Zeeshan Anwar (160931)
Zoya Nadeem (161000)

Materialized Views vs Non-Materialized Views in Contemporary Database Management Systems

Bachelor of Science in Computer Science

Supervisor: Shoaib Malik

Co-Supervisor: Dr Tauseef

Department of Computer Science
Air University, Islamabad

February 2020

C e r t i f i c a t e

We accept the work contained in the report titled “MATERIALIZED VIEWS VS NON-MATERIALIZED VIEWS IN CONTEMPORARY DATABASES”, written by Mr. Zeeshan Anwar AND Miss Zoya Nadeem as a confirmation to the required standard for the partial fulfillment of the degree of Bachelor of Science in Computer Science.

Approved by . . . :

Supervisor:

____

Internal Examiner:

____

External Examiner:

____

Project Coordinator:

____

Head of the Department:

____

June, 2020

Abstract

As the world is getting more and more modernized, data in the previous form of papers and documents are now being converted into digital form. Day by day the data is increasing in size. As the data size increases the retrieval of data is getting more and more complex. People are getting more agitated as it takes a lot of time to retrieve that data. To overcome these problems views were generated in databases. Views are defined as a virtual table. It is basically a result of a stored query on the data.

As the data started growing larger and larger. Suddenly views were no longer adaptable in large data warehouses. To tackle this problem Materialized Views were introduced. Materialized views are also a database object which contains the result of a query. Materialized views are better than a non-materialized views as its approach is rather different. The query result is cached as a concrete table that may be updated from the original base tables from time to time.

In this report we will talk about Materialized views and Non-Materialized views in contemporary databases to determine which approach is better for the same dataset. We will present a method in which query response time are shown in different form of visualizations as to get a better understanding.

We will perform multiple experiments in different database management systems to determine which system is more efficient for the same dataset and the same set of queries. We will compare materialized and non-materialized views for the same query and determine which query responses faster than the other.

Our results showed that view materialization is almost always faster than non-materialized views. During our experiments we found out that materializing all views was not optimal. Since view materialization depends on a database administrator, it would be his job to analyze the views and pick the optimal number of views for materialization.

Experimenting with different database management systems helped us realize that indeed are very different from each other. Their response time varies from each other. As these experiments were carried out on a single system, we can tell which database management system gave us the best and most optimal results. But these database management systems rely very much on the operating system they are being ran on and we cannot generally say which of these database Management systems is the best.

Since the generation and selection of these views were done manually. More work can be done in this field. Such as automating the process of generating and selecting these views using machine learning or artificial intelligence. Since the concept of view materialization is relatively newer than non-materialized views, it should be practiced globally as we have proved in our report that view materialization works better than non-materialized views.

Acknowledgments

We would first of all like to thank Almighty Allah for giving us the strength and opportunity to achieve the objective that we had set. and especially during the course of our Final Year Project. Only through His kind will, it was possible.

We would like to thank our parents without whom it would be impossible to be here. The prayers, sacrifices and encouragement of our parents and grandparents throughout these years and both in our whole lives, cannot be acknowledged in a few words.

Finally, we would like to acknowledge the support that has been given to us by our supervisor Sir Shoaib Malik for his patient and continuous guidance. He was always available to help us get out of difficult situations. We would like to show our gratitude to the faculty members of our prestigious Computer Science department for their help when we required it.

Muhammad Zeeshan
Zoya Nadeem
Islamabad, Pakistan

June, 2020

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

SQL	Structured Query Language.
GUI	Graphical User Interface.
TPC-DS	Transaction Processing Council-Decision Support Benchmark
RDBMS	Relational Database Management System
DBMS	Database Management System

Chapter 1

Introduction

Our FYP focusses on views in contemporary database management systems. This is a Research based project. We attempt to provide a comprehensive overview of the hurdles and possible solutions of this project.

To understand views, we must first talk about databases. A database is an organized collection of data, generally stored and accessed electronically from a computer system. For the retrieval of the data stored in those databases, many languages were introduced. The language we focused on is STRUCTRED QUERY LANGUAGE (SQL). As the data grew larger and larger it became more difficult to retrieve that data efficiently and optimally. The following is an introductory chapter detailing project background, problem description, objectives, project scope and the life cycle of our project.

1.1 Project Background/Overview:

A data warehouse is a very large database system that collects, summarizes and stores data from multiple remote and heterogeneous information sources. The decision-making queries are complex in nature and take a large amount of time when they are run against a large data warehouse. To reduce the time and cost of the queries, views are generated.

Traditionally, views are "virtual". The database system stores their definition queries but not their contents. Virtual views are often used to control access and provide alternative interfaces to base tables. They also support logical data independence: when the base table schema changes, views can be redefined to use the new schema, so application queries written against these views will continue to function. Over the years, however, the concept and practice of materialized views have steadily gained importance.

In a database, a view is the result set of stored queries on the data. A view does not form part of the physical schema. As a resultant set, it is a virtual table computed dynamically from data in the databases when access to the view is requested.

As the data size increases the simple views generated are not enough. For that purpose, a new method was first introduced by Oracle. That method is known as materializing the views. Materialized views are defined as the database objects that contains the result of the query. The process of setting up a materialized view is often called view materialization. The query result is cached as a concrete table that may be updated from original base tables from time to time. Meaning the materialized views are physically stored in databases.

Materialized views are physical structures in databases and are smaller in size. These views are usually created by database administrator which means the quality of views is dependent on the database administrator's experience and intuition.

We materialize a view by storing its contents. Once materialized, a view can facilitate queries that use it (or can be rewritten to use it), when the base tables are expensive or even unavailable for access.

1.2 Problem Description:

Queries posed against a large data warehouse, are typically analytic, intricate and recurring in nature. Such queries contain a lot of join operations over large volumes of records. To minimize the cost of these queries views are introduced into the system. Materialized views are an updated version of these database views. The main objective of using views is to minimize the cost of a query.

Since a materialized view is an advanced form a view, it is extremely fast retrieval of aggregate data, since it is precomputed and stored, at the expense of insert/update/delete. The database will keep the Materialized View in sync with the real data.

1.3 Project Objectives:

Following are the objectives and goals of this project:

- Generating materialized and non-materialized views in contemporary database management systems.
- Analyzing the query runtimes against materialized and non-materialized views.

1.4 Project Scope:

This project aims to analyze the runtime of the queries posed against the same dataset. Firstly, we selected 4 database management systems (SQL SERVER, POSTGRES, SQL ANYWHERE, ORACLE). After selecting these database management systems, data is loaded into these individually.

After loading the dataset, the management systems were connected to a programming language(python) via a connection string. The materialized views and non-materialized views were generated programmatically for each of these database management systems.

These views were executed programmatically using select statements and the run time (query response time) was calculated by using the inbuilt functions of the programming language. These results will then be displayed graphically using bar graphs for a better understanding.

1.5 Project Life Cycle:

This project was divided into multiple parts. The first step towards achieving our goal was to understand the problem. For this purpose, a research was conducted in which various research papers were studied which were related to view materialization. This research gave us an insight to what we had to achieve. To perform our experiments, we had to gather different database management systems which supported view materialization. Our experiments consisted of various steps, one of which was to gather a dataset. We used a decision support benchmark for experiments which was easily accessible to us. We conducted our experiment of generating materialized and non-materialized views and calculated their response time. The results of these experiments are explained comprehensively in subsequent chapters of the report.

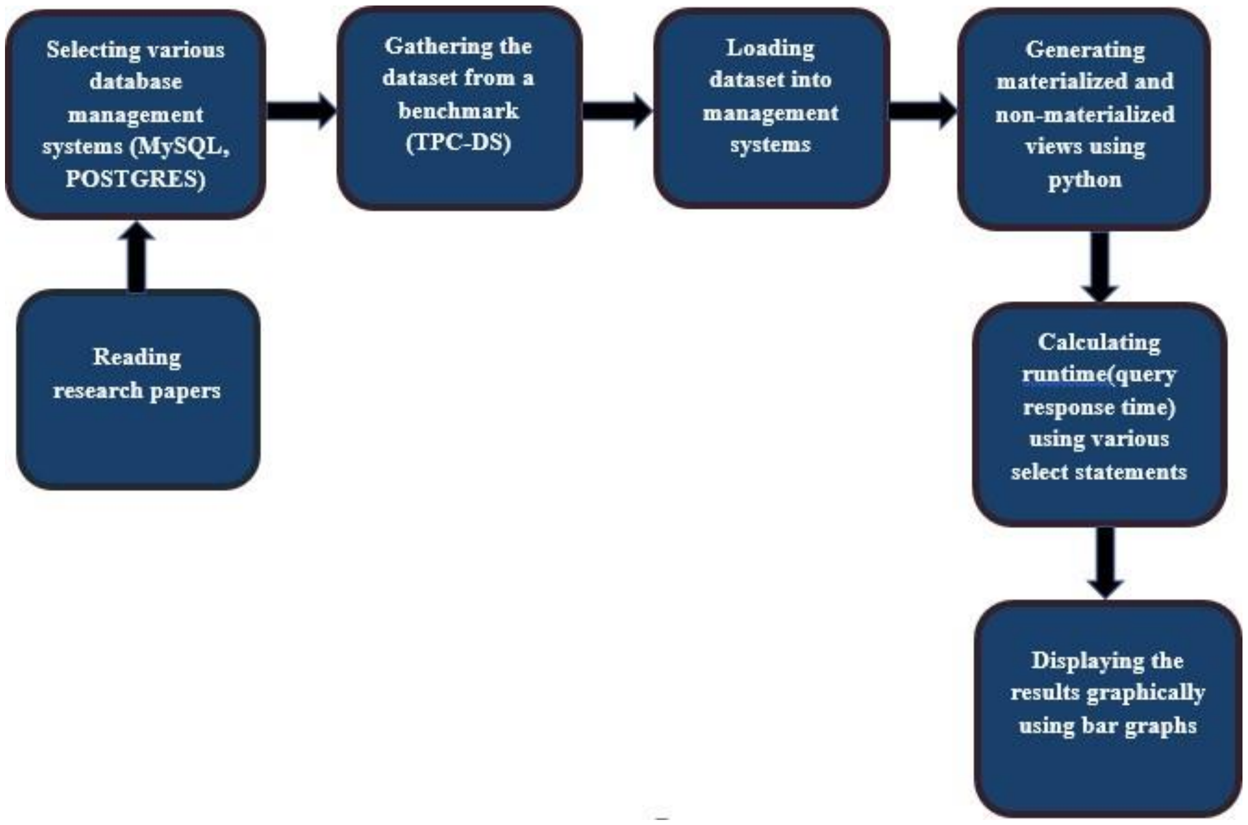

Selecting various database management systems (MySQL, POSTGRES)
Gathering the dataset from a benchmark (TPC-DS)
Loading dataset into management systems
Generating materialized and non-materialized views using python
Reading research papers
Calculating runtime(query response time) using various select statements
Displaying the results graphically using bar graphs

Figure 1.1: Final Year Project Lifecycle

Conclusion of Chapter:

Since materialized view is an advance form of view, it is extremely fast retrieval of aggregate data. Materialized views are an updated version of these database views. The decision-making queries are complex in nature and take a large amount of time when they are run against a large data warehouse. Materialized views can provide massive improvements in query processing time, especially for aggregation queries over large tables. To reduce the time and cost of the queries views are generated. It is concluded that, the main objective of using views is to minimize the cost of a query.

Chapter 2

Literature Review

Since this is a Research based project, for this purpose, we have studied various research papers that are related to materialized and non-materialized views. This is a 2nd chapter detailing review of research papers and their survey.

2.1 Review of Exiting Research Papers:

2.1.1 Optimizing Queries Using Materialized Views: A Practical, Scalable Solution (Jonathan Goldstein & Larson, 2001)

This paper talks about how materialized views can provide massive improvements in query processing time, especially for aggregated queries over larger tables. This paper performed experiments on TPC-R benchmark. Experimental results based on an implementation in Microsoft SQL Server showed excellent results and very fast. With 1000 views in the system, average optimization time remained as low as 0.15 seconds per query.

2.1.2 Optimizing Queries with Materialized Views (Sura jit Chaudhurix, Ravi Krishnamurthyy & Spyros Potamianosz, 2001)

This paper talks about of maintaining materialized views. Since all views cannot be materialized as it will take enormous space. This paper analyses the optimization problem and provides a comprehensive and efficient solution which reduces the cost of queries.

2.1.3 Materialized Views (Rada Chirkova & Jun Yang, 2012)

Rada Chirkova and Jun Yang in their paper “Materialized Views” talk about how In SQL settings the materialized views are considered a mature technology implemented by commercial database systems and tightly integrated into query optimization. They have covered 3 fundamental problems: Maintaining views efficiently when base tables change, using materialize views to improve performance and availability, selecting which views to materialize.

