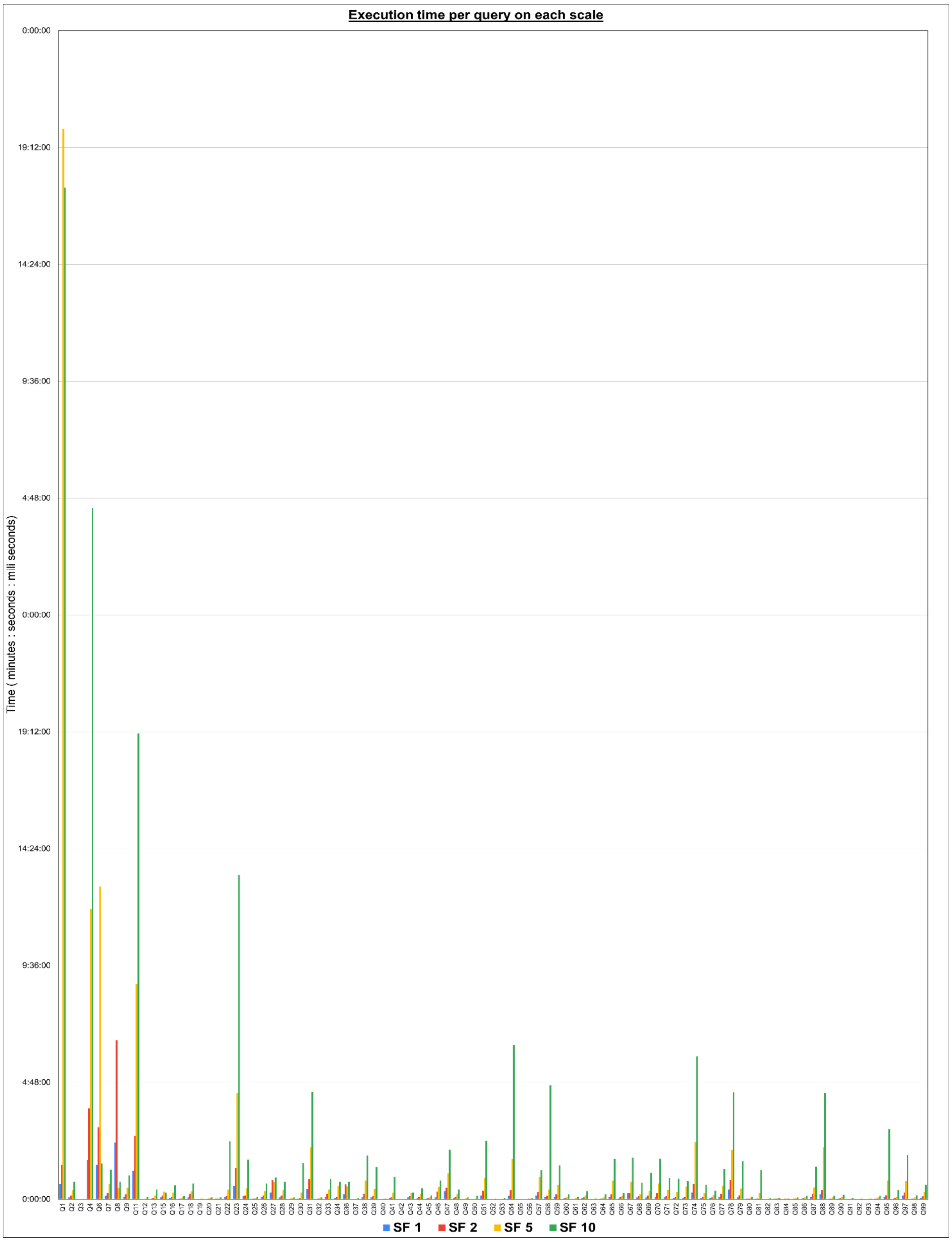
3 Observations

3.1 Individual Query Execution

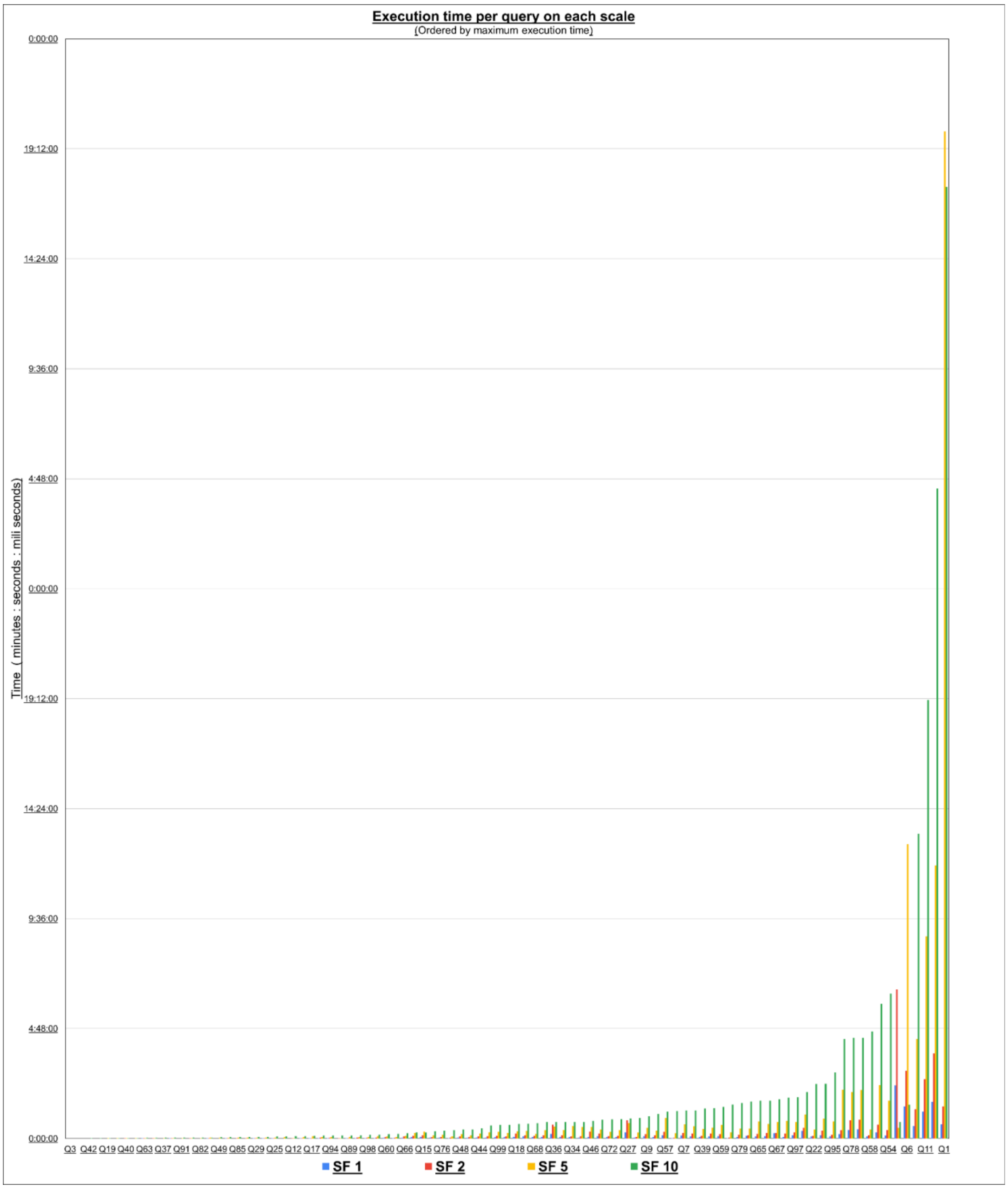
The data analysis phase began after executing the queries and gathering the results for each scale factor on each query. For this stage of the process, the queries were categorized according to their overall performance (high, intermediate-hig, intermediate-low and low) and their historic performance evolution along the scale factores.

In this section, each graph is followed by the table with the data that was used for its generation along with a description of the main observations made on each case.

