# ADBMS

15/15

SDS | Page No. | Date / /

Name :- Preksha Ashok Patel
Sapid :- 60004210126
Branch :- Computer Engineering
Div :- C2 , Batch :- C21
Date of Performance :- 09/09/2023
Date of Submission :- 16/09/2023

## EXPERIMENT - 1:

Case Study on professional and commercial database System

Aim :- Case Study on a professional & commercial database.

Database (The One You're Recommending):
The database that we are recommending is Snowflake.

Features and Description of the database :-
Snowflake is a cloud-hosted relational database for building data warehouses. It's built on AWS, Azure, and Google cloud platforms and combines the functionalities of traditional databases with a suite of new and creative capabilities.
The Features of the database are :-

1] CLOUD BASED :-
Snowflake is a cloud based data warehouse which means that it can be accessed from anywhere using an internet connection.

2] Elastic :-
Snowflake is elastic. It can automatically scale resources to meet the demands of your queries.

3] Versatility :-
Snowflake is elastic. It can be used for a variety of analytical workloads, including data warehousing, data lakes, and machine learning.

4] Provides Automated Scalability Solutions :-
Users of the Snowflake database can increase cloud computing

Speed and storage separately using the Snowflake cloud services. The Snowflake cloud data platform comes with auto-scaling and auto-suspend features to further reduce the strain from maint-ance and administrative work involved in running your cloud infrastru Snowflake supports both vertical & horizontal scaling of computing resources.

5] Provides Support for Semi-Structured Snowflake :-
Snowflake boasts one of the most versatile platforms for handling semi-structured data sets among database providers. Snowflake offers support for semi-structured data, which means it can handle data that doesn't fit neatly into traditional, structured database tables (rows & columns). Semi-structured data often includes various data formats, such as JSON, XML or Avro, where the data doesn't confirm to a rigid schema. This ensures data & schema flexibility.

6] Allows Compartmentalization and concurrency management :
In Snowflake, databases are logical containers that organize and isolate data and database objects. Each database can have its own schema (s), tables, views, and stored procedures. This Compartmentalization is useful for organizing data & managing access control. Snowflake's architecture is designed for concurrency. It uses a multi-cluster, shared-data architecture that separates compute resources from storage. This means multiple compute clusters can work on the do same data simultaneously, providing excellent concurrency capabilities.

7] Minimal Maintenance and Administration :-
The Unique Snowflake architecture means that it is possible to develop solutions, even without imposing heavy dependencies on the client's database administrators or IT teams. There's no need to commission software installation or install physical terminals.

8] Snowflake Data Warehouse Security :-

Snowflake is known for its robust security features, which are essential for protecting sensitive data in a cloud-based data warehousing environment. It uses data encryption, multifactor authentication :, audit logging in order to make snowflake Secure.

The Company to Whom you are recommending :- Walmart

Walmart is the world's largest retailer, with over 11,500 stores in 27 countries. Walmart sells a wide variety of products, including groceries, clothing, electronics & home goods. Walmart is known for its low prices & convenient locations. It is a big data company, which collects and stores vast amounts of data on its customers, products, and sales. Walmart uses this data to gain insights into its business & make better decisions. Walmart is using data analytics to improve its customer service, supply chain management, and fraud detection.

According to a 2016 article by Forbes, Walmart began using Amazon Redshift in 2015. The article states that Walmart was using Redshift to store & analyze data from its stores.

Requirements / Features of the Company

Walmart's database software must support the following requirements:

- Scalability : Walmart's database software must be able to scale to handle its vast amount of data. Walmart's data is constantly growing, so the database software needs to be able to keep up with the demand.
- Performance : Walmart's database software must be able to perform queries and analysis quickly. Walmart needs to be able to get insights from its data in real time.

- Ease of use : Walmart's database software must be easy to use for its data analysts. Walmart's data analysts need to be able to focus on analyzing data, not managing the database infrastructure.
- cost - effectiveness :- Walmart's database software needs to be able to afford to store & analyze all of its data.
- Support for Complex analytical workloads : Walmart's database software must support workloads, such as machine learning & NLP.
- Support for data integration :- Walmart's database must support from a variety of sources, such as its stores, e-commerce platform & suppliers.
- Security :- It must be secure to protect its data from unauthorized access.
- Reliability :- It must be reliable to ensure that its data is always available when needed.
- Compliance :- It must comply with all applicable laws & regulations.

Help Analyze customer's behaviour & improve customer Service :- Walmart's customer behavior data is needed to be analysed to identify trends and patterns which can help Walmart in its marketing campaigns, product offerings, & store layouts.

Optimize supply chain :- Snowflake could be used to optimize Walmart's supply chain by tracking inventory levels & demand. This information could be used to ensure that Walmart has the right products in the right stores at the right time.