2.2 Survey Table:

Papers	**Dataset**	**System**
2.1.1	TPC-R (benchmark)	SQL
2.1.2	Self-made dataset employee relation Emp (name, dno, sal, age) and a department relation Dept (dno, size, loc)	SQL
2.1.3	Self-made dataset Retailer pos (itemID, storeID, date) stores (storeID, city, region) items (itemID, name, cost)	SQL

Table 1.1: Survey

Conclusion of Chapter:

To conclude this chapter a lot of work has been done using materialized and non-materialized views. Since materialized views have been introduced a lot of experiments have been done and their accuracy vary based on their systems. We will be performing these experiments on the same system but different database management systems to show which management system stands out. Which management system works better with materialized and non-materialized views.

Chapter 3

Requirement Specifications

We have studied numerous research papers and documentations in order to find the best research till the time. This chapter, also includes the description of our proposed system and data collection and explains how it addresses and attempts to solve the letdowns and limitations of the existing ones. Furthermore, these research papers extensively gave us an idea as to how we should approach our problem and how can we use the best tools in order to solve it.

3.1 Existing System:

Traditionally, optimizations of database application query execution are performed manually by a database administrator. The administrator does so, by creating and implementing views onto the database. Materialized views are queries, whose results are stored and maintained in order to facilitate access to data in their underlying base tables.

Out of the research performed by me and my group partner, Materialized Views (Rada Chirkova & Jun Yang, 2011) gave us the best idea how we can approach our problem. In this paper they performed their experiments using SQL. The research covered three main factors "maintaining materialized views efficiently when the base table changes." "using materialized views effectively to improve performance." "selecting which views to materialize".

Following paper gives us an idea how the work has been done and how In SQL settings the materialized views are considered a mature technology implemented by commercial database systems and tightly integrated into query optimization. Our goal of this monograph is to provide an accessible introduction and reference to this topic, explain its core ideas and highlight its recent developments. Since this research was done on a smaller scale the decision of which views to materialize were made by a database administrator. These experiments were performed in Microsoft SQL server.

3.2 Proposed specifications:

The proposed system is to implement materialized vs non-materialized views in contemporary database management system. Since in our research base paper they only used Microsoft SQL Server. We expanded our experiment to more database management systems. For this project the materialized views and non-materialized views were generated programmatically for each of these database management systems.
In order to discover the best database management system for materialized and non-materialized views. Our research will consist of a vast difference between these management systems and we will be able to distinguish the best system amongst them. Since all of these experiments will be conducted on a single operating system. The results will be unbiased.

3.3 Data Collection:

The dataset that we will be using for our experiment was gathered from tpc.org. The name of the dataset is **TPC-DS.**

3.3.1 TPC-DS:

TPC-DS is a decision support benchmark. It models several generally applicable aspects of a decision support system, including queries and data maintenance. TPC-DS is one of the first benchmarks of the TPC benchmark which supports the idea of views.

The TPC Benchmark DS (TPC-DS) is a decision support benchmark that models several generally applicable aspects of a decision support system, including queries and data maintenance. The benchmark provides a representative evaluation of performance as a general purpose, decision support system. A benchmark result measures query response time in single user mode, query throughput in multi user mode and data maintenance performance for a given hardware, operating system, and data processing system configuration under a controlled, complex, multi-user decision support workload. The purpose of TPC benchmarks is to provide relevant, objective performance data to industry users.

Schema Type	Multiple Snowflake
Number of Tables	24
Number of Columns (min)	3
Number of Columns (max)	34
Number of Columns (avg)	18
Number of Foreign Keys	104

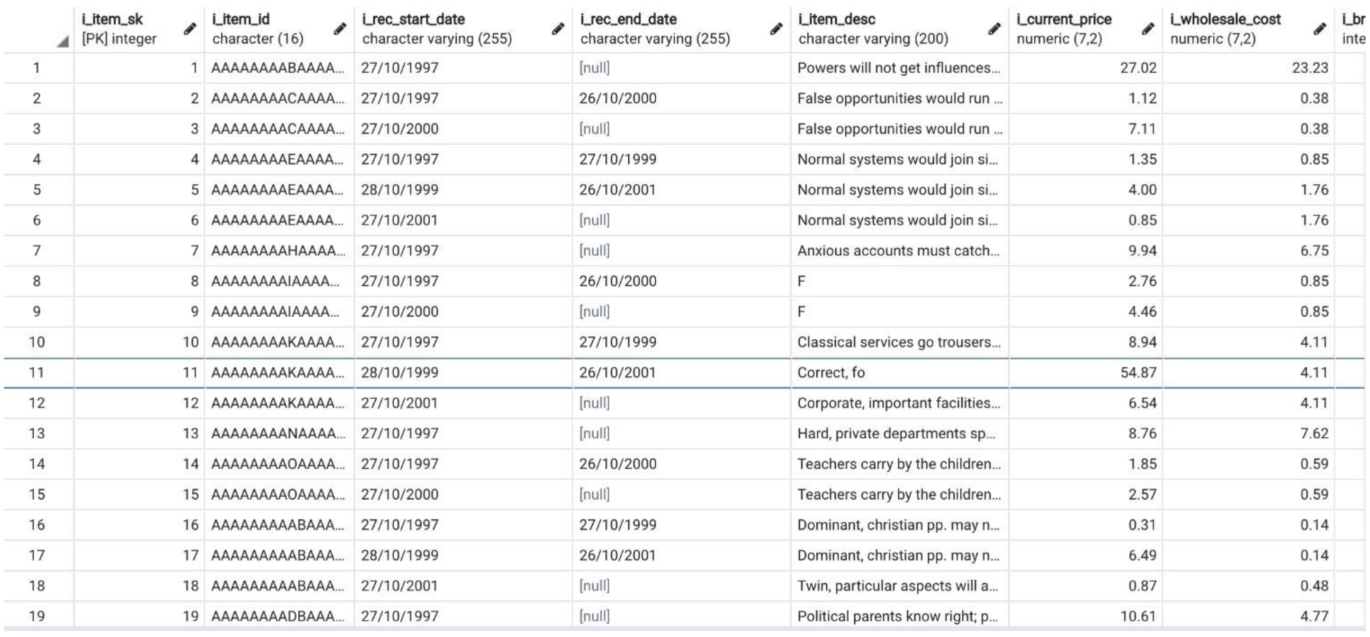
Table 1.2: TPC-DS

3.3.2 Data Format:

- The data was gathered from https://github.com/databricks/tpcds-kit.
- The data was generated in .dat files.
- The data generated was then converted into csv to import it into the database management systems.
- The scale factor used for this dataset was “SF=1” which is about 1GB of data.

3.3.3 Data Sample:

A data sample of the table item.



	i_item_sk [PK] integer	i_item_id character (16)	i_rec_start_date character varying (255)	i_rec_end_date character varying (255)	i_item_desc character varying (200)	i_current_price numeric (7,2)	i_wholesale_cost numeric (7,2)	i_br inte
1	1	AAAAAAAABAAAA...	27/10/1997	[null]	Powers will not get influences...	27.02	23.23	
2	2	AAAAAAAACAAAA...	27/10/1997	26/10/2000	False opportunities would run ...	1.12	0.38	
3	3	AAAAAAAACAAAA...	27/10/2000	[null]	False opportunities would run ...	7.11	0.38	
4	4	AAAAAAAAEAAAA...	27/10/1997	27/10/1999	Normal systems would join si...	1.35	0.85	
5	5	AAAAAAAAEAAAA...	28/10/1999	26/10/2001	Normal systems would join si...	4.00	1.76	
6	6	AAAAAAAAEAAAA...	27/10/2001	[null]	Normal systems would join si...	0.85	1.76	
7	7	AAAAAAAAHAAAA...	27/10/1997	[null]	Anxious accounts must catch...	9.94	6.75	
8	8	AAAAAAAAIAAAA...	27/10/1997	26/10/2000	F	2.76	0.85	
9	9	AAAAAAAAIAAAA...	27/10/2000	[null]	F	4.46	0.85	
10	10	AAAAAAAAKAAAA...	27/10/1997	27/10/1999	Classical services go trousers...	8.94	4.11	
11	11	AAAAAAAAKAAAA...	28/10/1999	26/10/2001	Correct, fo	54.87	4.11	
12	12	AAAAAAAAKAAAA...	27/10/2001	[null]	Corporate, important facilities...	6.54	4.11	
13	13	AAAAAAAANAAAA...	27/10/1997	[null]	Hard, private departments sp...	8.76	7.62	
14	14	AAAAAAAAOAAAA...	27/10/1997	26/10/2000	Teachers carry by the children...	1.85	0.59	
15	15	AAAAAAAAOAAAA...	27/10/2000	[null]	Teachers carry by the children...	2.57	0.59	
16	16	AAAAAAAAABAAA...	27/10/1997	27/10/1999	Dominant, christian pp. may n...	0.31	0.14	
17	17	AAAAAAAAABAAA...	28/10/1999	26/10/2001	Dominant, christian pp. may n...	6.49	0.14	
18	18	AAAAAAAAABAAA...	27/10/2001	[null]	Twin, particular aspects will a...	0.87	0.48	
19	19	AAAAAAAADBAAA...	27/10/1997	[null]	Political parents know right; p...	10.61	4.77	

Table 1.3: Item Table from Dataset

3.4 Hypothesis:

The data was obtained in data format. Which we converted into csv files for easy import into the database management system. The data was then loaded into the database management system using INSERT queries. After that the database were connected to python programming language via a connection string. Materialized and non-materialized views were programmatically generated into the database.

Conclusion of Chapter:

In this chapter we have explained our proposed system and how it overcomes the limitations and works of existing system. For testing purposes, hypothesis is done. The database is connected to python programming language via a connection string. The dataset used for testing purpose is the TPC-DS dataset, and the query workload is also provided by the TPC Benchmark.

Chapter 4

Design

This chapter explains and describes the design of our proposed system. Since our project is Research based hence, no graphical user interface was designed. The architecture of the proposed system is explained in detail.

4.1 Research Architecture:

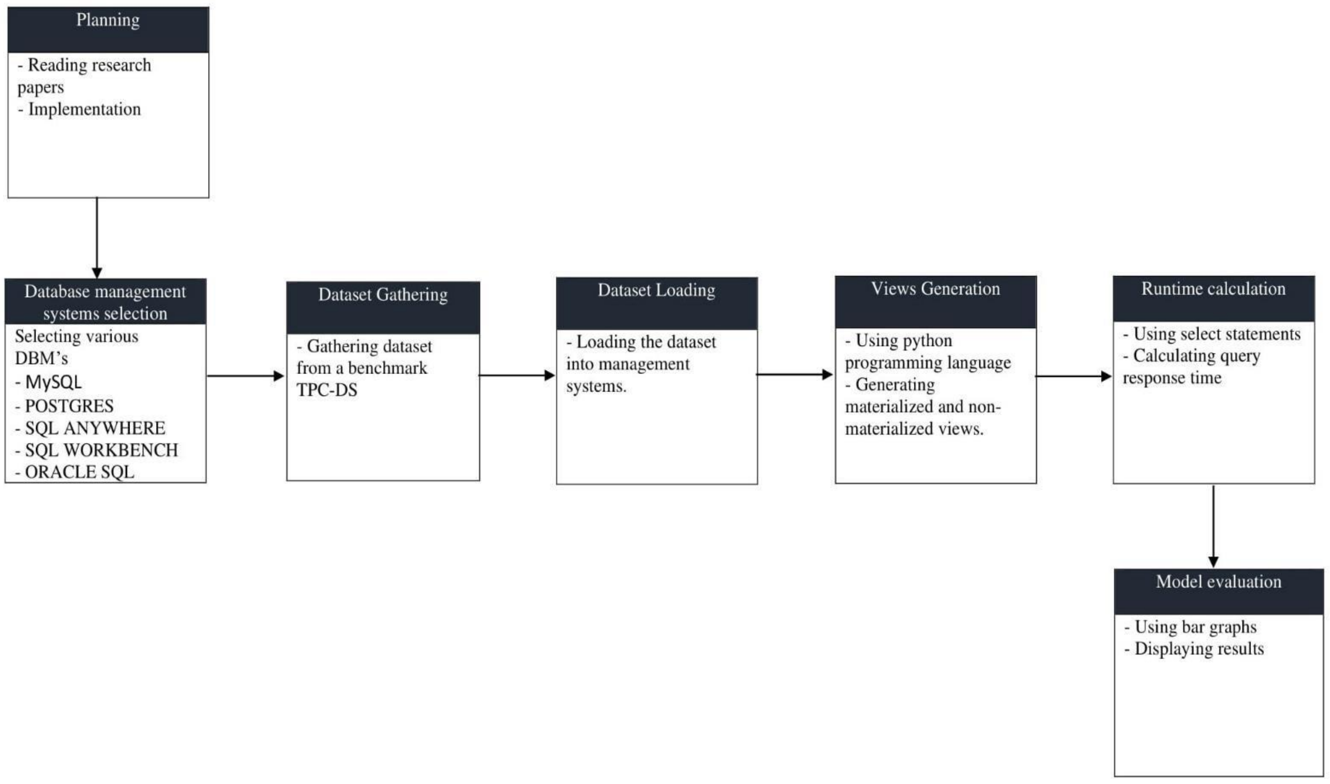

Planning
- Reading research papers
- Implementation
Database management systems selection
Selecting various DBM's
- MySQL
- POSTGRES
- SQL ANYWHERE
- SQL WORKBENCH
- ORACLE SQL
Dataset Gathering
- Gathering dataset from a benchmark TPC-DS
Dataset Loading
- Loading the dataset into management systems.
Views Generation
- Using python programming language
- Generating materialized and non-materialized views.
Runtime calculation
- Using select statements
- Calculating query response time
Model evaluation
- Using bar graphs
- Displaying results

Figure 1.2: Research Architecture

4.2 Research Constraint:

Not many constraints occurred during the experimentation of our work. The constraints that occurred were hardware related. Our hardware was hardly capable of supporting multiple database management systems Since we only had one system, we had to perform our experiments in such a way that each of the database management system used power equally and optimally.