The first approach to analyse the data was done by visualizing the results for all queries in each scale factor ordered by the query number. These results can be found in the following graph:



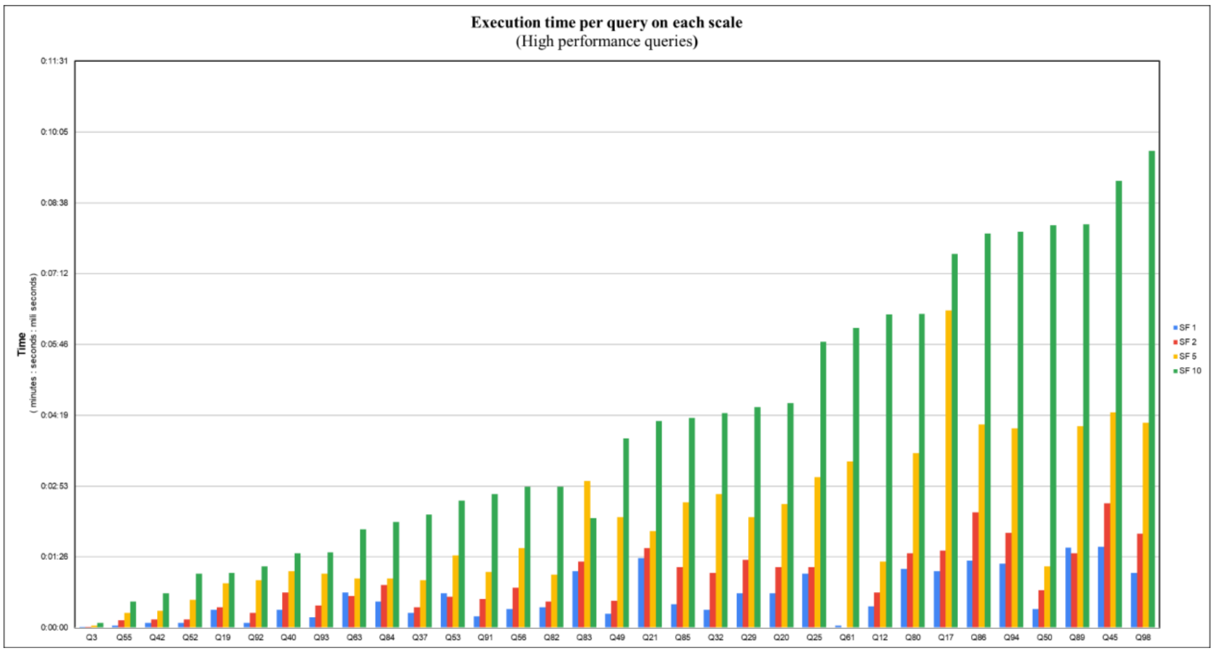
Secondly, the queries were ordered by performance based on the maximum execution time elapsed among their four scale factors, where a high difference in performance can be appreciated from the fastest query (Q3 with less than 0:00:01 ms in SF1 and SF2) to the slowest one (Q1 with 43:57:37 ms in SF5).



3.1.1 Performance classification

For visualization purposes, the queries were divided in four groups considering for each of them the maximum amount of time elapse among its scale factors:

High performance queries
This group contains the queries where its maximum performance time along all scale factors was less than 10 seconds.



For every query in this set, the amount of time elapsed for execution goes from:

➔ < 0:00:01 minutes (Q63) to 0:09:42 minutes (Q98) regardless of their scale factor

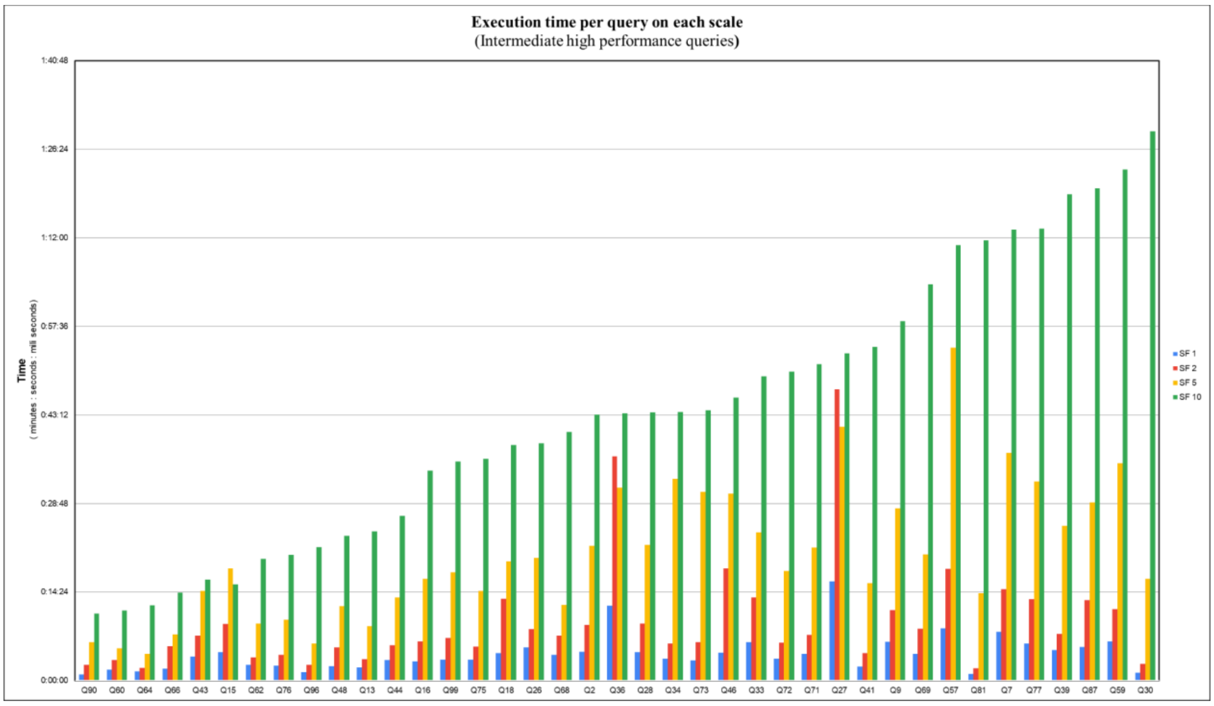
➔ 0:00:06 minutes (Q3) to 0:09:42 minutes (Q98) for the slowest execution per query among the scales

	SF 1	***SF 2***	***SF 5***	***SF 10***	***Slowest SF***	***Fastest SF***
Q3	0:00:01	0:00:01	0:00:03	0:00:06	0:00:06	0:00:01
Q55	0:00:03	0:00:09	0:00:18	0:00:32	0:00:32	0:00:03
Q42	0:00:06	0:00:10	0:00:21	0:00:42	0:00:42	0:00:06
Q52	0:00:06	0:00:10	0:00:34	0:01:06	0:01:06	0:00:06
Q19	0:00:22	0:00:25	0:00:54	0:01:07	0:01:07	0:00:22
Q92	0:00:06	0:00:18	0:00:58	0:01:15	0:01:15	0:00:06
Q40	0:00:22	0:00:43	0:01:09	0:01:31	0:01:31	0:00:22
Q93	0:00:13	0:00:27	0:01:06	0:01:32	0:01:32	0:00:13
Q63	0:00:43	0:00:39	0:01:00	0:02:00	0:02:00	0:00:39
Q84	0:00:32	0:00:52	0:01:00	0:02:09	0:02:09	0:00:32
Q37	0:00:18	0:00:25	0:00:58	0:02:18	0:02:18	0:00:18
Q53	0:00:42	0:00:38	0:01:28	0:02:35	0:02:35	0:00:38
Q91	0:00:14	0:00:35	0:01:08	0:02:43	0:02:43	0:00:14
Q56	0:00:23	0:00:49	0:01:37	0:02:52	0:02:52	0:00:23
Q82	0:00:25	0:00:32	0:01:05	0:02:52	0:02:52	0:00:25
Q83	0:01:09	0:01:21	0:02:59	0:02:14	0:02:59	0:01:09
Q49	0:00:17	0:00:33	0:02:15	0:03:51	0:03:51	0:00:17
Q21	0:01:25	0:01:37	0:01:58	0:04:12	0:04:12	0:01:25
Q85	0:00:29	0:01:14	0:02:33	0:04:16	0:04:16	0:00:29
Q32	0:00:22	0:01:07	0:02:43	0:04:22	0:04:22	0:00:22
Q29	0:00:42	0:01:23	0:02:15	0:04:29	0:04:29	0:00:42
Q20	0:00:42	0:01:14	0:02:31	0:04:34	0:04:34	0:00:42
Q25	0:01:06	0:01:14	0:03:04	0:05:49	0:05:49	0:01:06

Q61	0:00:03	0:00:00	0:03:23	0:06:06	0:06:06	0:00:00
Q12	0:00:26	0:00:43	0:01:21	0:06:22	0:06:22	0:00:26
Q80	0:01:12	0:01:31	0:03:33	0:06:23	0:06:23	0:01:12
Q17	0:01:09	0:01:34	0:06:27	0:07:36	0:07:36	0:01:09
Q86	0:01:22	0:02:21	0:04:08	0:08:01	0:08:01	0:01:22
Q94	0:01:18	0:01:56	0:04:03	0:08:03	0:08:03	0:01:18
Q50	0:00:23	0:00:46	0:01:15	0:08:11	0:08:11	0:00:23
Q89	0:01:38	0:01:31	0:04:06	0:08:12	0:08:12	0:01:31
Q45	0:01:39	0:02:32	0:04:23	0:09:05	0:09:05	0:01:39
Q98	0:01:07	0:01:55	0:04:10	0:09:42	0:09:42	0:01:07

Intermediate-High performance queries

This group contains the queries where its maximum performance time along all scale factors was over the 10 seconds but under 1.5 minutes.