## Comparison With the Currently used database :-

| Feature | SNOWFLAKE | REDSHIFT |
|---|---|---|
| Scalability | More Scalable | Less Scalable |
| Elasticity | More Elastic | Less Elastic |
| Security | Good | Good |
| Versatility | Very Versatile | Versatile |
| Pricing | Pay-as-you-go | Pay-as-you-go |
| Ease of Use | Easier to use | More difficult to use |

Snowflake is a bit more expensive than Redshift as it charges separately for warehousing & computation. However, when customers avail reserved instance pricing through Redshift, the total expense margin drops considerably when compared to Snowflake.

### Snowflake vs Redshift Security :-

Data Security is the most crucial aspect when it comes to warehouses. Loading data in Redshift comes in two types, Server-side Encryption and client-side encryption. The decryption process is taken up transparently when you load data from server-side & decrypts the data as it loads the table when done from the client-side. Data is always on a transit within the AWS cloud and to protect it. Amazon Redshift uses hardware accelerated SSL which helps to copy, backup & restore data. With Snowflake computing, each & every object in the account is secured, says, warehouse, database, clusters, tables, users, etc. The major advantage with Snowflake computing is that it encrypts data automatically that kept for both loading & unloading.

Snowflake vs Redshift Integrations :-

Integrations are one key factor users consider before opting for a data warehousing system.

However, when businesses ~~of~~ hugely rely on JSON storage then snowflake certainly has an upper hand over Redshift. The in-built architecture of snowflake schemas allows users to query & store easily whereas with Redshift, spillage of queries results in strained processes.

Snowflake vs Redshift Performance :-

The key difference is that Redshift generally takes a longer time for query optimization but as these queries are run repeatedly and on daily basis, they tend to be faster. This isn't the case when it comes to snowflakes, it offers a much better performance with read queries.

Snowflake has always been a tool that performed concurrent scaling, as its computation & storage are different. Amazon Redshift has newly implemented concurrent scaling too.

Overall, Snowflake is a powerful & versatile data warehouse that could help walmart to improve its business in a no. of ways.

SDS | Page No.
Date / /

## CONCLUSION :-

While both Amazon-Redshift & Snowflake have their own strengths and benefits, Snowflake can be concluded to better align with Walmart's requirements, specifically that of storing & analyzing large amounts of data from a variety of sources. Snowflake is known for its scalability, performance, and the ease in which it can be used by professionals.
It is an excellent choice for the organization, such as Walmart, that need to store & analyze large amounts of data from a variety of sources. Snowflake, thus offers a comprehensive solution for Walmart's database needs of high scalability, excellent performance, in terms of quick querying & analyzing of data, cost-effectiveness & ease of use.
Snowflake can hence be regarded as an effective option to meet Walmart's various requirements & upload its progress & success.