Ideally the best practice to conduct an experiment is to run it on different systems and gather the results. Since this project was done on a lower level and only one singular system was available, we cannot determine which of the database management system would be the best generally as each of these database management systems would work differently on a different system with higher specifications.

4.3 Database Design:

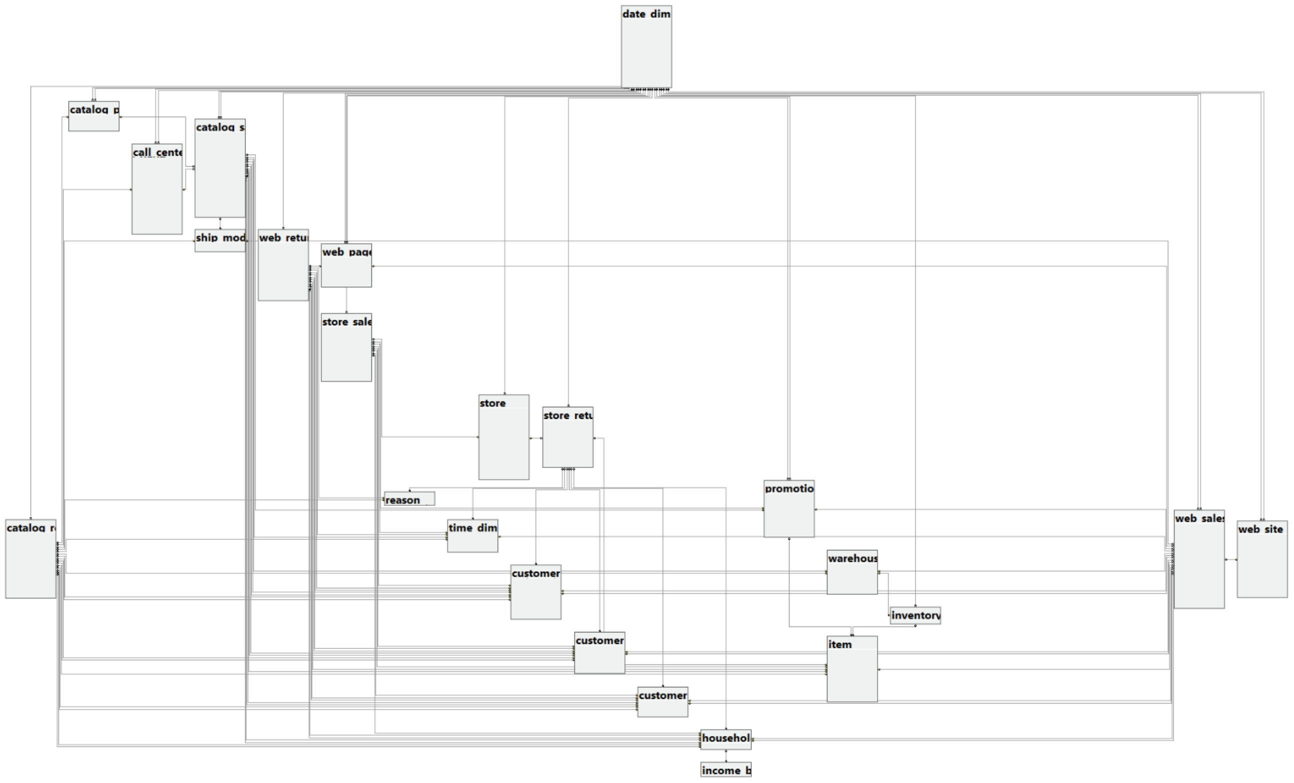


Figure 1.3: Database Design

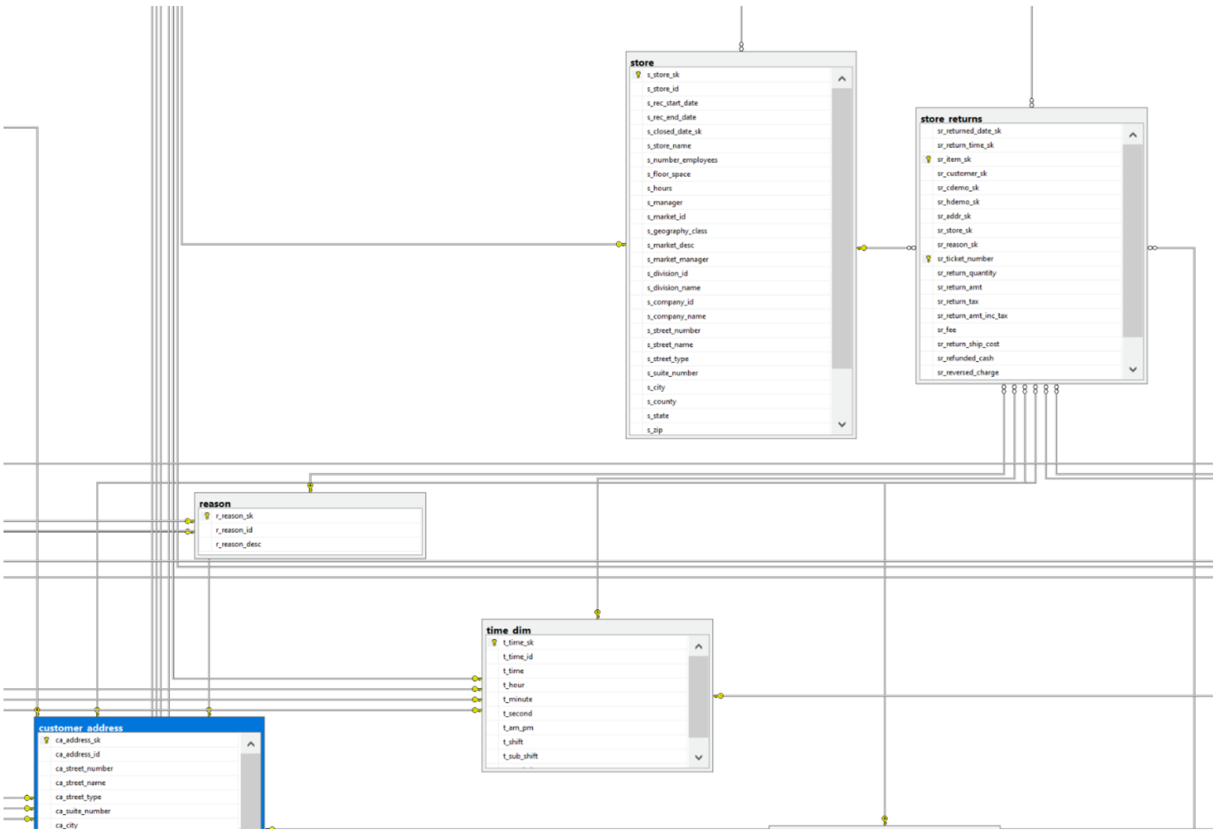


Figure 1.4: A Closer Look into the ER-Diagram

Conclusion of Chapter:

This describes the architecture of the proposed system in detail. Since this is a Research based project hence, no GUI was designed. Furthermore, we have design constraints explained. Not many constraints occurred during the experimentation of our work. The constraints that occurred were hardware related. Our hardware was hardly capable of supporting multiple database management systems. Since we only had one system, we had to perform our experiments in such a way that each of the database management system used power equally and optimally.

Chapter 5

System Implementation

The following chapter explains the process and development study that we applied in project. This chapter also explains the system architecture, Libraries and data collections. It lists the sorts of visuals that we used to represent our data and our experiments performed on that data. Moreover, the results and description of every version of the proposed system are presented.

5.1 System Architecture:

Our system architecture consists of three layers. We used relational database to store our data obtained from the data source. We used four database management systems. Microsoft SQL Server Management Studio, PostgreSQL, SQL Anywhere, SQL Workbench, Oracle SQL Developer. We had to carefully choose those database management systems which support the generation of materialized views.

We experimented on that data using management studios and python programming language.
Python is an interpreted, high-level, general-purpose programming language. Structured Query Language or SQL is a domain specific language, used in programming and designed for managing and interacting with a relational database management system. We performed our experiments on different database management systems testing their query response time under the same environment.

We performed our experiments on this data with the help of python programming language.
Python was connected to these database management systems via a connection string. Our experiments consisted of generating materialized and non-materialized views. Which were then called in their respective queries. The materialized and non-materialized were generated programmatically into the database. We represented our results in visuals in the form of tables and graphs.

5.1.1 Three-Tier Architecture:

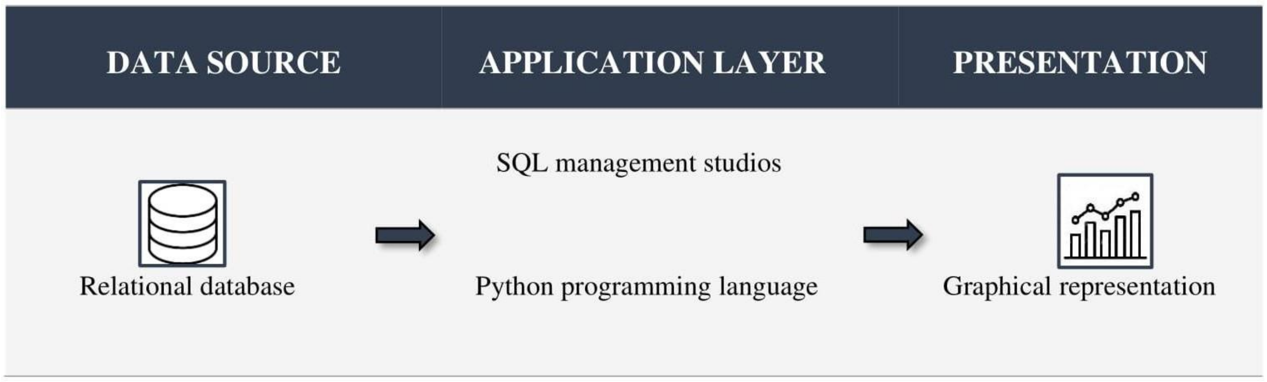


Figure 1.5: Three-Tier Architecture

Database Layer/Data source:

This layer Consists of four databases, populated with data obtained from the TPC-DS Benchmark.

Application Layer/Business Layer:

Our proposed system operates in Application layer. The system interacts and performs experiments using the database operations. It then outputs the results of the experiments onto the monitor screen. The user executes the experiment program, using the system terminal. The source program is executed, and the results are displayed onto the display screen within the terminal window.

5.1.2 Language:

Programming language is defined as a formal language, consisting of a set of instructions. These instructions are used to define a computer program that when executed, perform a specific task. For the development of our proposed system, **Python** programming language and **SQL** were used.

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum. Our proposed systems source code was developed and implemented using this programming language. Structured Query Language or SQL is a domain specific language, used in programming and designed for managing and interacting with a relational database management system.

SQL was used within the database layer and the business layer. In the database layer, it was used to create and populate the database, and in the business layer it was used to preform database queries, required to test the effectiveness of the system.

5.1.3 Libraries:

In programming language, a library is a collection of precompiled routines that a program can use. These routines sometimes called modules, are stored in object format. Libraries are particularly useful for storing frequently used routines, because you do not need to explicitly link them to every program that uses them.