For every query in this set, the amount of time elapsed for execution goes from:

➔ 0:01:01 minutes (Q90) to 1:29:17 minutes (Q30) regardless of their scale factor

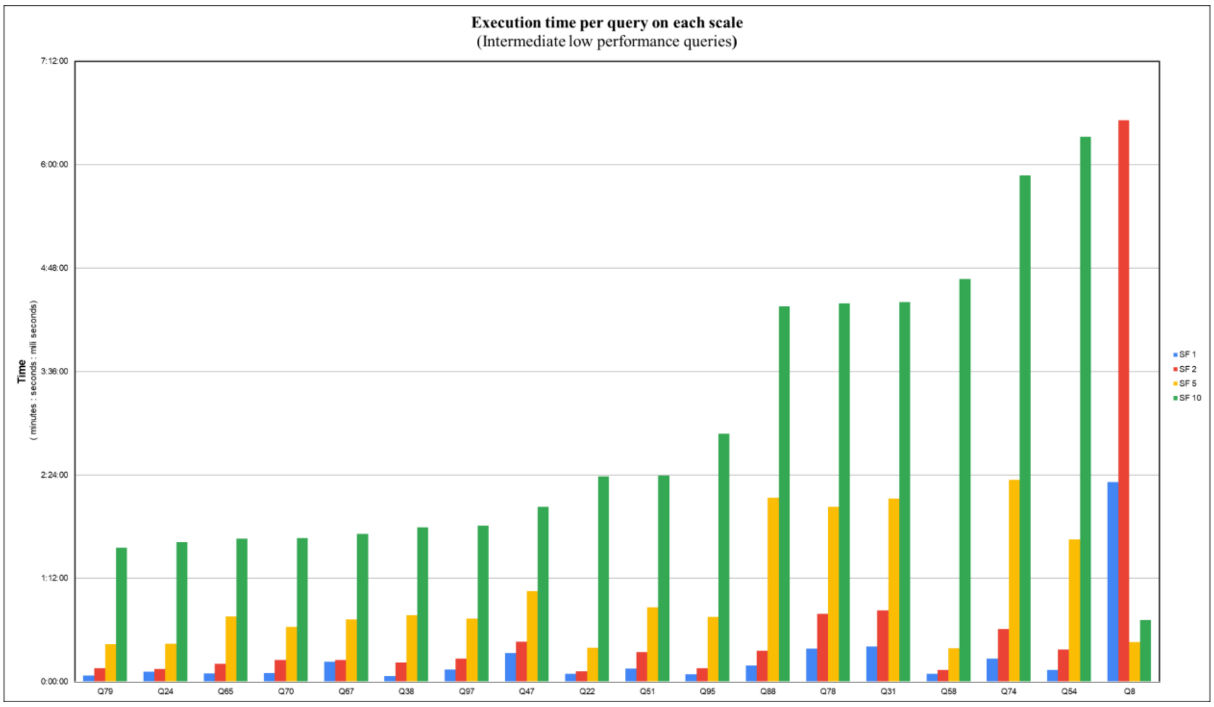
➔ 0:10:56 minutes (Q3) to 0:1:29:17 minutes (Q30) for the slowest execution per query among the scales

	SF 1	*SF 2*	*SF 5*	*SF 10*	*Slowest*	*Fastest*
Q90	0:01:01	0:02:32	0:06:16	0:10:56	0:10:56	0:01:01
Q60	0:01:48	0:03:22	0:05:17	0:11:25	0:11:25	0:01:48
Q64	0:01:31	0:02:03	0:04:22	0:12:14	0:12:14	0:01:31
Q66	0:01:56	0:05:38	0:07:29	0:14:18	0:14:18	0:01:56
Q43	0:03:53	0:07:18	0:14:33	0:16:26	0:16:26	0:03:53
Q15	0:04:35	0:09:14	0:18:15	0:15:37	0:18:15	0:04:35
Q62	0:02:32	0:03:47	0:09:17	0:19:48	0:19:48	0:02:32

Q76	0:02:27	0:04:11	0:09:53	0:20:27	0:20:27	0:02:27
Q96	0:01:20	0:02:32	0:06:01	0:21:42	0:21:42	0:01:20
Q48	0:02:20	0:05:25	0:12:05	0:23:33	0:23:33	0:02:20
Q13	0:02:09	0:03:29	0:08:53	0:24:14	0:24:14	0:02:09
Q44	0:03:21	0:05:45	0:13:30	0:26:49	0:26:49	0:03:21
Q16	0:03:07	0:06:21	0:16:35	0:34:09	0:34:09	0:03:07
Q99	0:03:24	0:06:55	0:17:37	0:35:39	0:35:39	0:03:24
Q75	0:03:24	0:05:30	0:14:35	0:36:01	0:36:01	0:03:24
Q18	0:04:30	0:13:19	0:19:22	0:38:17	0:38:17	0:04:30
Q26	0:05:22	0:08:21	0:19:57	0:38:37	0:38:37	0:05:22
Q68	0:04:12	0:07:19	0:12:20	0:40:24	0:40:24	0:04:12
Q2	0:04:40	0:09:04	0:21:55	0:43:16	0:43:16	0:04:40
Q36	0:12:11	0:36:30	0:31:22	0:43:29	0:43:29	0:12:11
Q28	0:04:36	0:09:16	0:22:04	0:43:38	0:43:38	0:04:36
Q34	0:03:33	0:06:01	0:32:46	0:43:39	0:43:39	0:03:33
Q73	0:03:18	0:06:13	0:30:42	0:43:57	0:43:57	0:03:18
Q46	0:04:32	0:18:16	0:30:24	0:46:01	0:46:01	0:04:32
Q33	0:06:13	0:13:29	0:24:05	0:49:29	0:49:29	0:06:13
Q72	0:03:33	0:06:12	0:17:48	0:50:13	0:50:13	0:03:33
Q71	0:04:20	0:07:25	0:21:38	0:51:25	0:51:25	0:04:20
Q27	0:16:07	0:47:23	0:41:16	0:53:13	0:53:13	0:16:07
Q41	0:02:16	0:04:27	0:15:50	0:54:18	0:54:18	0:02:16
Q9	0:06:20	0:11:26	0:28:01	0:58:28	0:58:28	0:06:20
Q69	0:04:21	0:08:26	0:20:31	1:04:27	1:04:27	0:04:21
Q57	0:08:31	0:18:10	0:54:08	1:10:48	1:10:48	0:08:31
Q81	0:01:03	0:02:02	0:14:12	1:11:36	1:11:36	0:01:03
Q7	0:07:58	0:14:52	0:37:03	1:13:20	1:13:20	0:07:58
Q77	0:06:03	0:13:16	0:32:22	1:13:30	1:13:30	0:06:03
Q39	0:04:59	0:07:37	0:25:11	1:19:05	1:19:05	0:04:59
Q87	0:05:27	0:13:06	0:29:00	1:20:04	1:20:04	0:05:27
Q59	0:06:21	0:11:35	0:35:22	1:23:04	1:23:04	0:06:21
Q30	0:01:17	0:02:42	0:16:34	1:29:17	1:29:17	0:01:17

Intermediate-Low performance queries

This group contains the queries where its maximum performance time along all scale factors was more than 1.5 minutes but less than 10 minutes.