We used the following python libraries for our project:

- **Psycopg2**
- **Pyodbc**
- **cx_Oracle**
- **mysql. Connector**

5.2 Experiments Performed:

We performed our experiments on different database management systems testing their query response time under the same environment. The query response time was then calculated individually of each database management system and compared with each other. The results were then converted into a visual form to get a better understanding.

5.2.1 Data Collection:

We used TPC-DS a decision support benchmark for our dataset. The tools for generating the dataset were downloaded from https://github.com/databricks/tpcds-kit.

The dataset was generated from these tools using command **./dsdgen -scale 1** scale being the size of the data. We used SF=1 to generate 1 GB of data. The data was generated in .dat format. Which was then converted into csv for easy import into the databases by using Microsoft Excel.

The data was the loaded into different database management systems using their respective data importing commands.

5.2.2 Database Management Systems:

A database management system (DBMS) is a software package designed to define, manipulate, retrieve and manage data in a database. A DBMS generally manipulates the data itself, the data format, field names, record structure and file structure. It also defines rules to validate and manipulate this data.

We are using variety of database management systems which include

- Microsoft SQL Server Management Studio.
- PostgreSQL.
- SQL Anywhere.
- SQL Workbench.
- Oracle SQL Developer

We had to carefully choose those database management systems which support the generation of materialized views. As our whole work revolves around materialized and non-materialized views.

5.2.3 Experiments Performed on Data:

We performed our experiments on this data with the help of python programming language. Python was connected to these database management systems via a connection string.

Our experiments consisted of generating materialized and non-materialized views. Which were then called in their respective queries. The materialized and non-materialized were generated programmatically into the database.

Every database management system has their own way of generating a materialized view. We have generated materialized view in Microsoft SQL Server and PostgreSQL. **PostgreSQL** has a better concurrency management system. It handles very well the case where multiple processes can access and modify shared data at the same time. On the other hand, **SQL Server** has underdeveloped concurrency and you can easily get various locked, blocked, and deadlocked reports in the log. Following is an example how materialized views are generated in both.

Microsoft SQL Server:
Microsoft SQL Server is a relational database management system developed by Microsoft. As a database server, it is a software product with the primary function of storing and retrieving data

as requested by other software applications, which may run either on the same computer or on another computer across a network.

```
Create View Myview
WITH
SchemaBinding
AS SELECT COL1, SUM(COL2)
From <table-
name> GROUP
BY COL1
```

PostgreSQL:

PostgreSQL, also known as Postgres, is a free and open-source relational database management system emphasizing extensibility and SQL compliance. It was originally named POSTGRES, referring to its origins as a successor to the Ingres database developed at the University of California, Berkeley.

```
Create MATERIALIZED VIEW
MyView AS SELECT COL1,COL2
FROM <table-
name> GROUP BY
COL2
```

These views were then called into various SQL statements. These SQL statements were specifically generated for these views.

5.2.4 Calculating Query Response Time:

The query response time for different database management systems was calculated in python using inbuilt libraries. The time was calculated for the SQL statements which were calling a view. Each statement was run 9 times and the median value was taken as the final execution time. Both materialized and non-materialized views were tested with those SQL statements and the response time was noted down. These response times were then converted into tabular and graphical representation.

5.3 Graphs:

Following is the graphical representation of the results we achieved during our experiments.

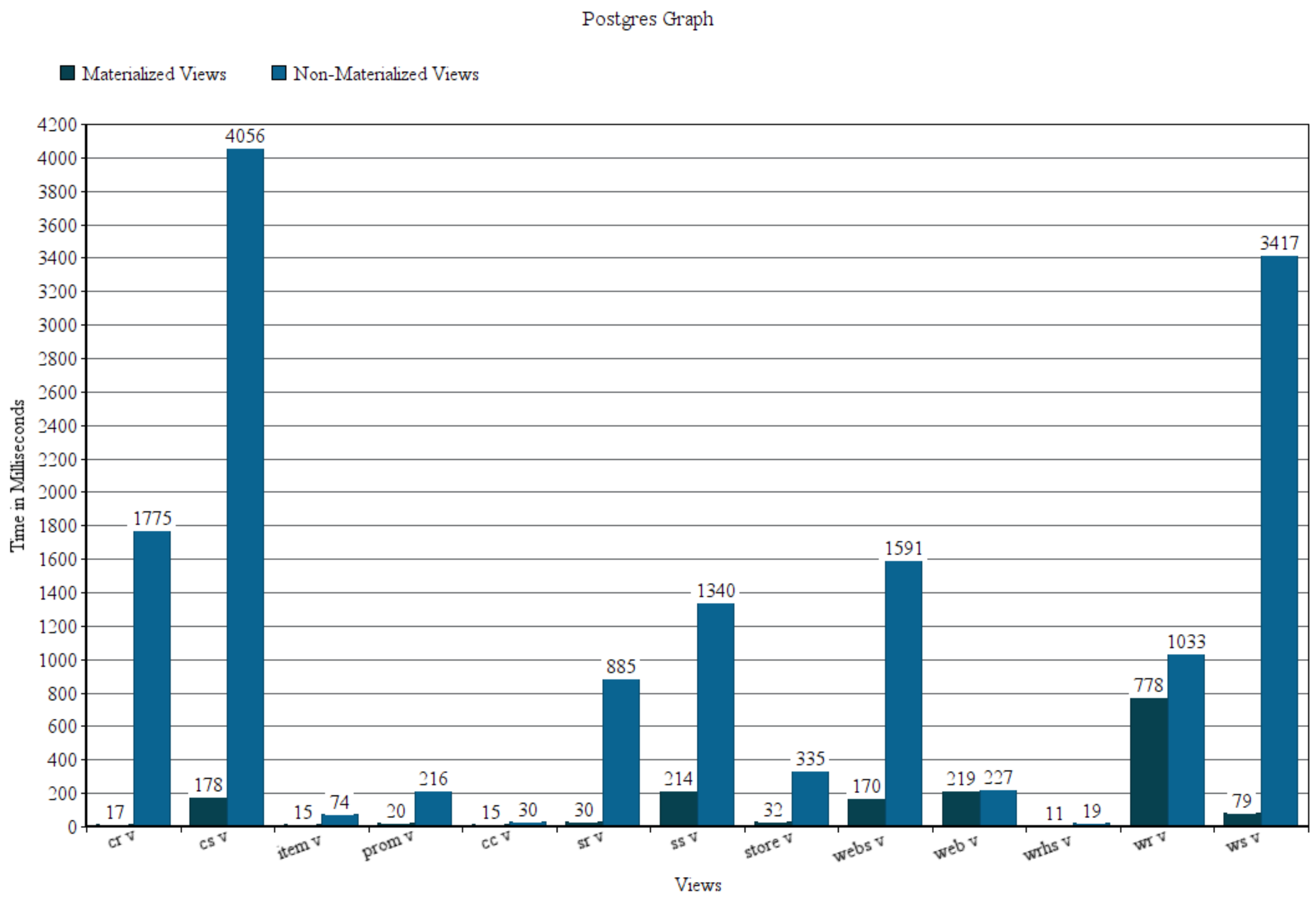


FIGURE 1.6: Query Response Time for POSTGRES Materialized and Non-Materialized Views

The above bar graph displays the difference between materialized views and non-materialized views in **POSTGRES**. Postgres directly supports view materialization which makes it easier to create a materialize view. As we can clearly see that the materialized views have a faster response time in Postgres. Every query was running 9 times and the execution time was noted down. After that the median value was taken from these run times. The execution time depended upon the size of the view. The size of the view was determined by how many tables were joined together to form that view. Each of these views were used to populate different tables. Both materialized and non-materialized views were run against **INSERT** statements.

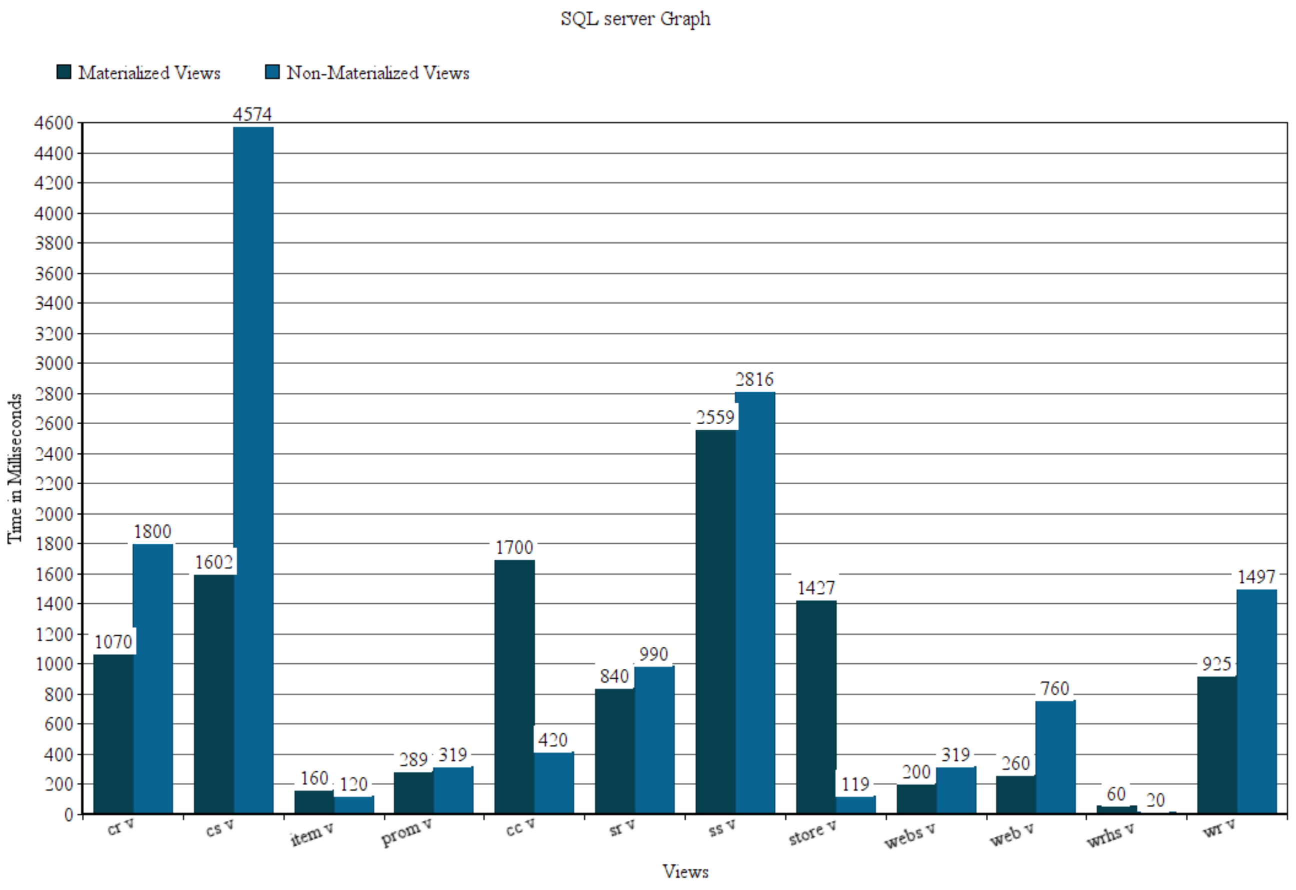


FIGURE 1.7: Query Response Time for SQL SERVER MANAGEMENT STUDIO Materialized and Non-Materialized Views

The above bar graph displays the difference between materialized and non-materialized views in **Microsoft SQL Server**. Microsoft SQL Server does not directly support view materialization. They differ from other RDBMS by the way of implementing materialized view via a concept known as "Indexed View". Since all views should be not materialized as that would not be an optimal approach. We can clearly see that in this bar graph that some non-materialized views respond faster than materialized views. Although it also depends upon the database management system. Since the job of materializing and non-materializing views is of the database administrator. We have proven from the above graph that materializing all views would not always be optimal.

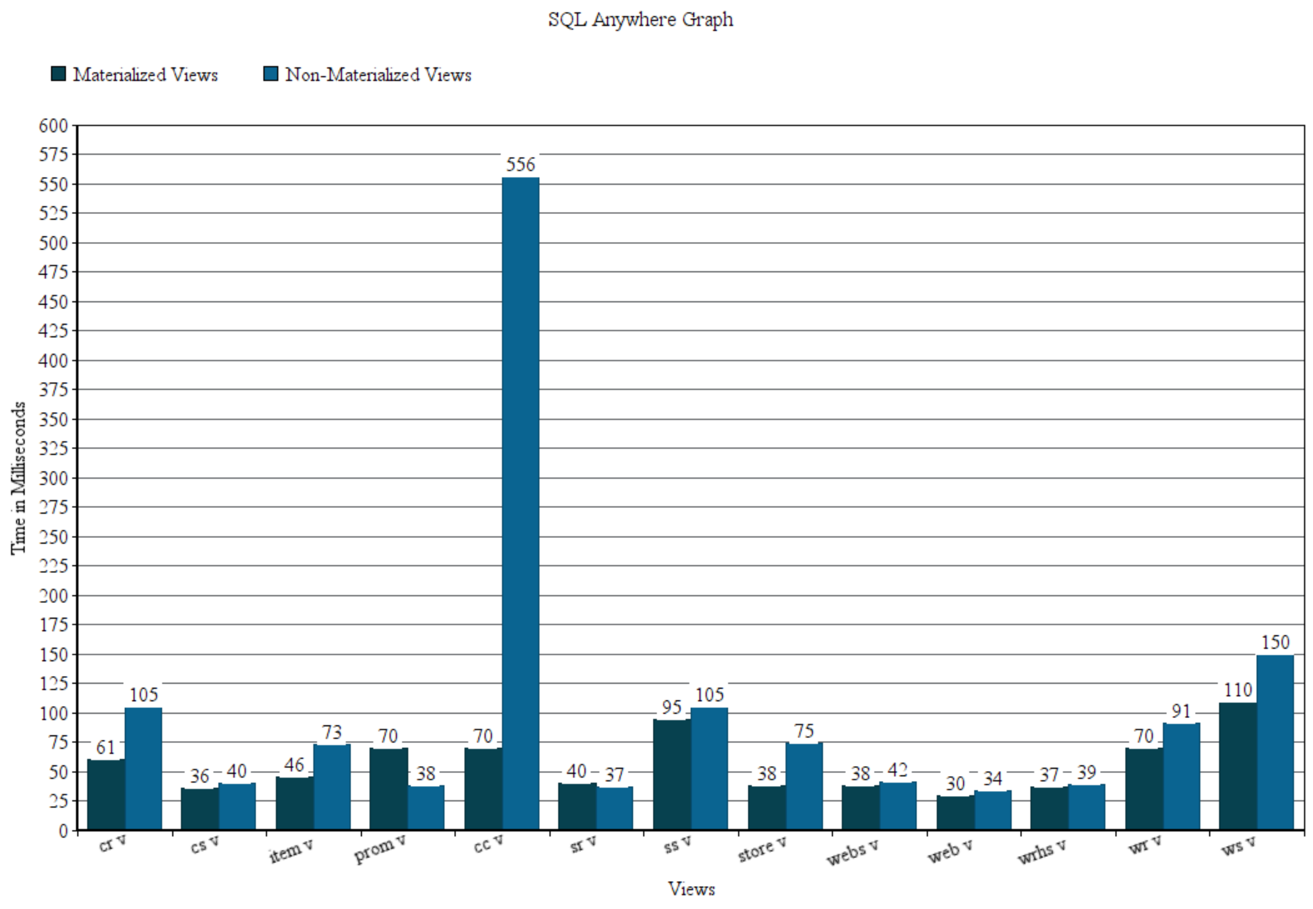


FIGURE 1.8: Query Response Time for SQL ANYWHERE Materialized and Non-Materialized Views

The above graph displays the difference between materialized views and non-materialized views in **SQL Anywhere**. SQL Anywhere directly supports view materialization. SQL anywhere proved to be very optimal in query response time. Upon creating the materialized views, the views had to be refreshed manually for initialization. If the materialized view was not refreshed it would not get initialized. In SQL anywhere we can see that some non-materialized views responded faster than the materialized views. This again proves our point that all views should not be materialized. It solely depends on the database administrator as to which views should be materialized and which not.

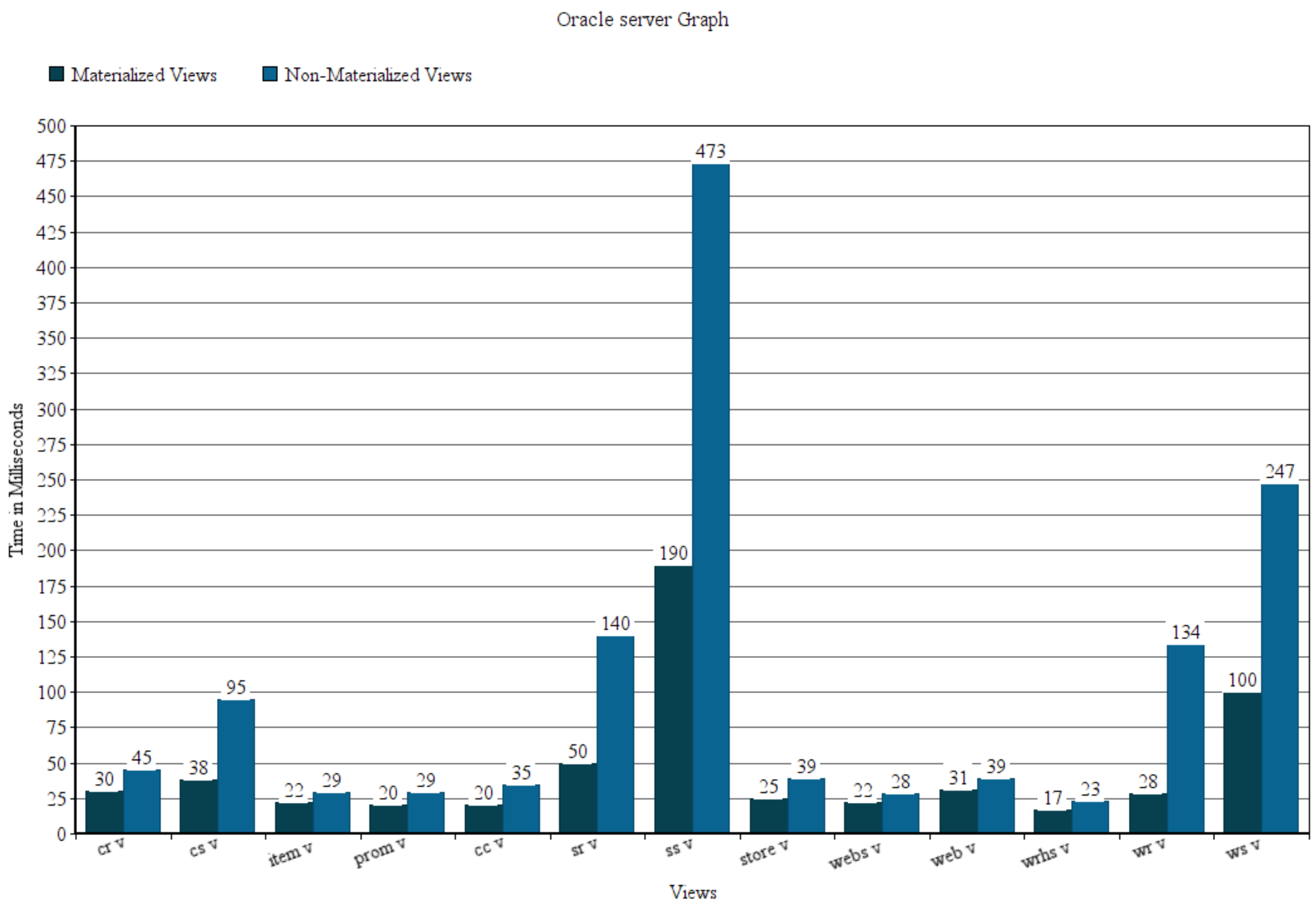


FIGURE 1.9: Query Response Time for ORACLE SQL Materialized and Non-Materialized Views.

The above graph displays the difference between materialized views and non-materialized views in **Oracle SQL**. Oracle SQL proved to be the most optimal for materialized views. As its response time was very less than the other database management systems.

5.4 Result Analysis:

According to our research and experiment results we can clearly see that materialized views are far better than non-materialized views. However, in some cases the non-materialized view response time is better than the materialized view. If we compare the materialized and non-materialized views. Both views consisted of the same query and were run against the same **SQL** Statements. Every database management system gave us different response times.

For Postgres the response for materialized views was way faster than non-materialized due to its support for materialized views. View materialization in Postgres is relatively easier than other database management systems. We can see that the non-materialized views which response time was longer, the materialized views were relatively faster. The difference between the response time of some materialized and non-materialized view is huge. The size of the views depended upon the number of **JOINS** used. Postgres handled these large views relatively better when these views were materialized.

For SQL SERVER query response time was comparatively larger from other database management systems even for materialized views. This is due to the fact that view materialization differs from other RDBMS. The concept used for materializing views in SQL Server was known as "Indexed Views". Since SQL Server does not directly supports view materialization the vast difference in query response time was expected. But we can see a difference between the materialized and non-materialized views. Even in SQL Server the materialized views responded faster even though the response time when compared with other RDBMS was larger. The view materialization did not disappoint.

For SQL Anywhere the query response time for materialize and non-materialized views was very optimal as compare to the previous two database management systems. The response time for both of these views was faster. SQL anywhere directly supports view materialization. Although upon creating the materialized views the views had to be initialized. These views were initialized through a command "Refresh Materialized View [View Name]".

For ORACLE SQL we determined that it was the best system for materialized views. View materialization was first introduced in ORACLE. Oracle provides many parameters for materializing a view which helps in executing the view much faster. From the above-mentioned results, we can clearly see that materialized views were much faster than non-materialized views. And if we compare oracle with other database management systems, we can see that it is relatively faster. For every query it hardly took time to respond.

Looking at the results of different database management systems we can see that materialized views responded faster than non-materialized views. However, in some cases we can see that the non-materialized view is faster than materialized view. This is due to the fact that materialized views are mainly used for aggregated data. If the query which we are storing in a view does not consist of aggregate functions. Then materializing that view would not make any sense. Since materialized views are stored physically on a database.

Database management systems depends upon the system requirements. A DBMS would require a fair amount of memory, CPU usage, RAM etc. In order to get the best results out of a database management system. The specifications of the system should either be matching their requirements or should be higher. We conducted our experiments on a single system and compared them with each other. We can not determine the best database management system based on a single system experiment. Ideally if we were to determine the best management system, we would have to perform these experiments on different systems with different specifications. But since this project was done on a lower level we had no choice but to experiment on a single system. We performed these experiments on a single system providing a fair environment for each of these database management systems.

5.5 Research Application:

Our work was mostly research and experimentations. A graphical application can be developed in order to help database administrators. Based on our work this application will help them test which database management system would be optimal for their system. It could help them to calculate response time for their views, which views would be optimal for materialization and which views do not needs to be Materialized. This application will save their time and will help reducing the response time for materialized and non-materialized views.

Conclusion of Chapter:

The above chapter explained various aspects of our project. The system architecture was briefly explained. The process of experimenting and calculating results was deeply explained. The testing environment was explained which tests were to be performed using which libraries and which language

The experiments performed were then shown graphically for a better understanding. Each of these graphs were briefly explained. The results calculated during these experiments were thoroughly analyzed in this chapter.

Chapter 6

System Testing and Evaluation

This chapter consist of database implementation how we implemented it, compatibility and what problems occurred during the implantation of it. This chapter also include the various software's that we have used. We will discuss the system we performed our experiments and implemented the database. Database management systems which were an essential part of our experiments will also be discussed.

6.1 Database Implementation:

The TPC-DS dataset consisted of large number of records. The maximum dataset size was 100TB. In order to handle the data easily only 1 GB of this data was downloaded into the system. The dataset originally consisted of 24 tables but since we were working on views. We needed update our dataset. The updated dataset was called refreshed data which then added up to 44 tables. The data was generated into flat files delimited "|". Here a problem arose.

The data was delimited by "|" in the flat files. Meaning the columns were separated using "|". The pipe "|" was present at end of every column which became a problem when loading the data into any database management system. The columns were delimited by "|" and the row terminator was also "|". Which made it difficult to terminate rows since it could not differentiate between a column end a row end.

The flat icons were the converted into csv files using Microsoft Excel. The "|" delimiter was replaced by "," and the row terminator was replaced "\n". Other than the data loading went very smoothly.

6.2 Compatibility:

The experiments were executed on a macOS system. In my opinion no compatibility issues will arise due to operating systems. Since all the software's that are used during these experiments are easily accessible.

The software's used during our experiments are as follows:

- **PostgreSQL (Pg Admin)**
- **Microsoft SQL Server**
- **SQL Anywhere (Sybase Centeral)**
- **Oracle SQL (Oracle SQL Developer)**
- **Visual Studio Code**

All these software's are easily available all over the internet for every operating system. The database management systems were only used for loading data into the database. Rest of the work was conducted in visual studio code using python as the programming language and anaconda as base environment.