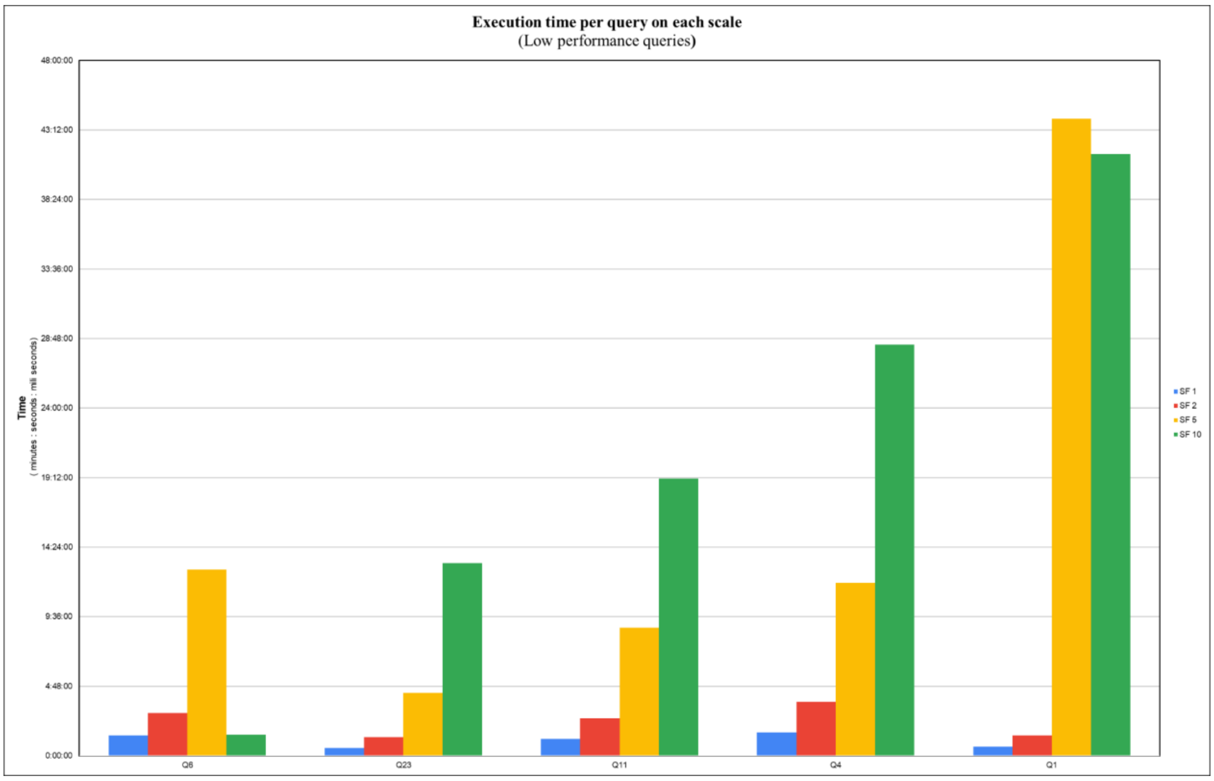


For every query in this set, the amount of time elapsed for execution goes from:
➔ 0:04:08 minutes (Q38) to 6:31:03 minutes (Q8) regardless of their scale factor
➔ 1:33:13 minutes (Q79) to 6:31:03 minutes (Q8) for the slowest execution per query among the scales

	SF 1	***SF 2***	***SF 5***	***SF 10***	***Slowest***	***Fastest***
Q79	0:04:15	0:09:31	0:26:03	1:33:13	1:33:13	0:04:15
Q24	0:07:01	0:08:48	0:26:17	1:37:07	1:37:07	0:07:01
Q65	0:05:48	0:12:21	0:45:32	1:39:38	1:39:38	0:05:48
Q70	0:06:08	0:15:10	0:38:13	1:40:02	1:40:02	0:06:08
Q67	0:14:02	0:15:11	0:43:27	1:42:53	1:42:53	0:14:02
Q38	0:04:08	0:13:28	0:46:24	1:47:40	1:47:40	0:04:08
Q97	0:08:38	0:16:07	0:43:57	1:48:42	1:48:42	0:08:38
Q47	0:20:01	0:27:47	1:03:09	2:01:38	2:01:38	0:20:01
Q22	0:05:29	0:07:26	0:23:32	2:22:58	2:22:58	0:05:29
Q51	0:09:14	0:20:30	0:51:48	2:23:31	2:23:31	0:09:14
Q95	0:05:17	0:09:25	0:45:09	2:52:37	2:52:37	0:05:17
Q88	0:11:13	0:21:33	2:08:04	4:21:26	4:21:26	0:11:13
Q78	0:23:08	0:47:19	2:01:42	4:23:34	4:23:34	0:23:08
Q31	0:24:28	0:49:39	2:07:23	4:24:19	4:24:19	0:24:28
Q58	0:05:39	0:08:20	0:23:20	4:40:18	4:40:18	0:05:39
Q74	0:16:11	0:36:44	2:20:27	5:52:20	5:52:20	0:16:11
Q54	0:08:18	0:22:20	1:39:06	6:19:17	6:19:17	0:08:18
Q8	2:19:08	6:31:03	0:27:36	0:43:09	6:31:03	0:27:36

Low performance queries

Finally, the low performance queries category contains the queries where its maximum performance time along all scale factors was more than 10 minutes.



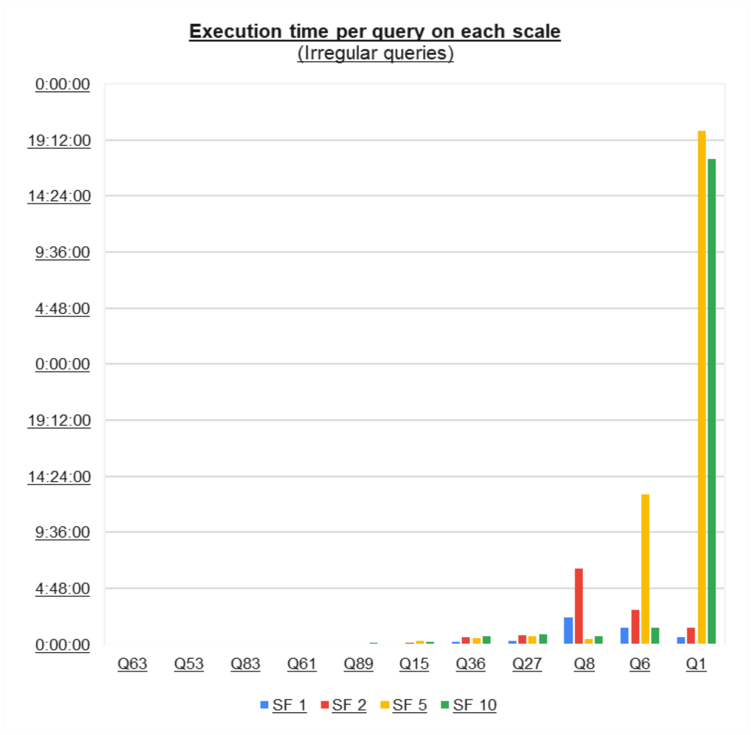
For every query in this set, the amount of time elapsed for execution goes from:
➔ 0:32:51 minutes (Q23) to 43:57:37 minutes (Q1) regardless of their scale factor
➔ 12:50:28 minutes (Q79) to 43:57:37 minutes (Q1) for the slowest execution per query among the scales

	SF 1	***SF 2***	***SF 5***	***SF 10***	***Slowest***	***Fastest***
Q6	1:24:10	2:57:29	12:50:28	1:28:19	12:50:28	1:24:10
Q23	0:32:51	1:17:21	4:21:35	13:18:43	13:18:43	0:32:51
Q11	1:10:30	2:36:00	8:49:42	19:08:05	19:08:05	1:10:30
Q4	1:36:24	3:43:39	11:55:08	28:22:17	28:22:17	1:36:24
Q1	0:37:09	1:24:25	43:57:37	41:32:24	43:57:37	0:37:09

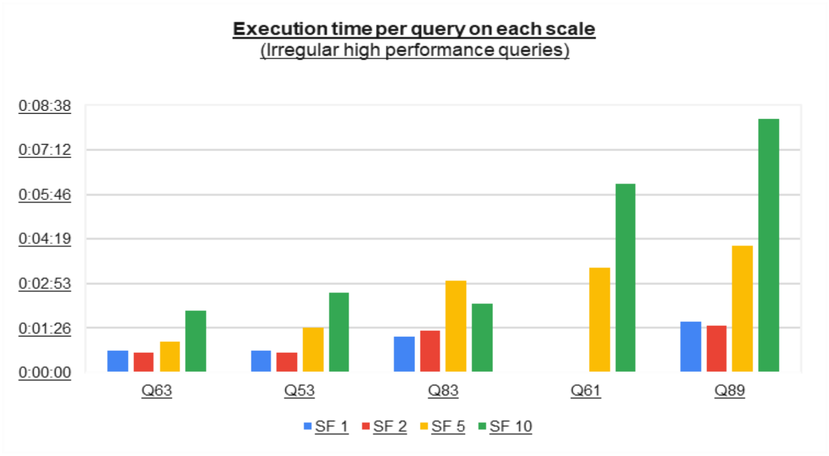
3.1.2 Queries with irregular performance along Scale Factors