Conclusion of Chapter:

This chapter explained the various software’s, compatibility and the system the experiments were conducted on. Since we are working on a single operating system the compatibility with another operating system was also discussed in this chapter. The problems occurred during the implementation and how we overcame and handled them is also mentioned in this chapter.

Chapter 7

Conclusions

This was a brief report detailing our project "Materialized vs Non-Materialized views in contemporary databases". In which we detailed that we aim to achieve i.e. Generation of materialized views. Generation of non-materialized views. Comparing the response time of these views programmatically. We have performed our experiment on different database management systems and compared their response times.

We gathered different database management systems which use "STRUCTURED QUERY LANGUAGE" (SQL) to perform our experiments. We connected our databases to python programming language via a connection string. After connecting with the database management systems, we started our experiments.

We generated materialized and non-materialized views programmatically through python. After generating these views, we ran multiple different queries on these views. We calculated the query response time using python's inbuilt functions.

We calculated accurate results of every query which used materialized and non-materialized views. The response time was calculated in milliseconds. These results were displayed in the form of tables and graphs. Since we calculated our results on the same system to provide a fair environment for our database management systems, we can say that the accuracy of our results is satisfying.

We achieved our goal of providing results between the response time of materialized and non-materialized views. We can say that which database management system was the best by looking at our results. But since the database management systems depend on the environment, they are used in we are more focused on the response times of materialized and non-materialized views. We cannot say which database management system is the best based on our single system environment.

Since in our experiments the views were generated and selected by the database administrator. For our future work we can focus on generating and selecting these views automatically using artificially using either machine learning or artificial intelligence. As the world is converting its data into digital form the data is going to keep on increasing. We need to focus on how we are going to maintain that data. Since data is growing larger by the day. It will be very optimal if we can automate the maintenance of this data using modern technology.

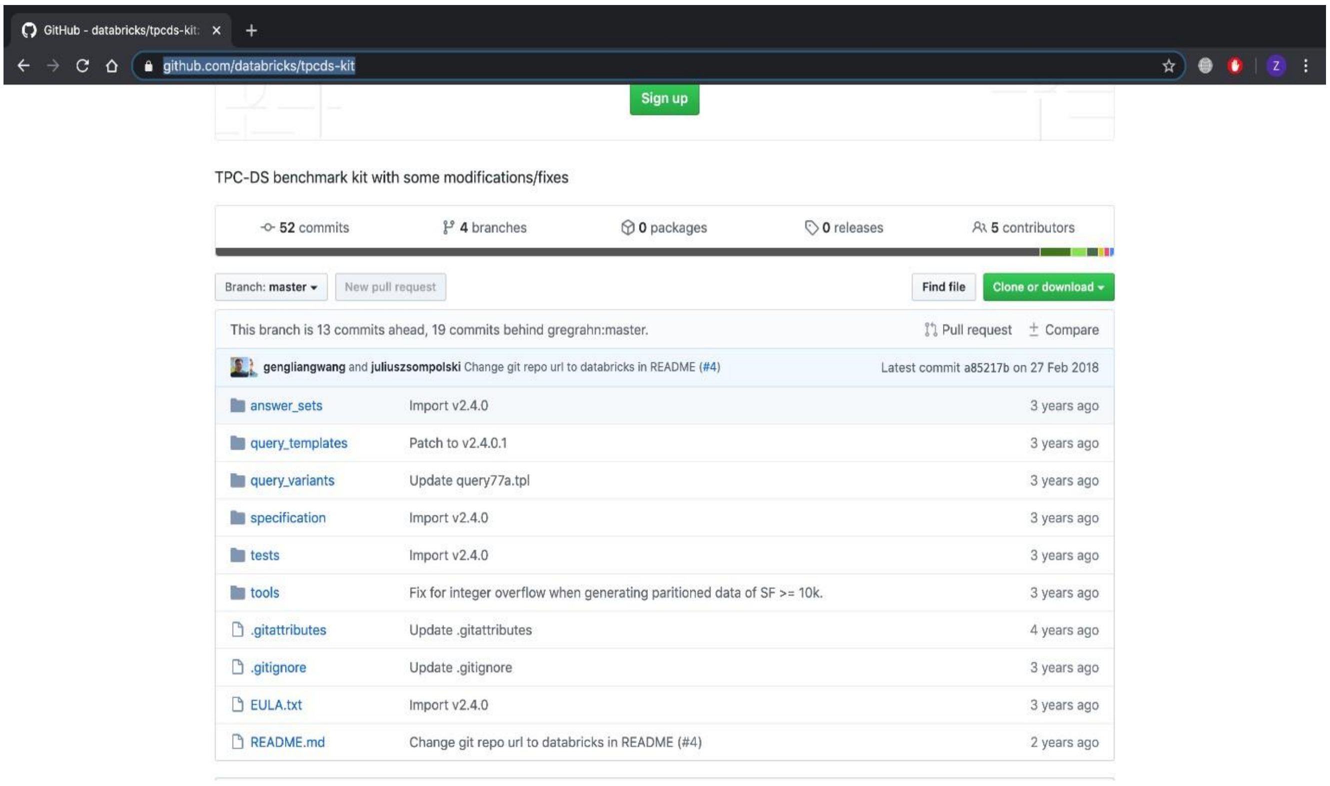
Appendix A

User Manual

Following is a step by step guide on how to perform the above-mentioned experiments.

Step 1:

Download the TPC-DS benchmark from https://github.com/databricks/tpcds-kit



GitHub - databricks/tpcds-kit
github.com/databricks/tpcds-kit
Sign up

TPC-DS benchmark kit with some modifications/fixes

52 commits | 4 branches | 0 packages | 0 releases | 5 contributors

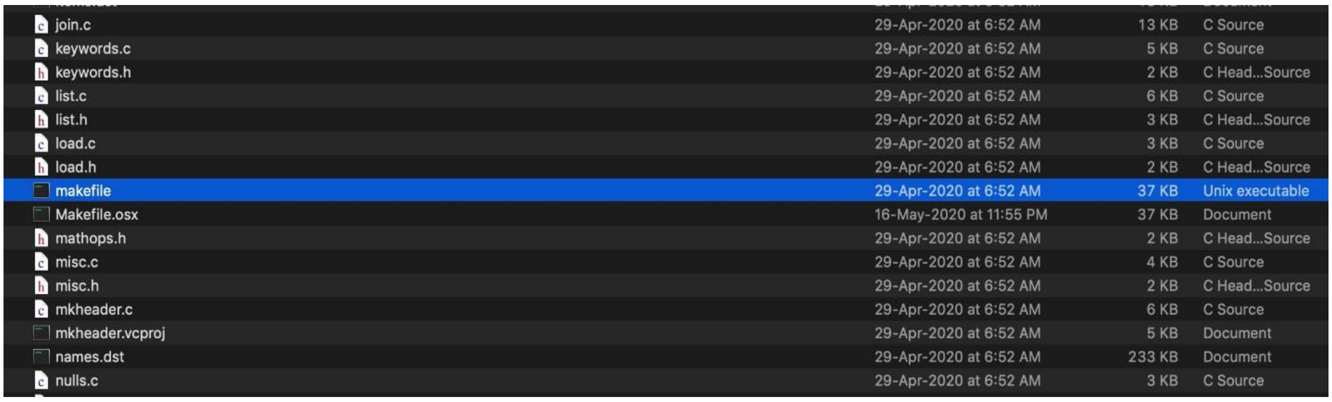
This branch is 13 commits ahead, 19 commits behind gregrahn:master.

gengliangwang and juliuszsompolski Change git repo url to databricks in README (#4) Latest commit a85217b on 27 Feb 2018

answer_sets	Import v2.4.0	3 years ago
query_templates	Patch to v2.4.0.1	3 years ago
query_variants	Update query77a.tpl	3 years ago
specification	Import v2.4.0	3 years ago
tests	Import v2.4.0	3 years ago
tools	Fix for integer overflow when generating paritioned data of SF >= 10k.	3 years ago
.gitattributes	Update .gitattributes	4 years ago
.gitignore	Update .gitignore	3 years ago
EULA.txt	Import v2.4.0	3 years ago
README.md	Change git repo url to databricks in README (#4)	2 years ago

Step 2:

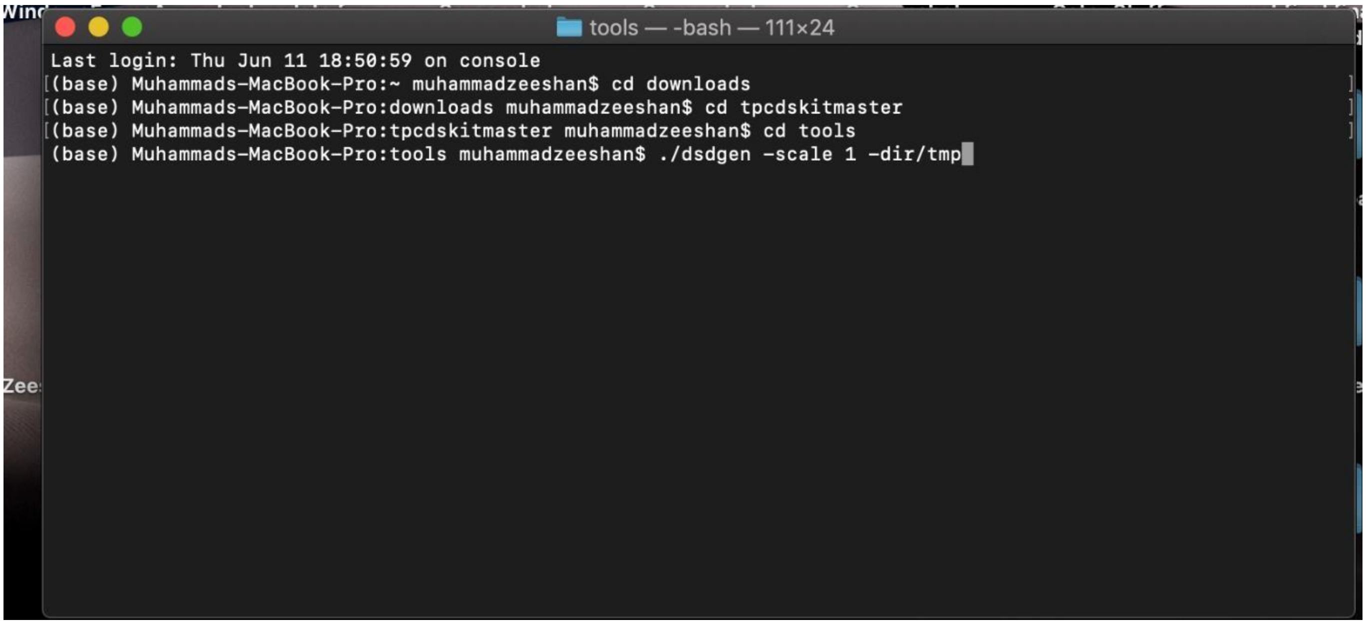
After downloading and extracting the benchmark open the tools folder in the TPC-DS kit folder and run an executable file called MAKEFILE.



join.c	29-Apr-2020 at 6:52 AM	13 KB	C Source
keywords.c	29-Apr-2020 at 6:52 AM	5 KB	C Source
keywords.h	29-Apr-2020 at 6:52 AM	2 KB	C Head...Source
list.c	29-Apr-2020 at 6:52 AM	6 KB	C Source
list.h	29-Apr-2020 at 6:52 AM	3 KB	C Head...Source
load.c	29-Apr-2020 at 6:52 AM	3 KB	C Source
load.h	29-Apr-2020 at 6:52 AM	2 KB	C Head...Source
makefile	29-Apr-2020 at 6:52 AM	37 KB	Unix executable
Makefile.osx	16-May-2020 at 11:55 PM	37 KB	Document
mathops.h	29-Apr-2020 at 6:52 AM	2 KB	C Head...Source
misc.c	29-Apr-2020 at 6:52 AM	4 KB	C Source
misc.h	29-Apr-2020 at 6:52 AM	2 KB	C Head...Source
mkheader.c	29-Apr-2020 at 6:52 AM	6 KB	C Source
mkheader.vcproj	29-Apr-2020 at 6:52 AM	5 KB	Document
names.dst	29-Apr-2020 at 6:52 AM	233 KB	Document
nulls.c	29-Apr-2020 at 6:52 AM	3 KB	C Source

Step 3:

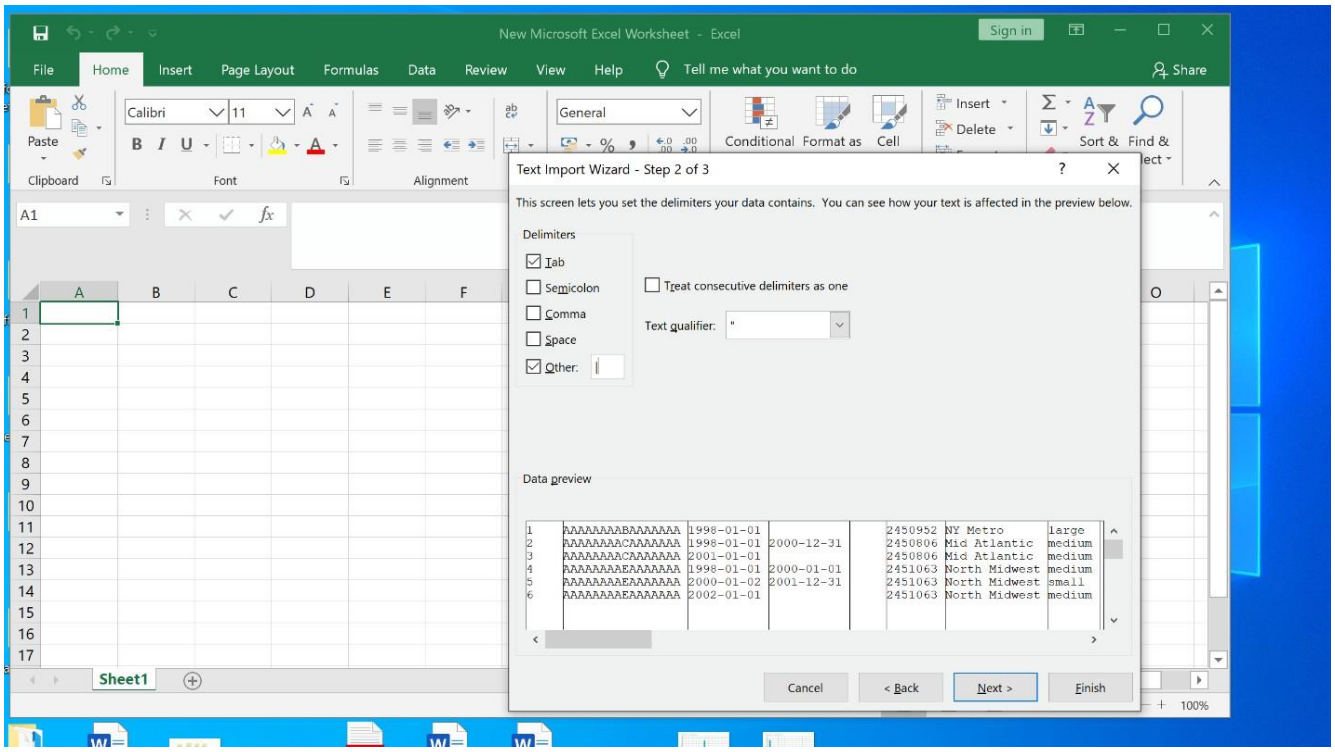
After running the make file. Open up terminal (or command prompt if using windows) go into the TPC-DS tools directory. In the tools directory run command "./dsdgen – scale 100 -dir/tmp" Scale defines the amount of data you need to generate. The official scale factors are 1TB, TB, 10 TB etc. We generated a total of 1gb data so we used SF= 1. -dir/tmp defines which directory to store the generated data in.



```
Last login: Thu Jun 11 18:50:59 on console
(base) Muhammads-MacBook-Pro:~ muhammadzeeshan$ cd downloads
(base) Muhammads-MacBook-Pro:downloads muhammadzeeshan$ cd tpcdskitmaster
(base) Muhammads-MacBook-Pro:tpcdskitmaster muhammadzeeshan$ cd tools
(base) Muhammads-MacBook-Pro:tools muhammadzeeshan$ ./dsdgen -scale 1 -dir/tmp
```

Step 4:

The files generated from the dsdgen are in .dat format which are then converted into csv files using Microsoft Excel. The columns are delimited by “|”



Text Import Wizard - Step 2 of 3

This screen lets you set the delimiters your data contains. You can see how your text is affected in the preview below.

Delimiters

- [✓] Tab
- [ ] Semicolon
- [ ] Comma
- [ ] Space
- [✓] Other: |

[ ] Treat consecutive delimiters as one

Text qualifier: "

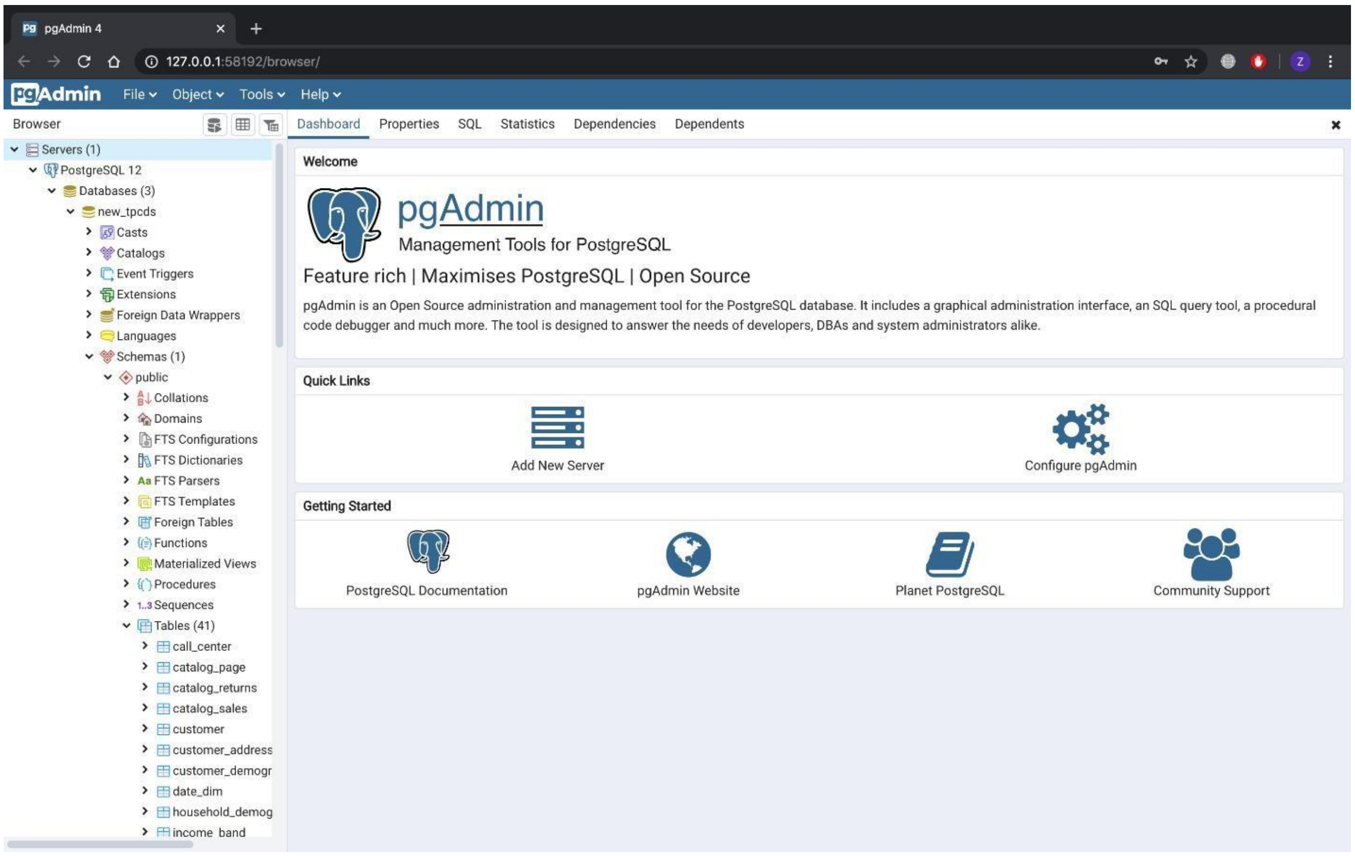
Data preview

```
1  AAAAAAAABAAAAAAA 1998-01-01                       2450952 NY Metro       large
2  AAAAAAAACAAAAAAA 1998-01-01 2000-12-31            2450806 Mid Atlantic   medium
3  AAAAAAAACAAAAAAA 2001-01-01                       2450806 Mid Atlantic   medium
4  AAAAAAAAEAAAAAAA 1998-01-01 2000-01-01            2451063 North Midwest  medium
5  AAAAAAAAEAAAAAAA 2000-01-02 2001-12-31            2451063 North Midwest  small
6  AAAAAAAAEAAAAAAA 2002-01-01                       2451063 North Midwest  medium
```

Cancel | < Back | Next > | Finish

Step 5:

After you have generated the dataset. Download any database management studio which support View Materialization. For this manual purpose I will be using POSTGRES.

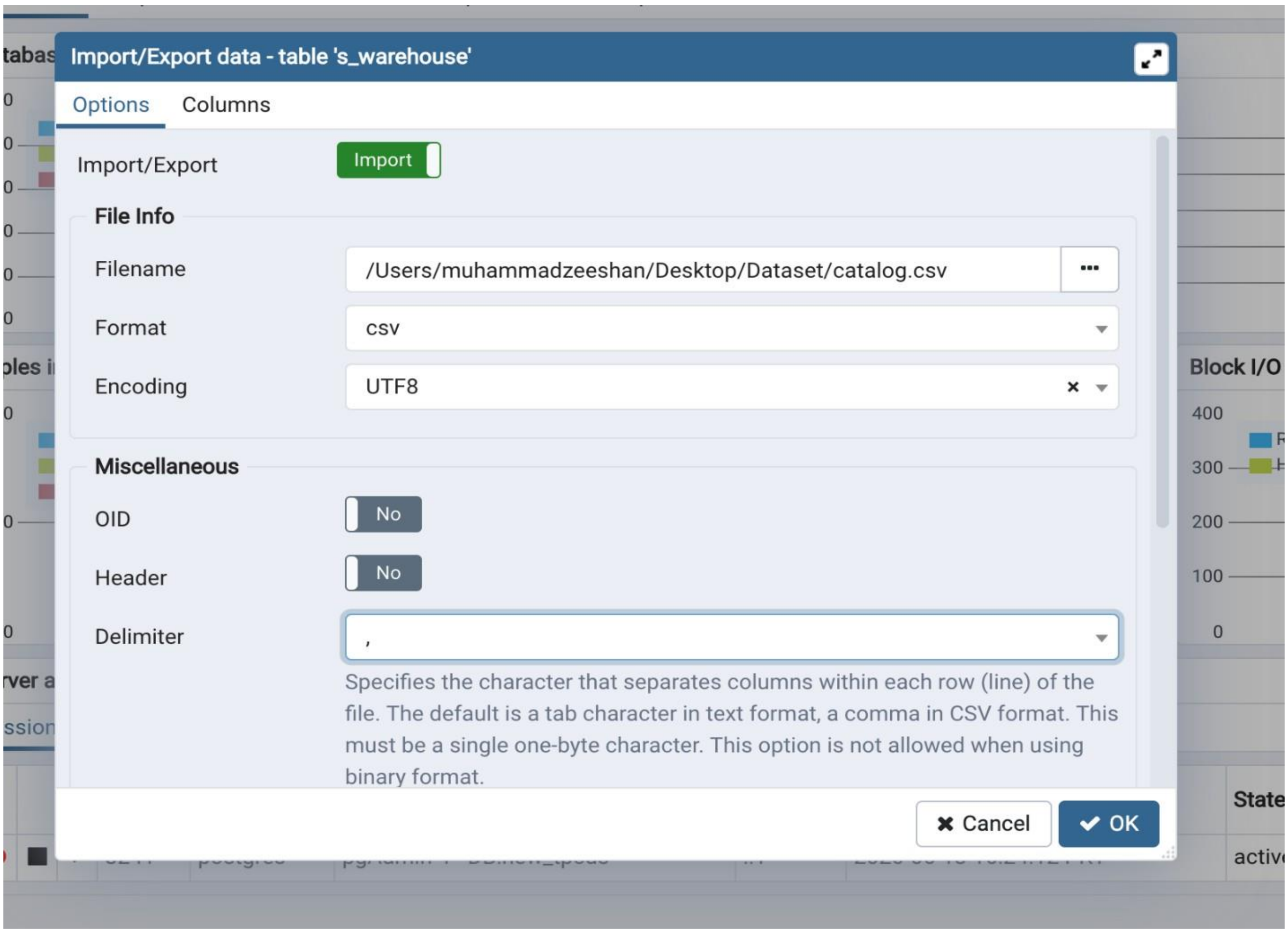


Feature rich | Maximises PostgreSQL | Open Source

pgAdmin is an Open Source administration and management tool for the PostgreSQL database. It includes a graphical administration interface, an SQL query tool, a procedural code debugger and much more. The tool is designed to answer the needs of developers, DBAs and system administrators alike.