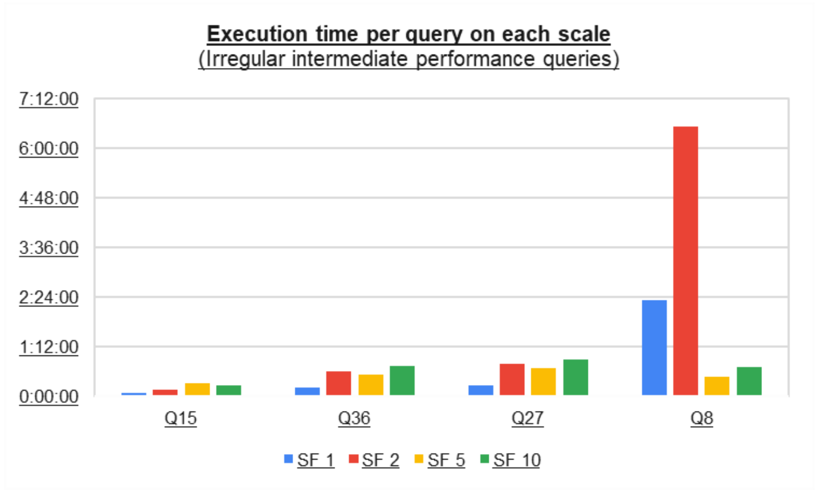
An original hypothesis, before the execution of this benchmark, was that the elapsed time of execution would increase as the scale factor did, however, looking at the results this was not always the case. These queries that presented an improvement on performance when increasing the scale factor were categorized as ‘irregular’, considering that they did not followed the expected behaviour given the hypothesis mentioned.

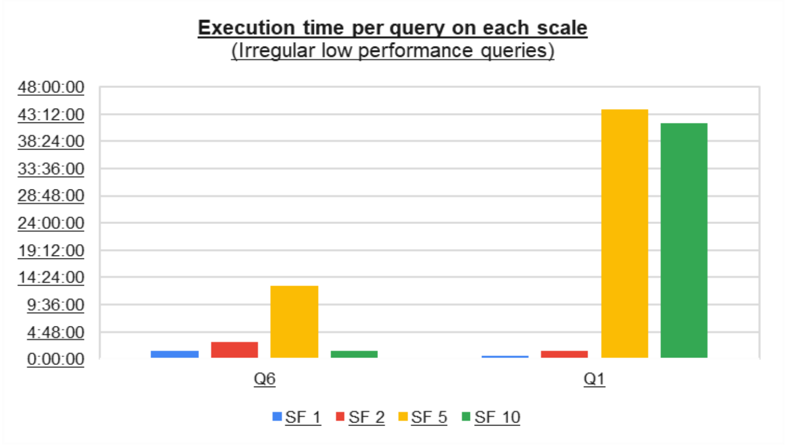
The following graph gathers all of these ‘irregular’ queries. It can be noticed that the difference in each scale performance is different for each query, where the performance of a query in a bigger scale can be better than its result in the previous Scale Factor.



To improve the visualization, these irregular queries were divided by their performance classification in the following diagrams so that their behavior along the benchmark can be better appreciated visually.







Cases in which the SF5 has better performance than SF10

	SF 5	**SF 10**
Q83	0:02:59	0:02:14
Q15	0:18:15	0:15:37
Q6	12:50:28	1:28:19
Q1	43:57:37	41:32:24

Cases in which the SF2 has better performance than SF4

	SF 2	**SF 5**
Q36	0:36:30	0:31:22
Q8	6:31:03	0:27:36

Cases in which the SF1 has better performance than SF2

	SF 1	**SF 2**
Q63	0:00:43	0:00:39
Q53	0:00:42	0:00:38
Q61	0:00:03	0:00:00
Q89	0:01:38	0:01:31

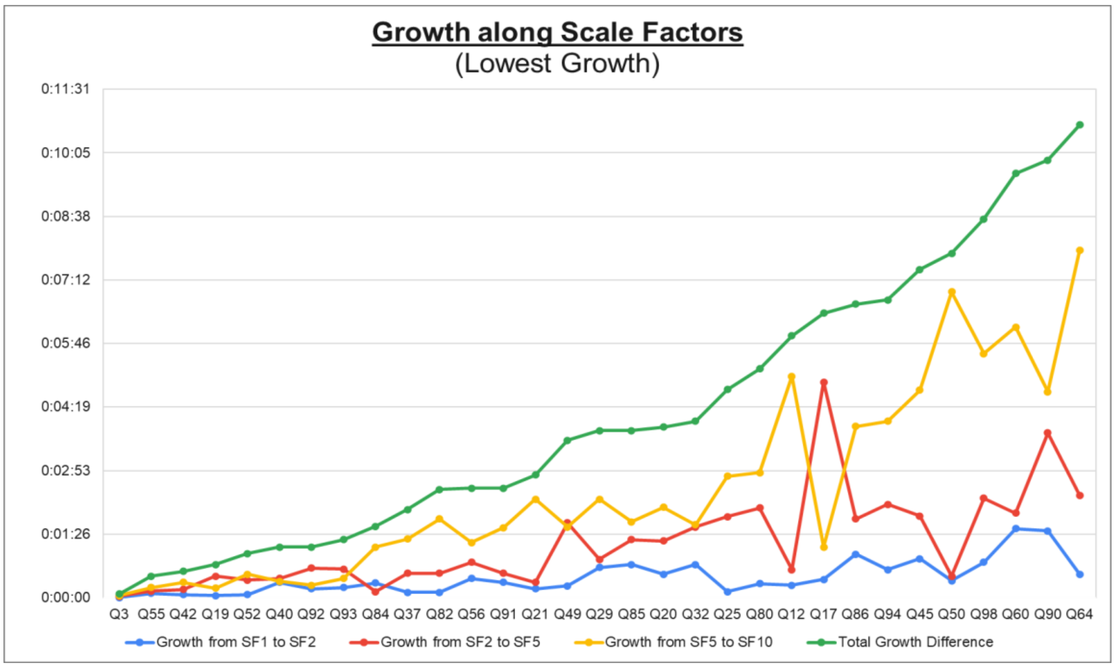
Since differences between queries in SF1 and SF2 are minor and the difference in the mentioned Scale Factors is not considerably bigger, these results are considered as possible under the hypothesis that there may have been another process on the machine that could have taken resources from it. However, for the other cases it is being considered that the difference is considerable enough and it would be required to conduct a deeper study to ensure the veracity of the values and understand the causes of the irregularities presented.

3.1.3 Queries clasification by growth along the Scale Factors

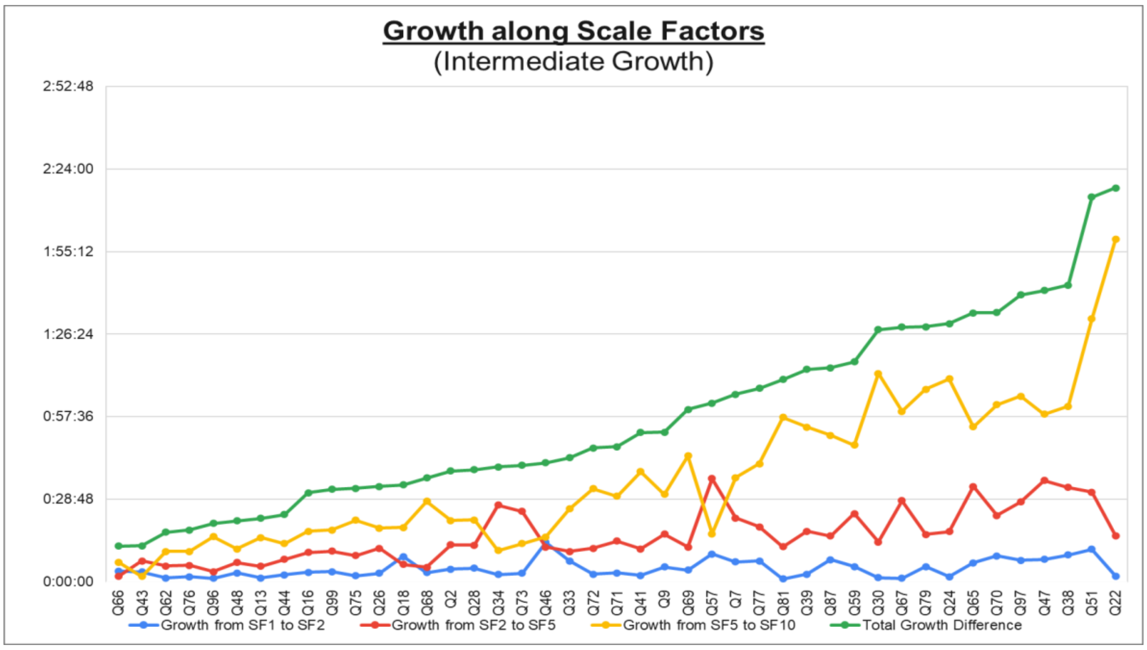
Without taking into consideration the irregular ones, the rest of the queries were analysed in therms of their growth along the increase of the Scale Factors and classified according to the total growth difference (addition of differences along all Scale Fators).

For this analysis, four values were calculated and can be appreciated in the graph below:
➔ The difference between SF1 and SF2 execution time per query
➔ The difference between SF2 and SF5 execution time per query
➔ The difference between SF5 and SF10 execution time per query
➔ The sum of the previous values to calculate the total difference along all Scale Factors

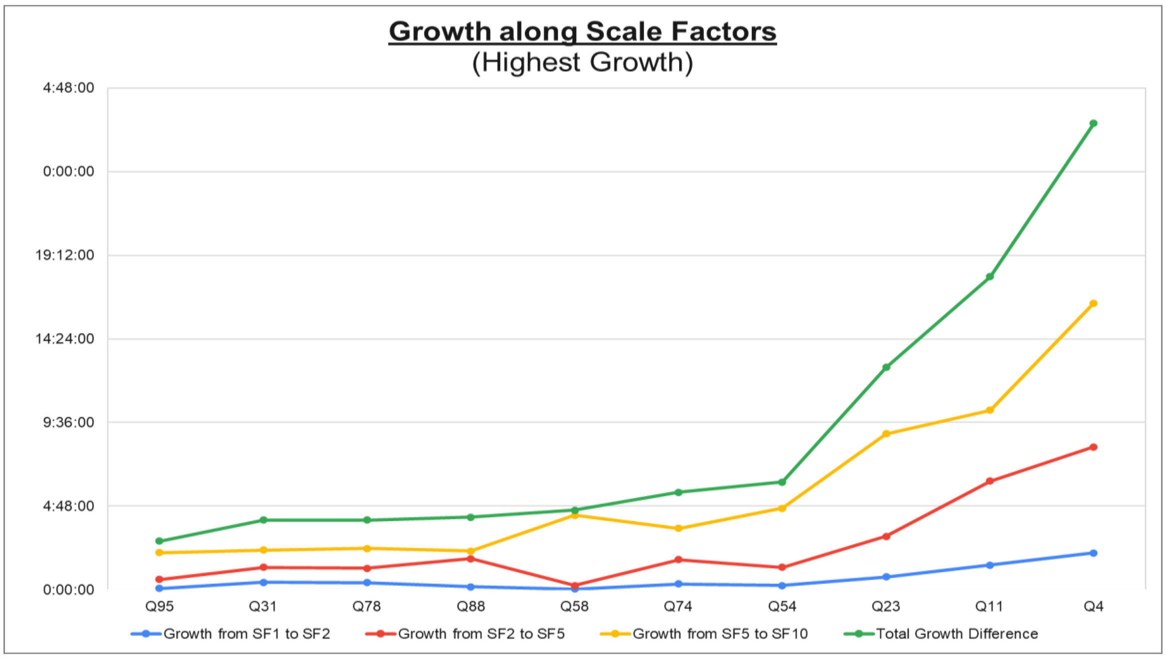
A total of 31 queries were categorized as "Lowest growth" with total growth difference going from 0:00:05 ms to 0:10:43 ms.



Regarding the "Intermediate growth", 43 queries belong to this group with total growth difference going from 0:12:22 ms to 2:17:29 ms.



Finally, in the Highest Growth" category, there are 10 queries with total growth difference going from 2:47:20 ms to 26:45:53 ms.



3.2 Optimization Attempts

Considering the results from the low performance queries, optimization attempts for Query1 were made.

1. The following modification of the query structure was tested, removing the inner subquery

```
with customer_total_return as
(select sr_customer_sk as ctr_customer_sk
,sr_store_sk as ctr_store_sk
,sum(SR_FEE) as ctr_total_return
,avg(SR_FEE) * 1.2 as avg_ctr_return
from store_returns
,date_dim
where sr_returned_date_sk = d_date_sk
and d_year =2000
group by sr_customer_sk
,sr_store_sk)
select  c_customer_id
from customer_total_return ctr1, customer_total_return ctr2
,store
,customer
where
ctr1.ctr_store_sk = ctr2.ctr_store_sk
and ctr1.ctr_total_return > ctr2.avg_ctr_return
and s_store_sk = ctr1.ctr_store_sk
and s_state = 'TN'
and ctr1.ctr_customer_sk = c_customer_sk
order by c_customer_id
limit 100;
```

2. The following indexes were created

```
create index sr_customer_sk_idx using hash on store_returns(sr_customer_sk);
create index s_state_idx using hash on store(s_state);
create index sr_fee_idex using btree on store_returns(sr_fee);

ALTER TABLE `customer` ADD INDEX `customer_idx_c_sk_c_id`
(`c_customer_sk`,`c_customer_id`);
ALTER TABLE `customer` ADD INDEX `customer_idx_c_id` (`c_customer_id`);
ALTER TABLE `date_dim` ADD INDEX `date_dim_idx_d_year_d_sk`
(`d_year`,`d_date_sk`);
ALTER TABLE `store` ADD INDEX `store_idx_s_state_s_sk` (`s_state`,`s_store_sk`);
ALTER TABLE `store_returns` ADD INDEX `store_returns_idx_sr_sk_sr_sk_sr_sk`
(`sr_returned_date_sk`,`sr_customer_sk`,`sr_store_sk`);
```

As a result, testing these changes in SF1, the Query 1 went from 37:09 seconds to 32:00 seconds, which originally gave the idea that the optimization could have been successful. However, on further trials with SF10 the overall SF execution time increased instead of decreasing, where the elapsed time went up to 48 minutes against the previous measure of 40 minutes.

For a proper optimization, further trials would be required. A hypothesis from these results is that the indexes may be affecting other queries decreasing their performance, and because of that, even if the execution time from the first query suggests an individual improvement, it would be required to priorize the overall execution.

Conclusion

After analysing the results of the execution time of the 99 queries along the four Scale Factors, the following conclusions were reached:

➔ MySQL has 66.6% compatibility with TPC-DS, considering that 33.3% of the queries needed to be adapted for this benchmark.
➔ Even if the same device is used for all queries in every Scale Factors, it cannot be guaranteed that all queries will be executed under the same condition since the operative system can assign different resources to other backend processes, therefore, some queries may be running with more or less resources than the rest.
➔ Performance along Scale Factors differ for each query tested, giving as a result execution times of less than a millisecond or execution times of several minutes.
➔ Irregularities were found in 11 queries considering their historical behaviors along the Scales Factor. This made them unreliable, and therefore, their results cannot be taken into consideration until further research is made.
➔ Not all the queries presented the same growth rate along the Scales Factors. This may depend on the complexity of the execution plan required for each of them.
➔ Optimization through indexes is not viable.

References

1. MySQL Products. (n.d.). Retrieved September 27, 2022. [Online source]. The link: https://www.mysql.com/products/.
2. MySQL 8.0 Reference Manual :: 1.2.2 The Main Features of MySQL. (n.d.). Retrieved September 27, 2022. [Online source]. The link: https://dev.mysql.com/doc/refman/8.0/en/features.html.
3. DB-Engines Ranking of Relational DBMS. Retrieved October 7, 2022. [Online source]. The link: https://db-engines.com/en/ranking/relational+dbms.
4. TPC Benchmark DS - Standard Specification, Version 3.2.0. 2021. Retrieved October 7, 2022. [Online source]. The link: https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v3.2.0.pdf.
5. Nambiar, Raghu & Poess, Meikel. (2006). The Making of TPC-DS. 1049-1058. Retrieved October 7, 2022. [Online source]. The link: https://www.researchgate.net/publication/221311196_The_Making_of_TPC-DS.
6. Barata, Melyssa & Bernardino, Jorge & Furtado, Pedro. (2015). An overview of decision support benchmarks: TPC-DS, TPC-H and SSB. Advances in Intelligent Systems and Computing. 353. 619-628. 10.1007/978-3-319-16486-1_61. Retrieved October 7, 2022. [Online source]. The link: https://www.researchgate.net/publication/283797188_An_overview_of_decision_support_benchmarks_TPC-DS_TPC-H_and_SSB.