Step 6:

Create a database in the database management system and create tales. The create table SQL queries are provided in the TPC-DS-tools folder named “tpcds source.sql”. After creating the tables import the data into those tables.



Import/Export data - table 's_warehouse'

Options | Columns

Import/Export: Import

File Info

Filename: /Users/muhammadzeeshan/Desktop/Dataset/catalog.csv

Format: csv

Encoding: UTF8

Miscellaneous

OID: No

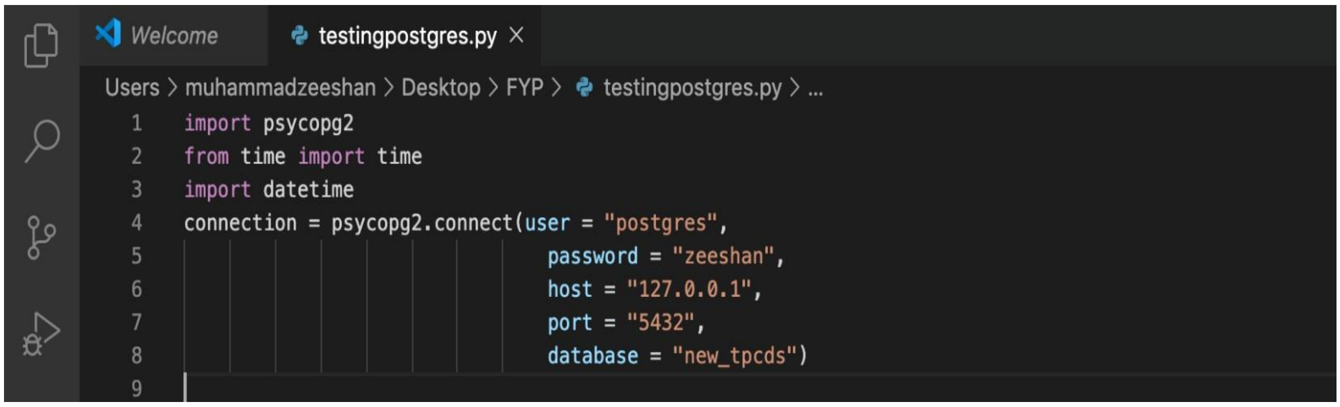
Header: No

Delimiter: ,

Specifies the character that separates columns within each row (line) of the file. The default is a tab character in text format, a comma in CSV format. This must be a single one-byte character. This option is not allowed when using binary format.

Step 7:

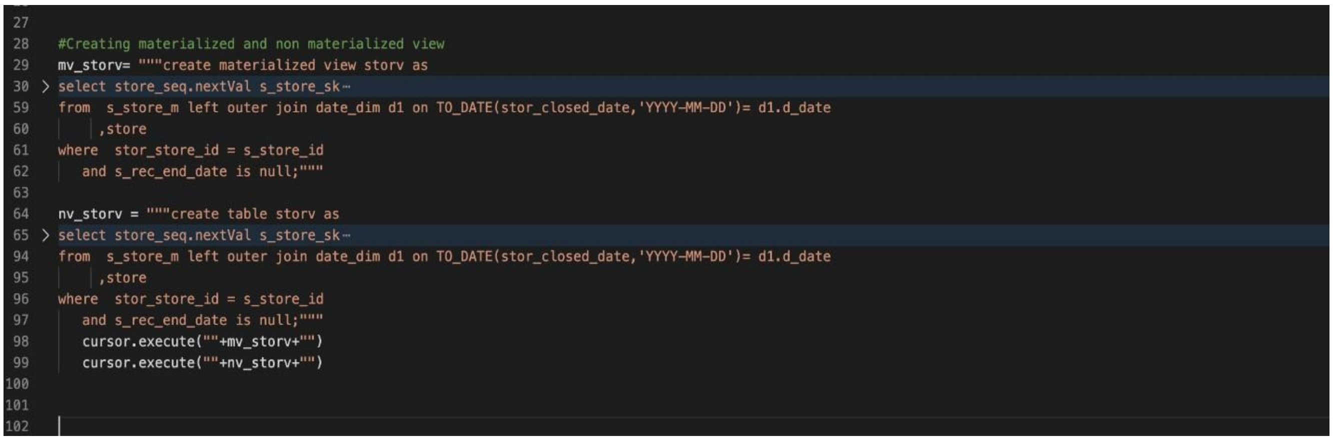
Connect the database management system with python programming language using appropriate connection string. For every database management system, the connection string varies.


Welcome
testingpostgres.py
Users > muhammadzeeshan > Desktop > FYP > testingpostgres.py > ...

```python
import psycopg2
from time import time
import datetime
connection = psycopg2.connect(user = "postgres",
                              password = "zeeshan",
                              host = "127.0.0.1",
                              port = "5432",
                              database = "new_tpcds")

```

Step 8:

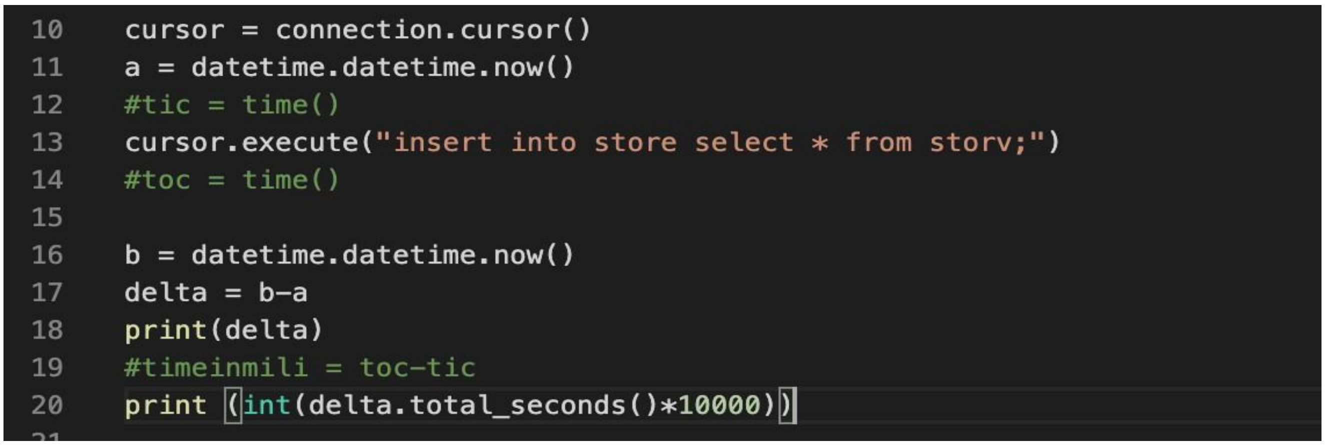
After connecting with python generate materialized and non-materialized views into the database management system. The view queries can be found in TPC-DS documentation found on www.tpc.org



```python

#Creating materialized and non materialized view
mv_storv= """create materialized view storv as
select store_seq.nextVal s_store_sk⋯
from  s_store_m left outer join date_dim d1 on TO_DATE(stor_closed_date,'YYYY-MM-DD')= d1.d_date
        ,store
where  stor_store_id = s_store_id
   and s_rec_end_date is null;"""

nv_storv = """create table storv as
select store_seq.nextVal s_store_sk⋯
from  s_store_m left outer join date_dim d1 on TO_DATE(stor_closed_date,'YYYY-MM-DD')= d1.d_date
        ,store
where  stor_store_id = s_store_id
   and s_rec_end_date is null;"""
   cursor.execute(""+mv_storv+"")
   cursor.execute(""+nv_storv+"")



```

Step 9:

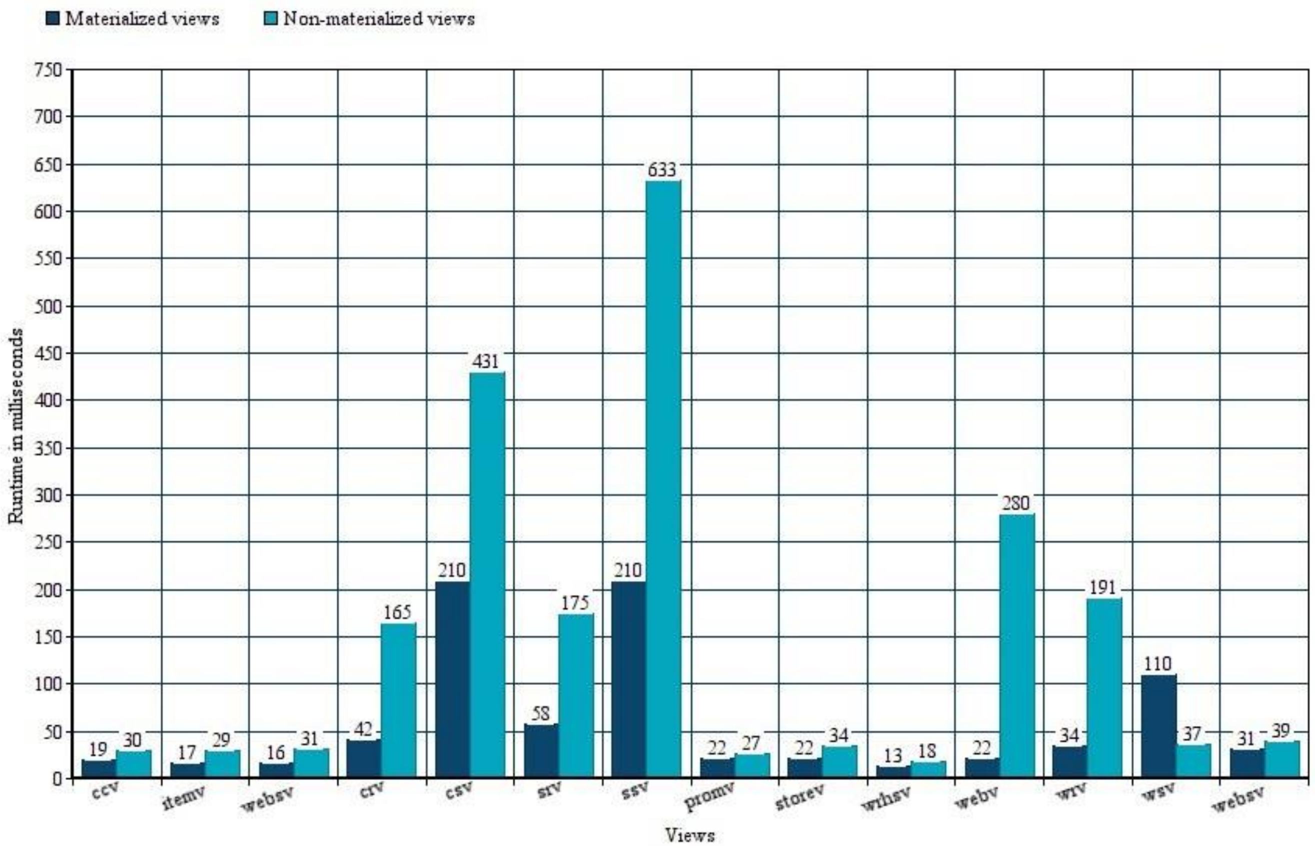
After generating views. Execute the select queries which use these views in them. Those queries can be found in tpcds/tests folder with “sql” extension. Calculate the execution time of these queries using in built functions of Python.



```
cursor = connection.cursor()
a = datetime.datetime.now()
#tic = time()
cursor.execute("insert into store select * from storv;")
#toc = time()

b = datetime.datetime.now()
delta = b-a
print(delta)
#timeinmili = toc-tic
print (int(delta.total_seconds()*10000))
```

Step 10:

Display the time calculated in the form of tables and graphs.



References

[1] Jonathan Goldstein & Per-Ake Larson. Optimizing Queries Using Materialized Views: A Practical Scalable Solution,2001. Microsoft Research, One Microsoft Way, Redmond, WA 98052.

[2] Surajid Chaudhuri, Ravi Krishnamurthy, Spyros Potamianos & Kyuseok Shim. Optimizing Queries with Materialized Views, 2001. Microsoft Research, Redmond, WA. Hewlett-Packard Laboratories, Palo Alto, CA. Momferatory 71, Athens, Greece. IBM Almdaen Research Center, San Jose, CA.

[3] Rada Chirkova, Jun Yang. Materialized Views, November 2012. Now Publishers Inc, P.O. Box 1024, Hanover, MA, United States.