Annex

Execution time Growth

	Growth from SF1 to SF2	Growth from SF2 to SF5	Growth from SF5 to SF10	Total Growth Difference
Q3	0:00:00	0:00:02	0:00:03	0:00:05
Q55	0:00:06	0:00:09	0:00:14	0:00:29
Q42	0:00:04	0:00:11	0:00:21	0:00:36
Q19	0:00:03	0:00:29	0:00:13	0:00:45
Q52	0:00:04	0:00:24	0:00:32	0:01:00
Q40	0:00:21	0:00:26	0:00:22	0:01:09
Q92	0:00:12	0:00:40	0:00:17	0:01:09
Q93	0:00:14	0:00:39	0:00:26	0:01:19
Q84	0:00:20	0:00:08	0:01:09	0:01:37
Q37	0:00:07	0:00:33	0:01:20	0:02:00
Q82	0:00:07	0:00:33	0:01:47	0:02:27
Q56	0:00:26	0:00:48	0:01:15	0:02:29
Q91	0:00:21	0:00:33	0:01:35	0:02:29
Q21	0:00:12	0:00:21	0:02:14	0:02:47
Q49	0:00:16	0:01:42	0:01:36	0:03:34
Q29	0:00:41	0:00:52	0:02:14	0:03:47
Q85	0:00:45	0:01:19	0:01:43	0:03:47
Q20	0:00:32	0:01:17	0:02:03	0:03:52
Q32	0:00:45	0:01:36	0:01:39	0:04:00
Q25	0:00:08	0:01:50	0:02:45	0:04:43
Q80	0:00:19	0:02:02	0:02:50	0:05:11
Q12	0:00:17	0:00:38	0:05:01	0:05:56
Q17	0:00:25	0:04:53	0:01:09	0:06:27
Q86	0:00:59	0:01:47	0:03:53	0:06:39

Q94	0:00:38	0:02:07	0:04:00	0:06:45
Q45	0:00:53	0:01:51	0:04:42	0:07:26
Q50	0:00:23	0:00:29	0:06:56	0:07:48
Q98	0:00:48	0:02:15	0:05:32	0:08:35
Q60	0:01:34	0:01:55	0:06:08	0:09:37
Q90	0:01:31	0:03:44	0:04:40	0:09:55
Q64	0:00:32	0:02:19	0:07:52	0:10:43
Q66	0:03:42	0:01:51	0:06:49	0:12:22
Q43	0:03:25	0:07:15	0:01:53	0:12:33
Q62	0:01:15	0:05:30	0:10:31	0:17:16
Q76	0:01:44	0:05:42	0:10:34	0:18:00
Q96	0:01:12	0:03:29	0:15:41	0:20:22
Q48	0:03:05	0:06:40	0:11:28	0:21:13
Q13	0:01:20	0:05:24	0:15:21	0:22:05
Q44	0:02:24	0:07:45	0:13:19	0:23:28
Q16	0:03:14	0:10:14	0:17:34	0:31:02
Q99	0:03:31	0:10:42	0:18:02	0:32:15
Q75	0:02:06	0:09:05	0:21:26	0:32:37
Q26	0:02:59	0:11:36	0:18:40	0:33:15
Q18	0:08:49	0:06:03	0:18:55	0:33:47
Q68	0:03:07	0:05:01	0:28:04	0:36:12
Q2	0:04:24	0:12:51	0:21:21	0:38:36
Q28	0:04:40	0:12:48	0:21:34	0:39:02
Q34	0:02:28	0:26:45	0:10:53	0:40:06
Q73	0:02:55	0:24:29	0:13:15	0:40:39
Q46	0:13:44	0:12:08	0:15:37	0:41:29
Q33	0:07:16	0:10:36	0:25:24	0:43:16

Q72	0:02:39	0:11:36	0:32:25	0:46:40
Q71	0:03:05	0:14:13	0:29:47	0:47:05
Q41	0:02:11	0:11:23	0:38:28	0:52:02
Q9	0:05:06	0:16:35	0:30:27	0:52:08
Q69	0:04:05	0:12:05	0:43:56	1:00:06
Q57	0:09:39	0:35:58	0:16:40	1:02:17
Q7	0:06:54	0:22:11	0:36:17	1:05:22
Q77	0:07:13	0:19:06	0:41:08	1:07:27
Q81	0:00:59	0:12:10	0:57:24	1:10:33
Q39	0:02:38	0:17:34	0:53:54	1:14:06
Q87	0:07:39	0:15:54	0:51:04	1:14:37
Q59	0:05:14	0:23:47	0:47:42	1:16:43
Q30	0:01:25	0:13:52	1:12:43	1:28:00
Q67	0:01:09	0:28:16	0:59:26	1:28:51
Q79	0:05:16	0:16:32	1:07:10	1:28:58
Q24	0:01:47	0:17:29	1:10:50	1:30:06
Q65	0:06:33	0:33:11	0:54:06	1:33:50
Q70	0:09:02	0:23:03	1:01:49	1:33:54
Q97	0:07:29	0:27:50	1:04:45	1:40:04
Q47	0:07:46	0:35:22	0:58:29	1:41:37
Q38	0:09:20	0:32:56	1:01:16	1:43:32
Q51	0:11:16	0:31:18	1:31:43	2:14:17
Q22	0:01:57	0:16:06	1:59:26	2:17:29
Q95	0:04:08	0:35:44	2:07:28	2:47:20
Q31	0:25:11	1:17:44	2:16:56	3:59:51
Q78	0:24:11	1:14:23	2:21:52	4:00:26
Q88	0:10:20	1:46:31	2:13:22	4:10:13

Q58	0:02:41	0:15:00	4:16:58	4:34:39
Q74	0:20:33	1:43:43	3:31:53	5:36:09
Q54	0:14:02	1:16:46	4:40:11	6:10:59
Q23	0:44:30	3:04:14	8:57:08	12:45:52
Q11	1:25:30	6:13:42	10:18:23	17:57:35
Q4	2:07:15	8:11:29	16:27:09	26:45:53

Irregular Queries

	SF 1	SF 2	SF 5	SF 10	Slowest	Fastest
Q63	0:00:43	0:00:39	0:01:00	0:02:00	0:02:00	0:00:39
Q53	0:00:42	0:00:38	0:01:28	0:02:35	0:02:35	0:00:38
Q83	0:01:09	0:01:21	0:02:59	0:02:14	0:02:59	0:01:09
Q61	0:00:03	0:00:00	0:03:23	0:06:06	0:06:06	0:00:00
Q89	0:01:38	0:01:31	0:04:06	0:08:12	0:08:12	0:01:31
Q15	0:04:35	0:09:14	0:18:15	0:15:37	0:18:15	0:04:35
Q36	0:12:11	0:36:30	0:31:22	0:43:29	0:43:29	0:12:11
Q27	0:16:07	0:47:23	0:41:16	0:53:13	0:53:13	0:16:07
Q8	2:19:08	6:31:03	0:27:36	0:43:09	6:31:03	0:27:36
Q6	1:24:10	2:57:29	12:50:28	1:28:19	12:50:28	1:24:10
Q1	0:37:09	1:24:25	43:57:37	41:32:24	43:57:37	0:37:09

Performance Analysis

	SF 1	SF 2	SF 5	SF 10	Slowest SF	Fastest SF
Q3	0:00:01	0:00:01	0:00:03	0:00:06	0:00:06	0:00:01
Q55	0:00:03	0:00:09	0:00:18	0:00:32	0:00:32	0:00:03
Q42	0:00:06	0:00:10	0:00:21	0:00:42	0:00:42	0:00:06
Q52	0:00:06	0:00:10	0:00:34	0:01:06	0:01:06	0:00:06

Q19	0:00:22	0:00:25	0:00:54	0:01:07	0:01:07	0:00:22
Q92	0:00:06	0:00:18	0:00:58	0:01:15	0:01:15	0:00:06
Q40	0:00:22	0:00:43	0:01:09	0:01:31	0:01:31	0:00:22
Q93	0:00:13	0:00:27	0:01:06	0:01:32	0:01:32	0:00:13
Q63	0:00:43	0:00:39	0:01:00	0:02:00	0:02:00	0:00:39
Q84	0:00:32	0:00:52	0:01:00	0:02:09	0:02:09	0:00:32
Q37	0:00:18	0:00:25	0:00:58	0:02:18	0:02:18	0:00:18
Q53	0:00:42	0:00:38	0:01:28	0:02:35	0:02:35	0:00:38
Q91	0:00:14	0:00:35	0:01:08	0:02:43	0:02:43	0:00:14
Q56	0:00:23	0:00:49	0:01:37	0:02:52	0:02:52	0:00:23
Q82	0:00:25	0:00:32	0:01:05	0:02:52	0:02:52	0:00:25
Q83	0:01:09	0:01:21	0:02:59	0:02:14	0:02:59	0:01:09
Q49	0:00:17	0:00:33	0:02:15	0:03:51	0:03:51	0:00:17
Q21	0:01:25	0:01:37	0:01:58	0:04:12	0:04:12	0:01:25
Q85	0:00:29	0:01:14	0:02:33	0:04:16	0:04:16	0:00:29
Q32	0:00:22	0:01:07	0:02:43	0:04:22	0:04:22	0:00:22
Q29	0:00:42	0:01:23	0:02:15	0:04:29	0:04:29	0:00:42
Q20	0:00:42	0:01:14	0:02:31	0:04:34	0:04:34	0:00:42
Q25	0:01:06	0:01:14	0:03:04	0:05:49	0:05:49	0:01:06
Q61	0:00:03	0:00:00	0:03:23	0:06:06	0:06:06	0:00:00
Q12	0:00:26	0:00:43	0:01:21	0:06:22	0:06:22	0:00:26
Q80	0:01:12	0:01:31	0:03:33	0:06:23	0:06:23	0:01:12
Q17	0:01:09	0:01:34	0:06:27	0:07:36	0:07:36	0:01:09
Q86	0:01:22	0:02:21	0:04:08	0:08:01	0:08:01	0:01:22
Q94	0:01:18	0:01:56	0:04:03	0:08:03	0:08:03	0:01:18
Q50	0:00:23	0:00:46	0:01:15	0:08:11	0:08:11	0:00:23
Q89	0:01:38	0:01:31	0:04:06	0:08:12	0:08:12	0:01:31

Q45	0:01:39	0:02:32	0:04:23	0:09:05	0:09:05	0:01:39
Q98	0:01:07	0:01:55	0:04:10	0:09:42	0:09:42	0:01:07
Q90	0:01:01	0:02:32	0:06:16	0:10:56	0:10:56	0:01:01
Q60	0:01:48	0:03:22	0:05:17	0:11:25	0:11:25	0:01:48
Q64	0:01:31	0:02:03	0:04:22	0:12:14	0:12:14	0:01:31
Q66	0:01:56	0:05:38	0:07:29	0:14:18	0:14:18	0:01:56
Q43	0:03:53	0:07:18	0:14:33	0:16:26	0:16:26	0:03:53
Q15	0:04:35	0:09:14	0:18:15	0:15:37	0:18:15	0:04:35
Q62	0:02:32	0:03:47	0:09:17	0:19:48	0:19:48	0:02:32
Q76	0:02:27	0:04:11	0:09:53	0:20:27	0:20:27	0:02:27
Q96	0:01:20	0:02:32	0:06:01	0:21:42	0:21:42	0:01:20
Q48	0:02:20	0:05:25	0:12:05	0:23:33	0:23:33	0:02:20
Q13	0:02:09	0:03:29	0:08:53	0:24:14	0:24:14	0:02:09
Q44	0:03:21	0:05:45	0:13:30	0:26:49	0:26:49	0:03:21
Q16	0:03:07	0:06:21	0:16:35	0:34:09	0:34:09	0:03:07
Q99	0:03:24	0:06:55	0:17:37	0:35:39	0:35:39	0:03:24
Q75	0:03:24	0:05:30	0:14:35	0:36:01	0:36:01	0:03:24
Q18	0:04:30	0:13:19	0:19:22	0:38:17	0:38:17	0:04:30
Q26	0:05:22	0:08:21	0:19:57	0:38:37	0:38:37	0:05:22
Q68	0:04:12	0:07:19	0:12:20	0:40:24	0:40:24	0:04:12
Q2	0:04:40	0:09:04	0:21:55	0:43:16	0:43:16	0:04:40
Q36	0:12:11	0:36:30	0:31:22	0:43:29	0:43:29	0:12:11
Q28	0:04:36	0:09:16	0:22:04	0:43:38	0:43:38	0:04:36
Q34	0:03:33	0:06:01	0:32:46	0:43:39	0:43:39	0:03:33
Q73	0:03:18	0:06:13	0:30:42	0:43:57	0:43:57	0:03:18
Q46	0:04:32	0:18:16	0:30:24	0:46:01	0:46:01	0:04:32
Q33	0:06:13	0:13:29	0:24:05	0:49:29	0:49:29	0:06:13

Q72	0:03:33	0:06:12	0:17:48	0:50:13	0:50:13	0:03:33
Q71	0:04:20	0:07:25	0:21:38	0:51:25	0:51:25	0:04:20
Q27	0:16:07	0:47:23	0:41:16	0:53:13	0:53:13	0:16:07
Q41	0:02:16	0:04:27	0:15:50	0:54:18	0:54:18	0:02:16
Q9	0:06:20	0:11:26	0:28:01	0:58:28	0:58:28	0:06:20
Q69	0:04:21	0:08:26	0:20:31	1:04:27	1:04:27	0:04:21
Q57	0:08:31	0:18:10	0:54:08	1:10:48	1:10:48	0:08:31
Q81	0:01:03	0:02:02	0:14:12	1:11:36	1:11:36	0:01:03
Q7	0:07:58	0:14:52	0:37:03	1:13:20	1:13:20	0:07:58
Q77	0:06:03	0:13:16	0:32:22	1:13:30	1:13:30	0:06:03
Q39	0:04:59	0:07:37	0:25:11	1:19:05	1:19:05	0:04:59
Q87	0:05:27	0:13:06	0:29:00	1:20:04	1:20:04	0:05:27
Q59	0:06:21	0:11:35	0:35:22	1:23:04	1:23:04	0:06:21
Q30	0:01:17	0:02:42	0:16:34	1:29:17	1:29:17	0:01:17
Q79	0:04:15	0:09:31	0:26:03	1:33:13	1:33:13	0:04:15
Q24	0:07:01	0:08:48	0:26:17	1:37:07	1:37:07	0:07:01
Q65	0:05:48	0:12:21	0:45:32	1:39:38	1:39:38	0:05:48
Q70	0:06:08	0:15:10	0:38:13	1:40:02	1:40:02	0:06:08
Q67	0:14:02	0:15:11	0:43:27	1:42:53	1:42:53	0:14:02
Q38	0:04:08	0:13:28	0:46:24	1:47:40	1:47:40	0:04:08
Q97	0:08:38	0:16:07	0:43:57	1:48:42	1:48:42	0:08:38
Q47	0:20:01	0:27:47	1:03:09	2:01:38	2:01:38	0:20:01
Q22	0:05:29	0:07:26	0:23:32	2:22:58	2:22:58	0:05:29
Q51	0:09:14	0:20:30	0:51:48	2:23:31	2:23:31	0:09:14
Q95	0:05:17	0:09:25	0:45:09	2:52:37	2:52:37	0:05:17
Q88	0:11:13	0:21:33	2:08:04	4:21:26	4:21:26	0:11:13
Q78	0:23:08	0:47:19	2:01:42	4:23:34	4:23:34	0:23:08

Q31	0:24:28	0:49:39	2:07:23	4:24:19	4:24:19	0:24:28
Q58	0:05:39	0:08:20	0:23:20	4:40:18	4:40:18	0:05:39
Q74	0:16:11	0:36:44	2:20:27	5:52:20	5:52:20	0:16:11
Q54	0:08:18	0:22:20	1:39:06	6:19:17	6:19:17	0:08:18
Q8	2:19:08	6:31:03	0:27:36	0:43:09	6:31:03	0:27:36
Q6	1:24:10	2:57:29	12:50:28	1:28:19	12:50:28	1:24:10
Q23	0:32:51	1:17:21	4:21:35	13:18:43	13:18:43	0:32:51
Q11	1:10:30	2:36:00	8:49:42	19:08:05	19:08:05	1:10:30
Q4	1:36:24	3:43:39	11:55:08	28:22:17	28:22:17	1:36:24
Q1	0:37:09	1:24:25	43:57:37	41:32:24	43:57:37	0:37:09