**NAME:-PREKSHA ASHOK PATEL**

**SAPID:-60004210126**

**BRANCH:-COMPUTER ENGINEERING**

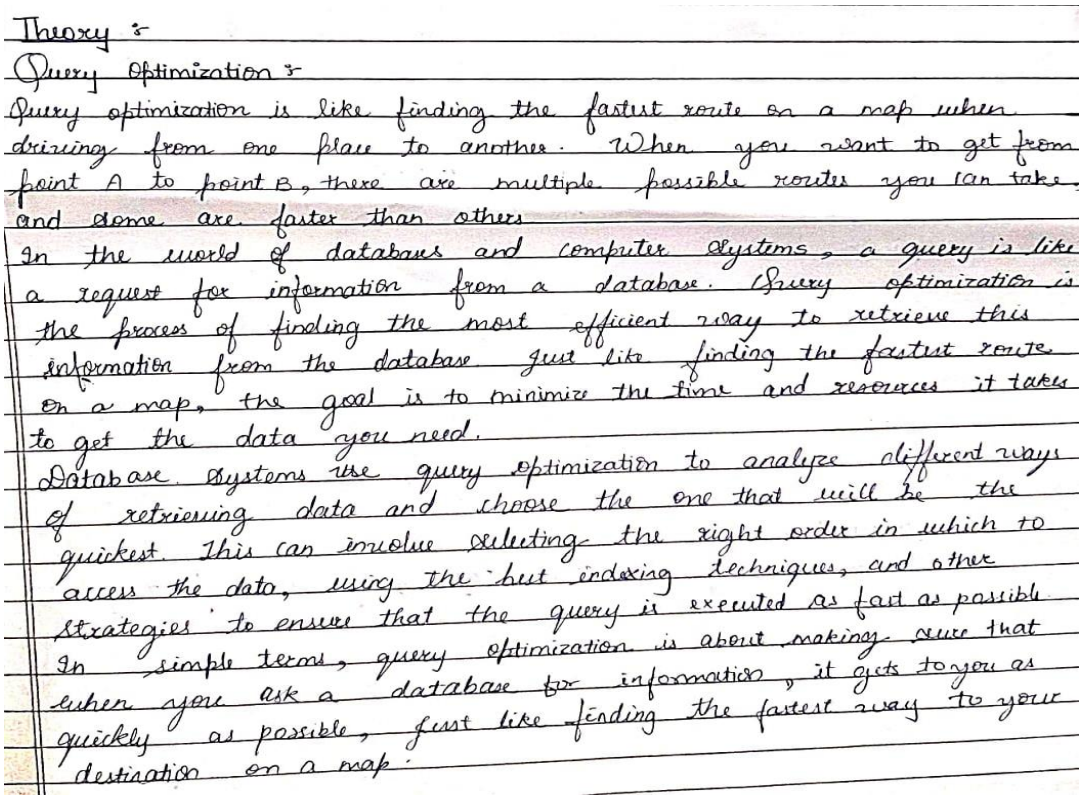
**DIV:-C2 ; BATCH:-1**

# ADBMS

# EXPERIMENT-02

**AIM:** STIMULATE QUERY OPTIMISATION BY APPLYING SQL QUERY ON YOUR SELECTED DATABASE.

**THEORY:**



Theory :-

Query Optimization :-

Query optimization is like finding the fastest route on a map when driving from one place to another. When you want to get from point A to point B, there are multiple possible routes you can take, and some are faster than others.

In the world of databases and computer systems, a query is like a request for information from a database. Query optimization is the process of finding the most efficient way to retrieve this information from the database. Just like finding the fastest route on a map, the goal is to minimize the time and resources it takes to get the data you need.

Database systems use query optimization to analyze different ways of retrieving data and choose the one that will be the quickest. This can involve selecting the right order in which to access the data, using the best indexing techniques, and other strategies to ensure that the query is executed as fast as possible.

In simple terms, query optimization is about making sure that when you ask a database for information, it gets to you as quickly as possible, just like finding the fastest way to your destination on a map.

Indexing :-

Indexing in a database is like creating a quick-reference guide for data. It organizes specific columns in a table, making data retrieval faster by providing a roadmap to find records matching certain criteria, but it requires maintenance & should be used judiciously to balance query performance with storage and update efficiency.

An index in a relational database is like an ordered list of values from one or more columns in a table, with pointers to the actual rows. It's used to speed up data retrieval operations such as SELECT queries.

Syntax for creating an index :-

```
CREATE INDEX index_name
ON table_name (column1, column2, ....);
```

Optimize table :-

Optimize table is a SQL statement that defragments and reorganizes the data in a table to improve performance. It is especially useful for tables that have been heavily updated or deleted from.

Syntax :-

```
optimize table table-name ;
```

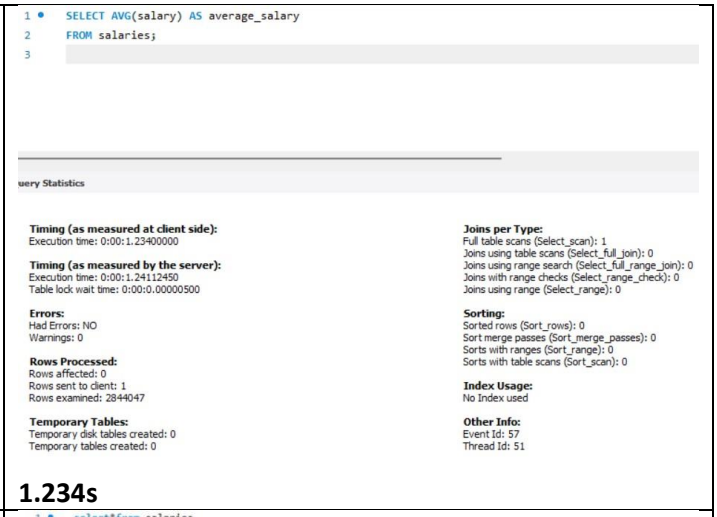
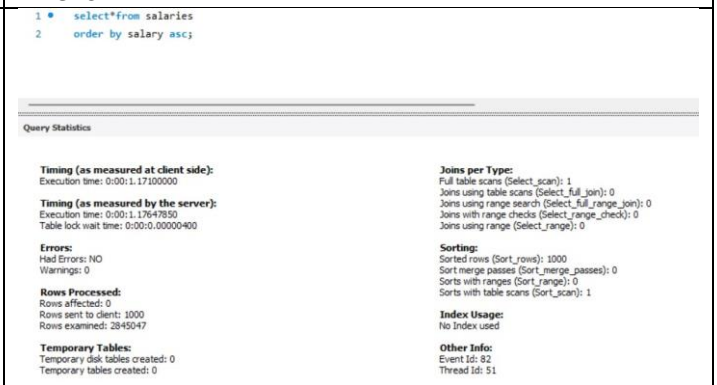
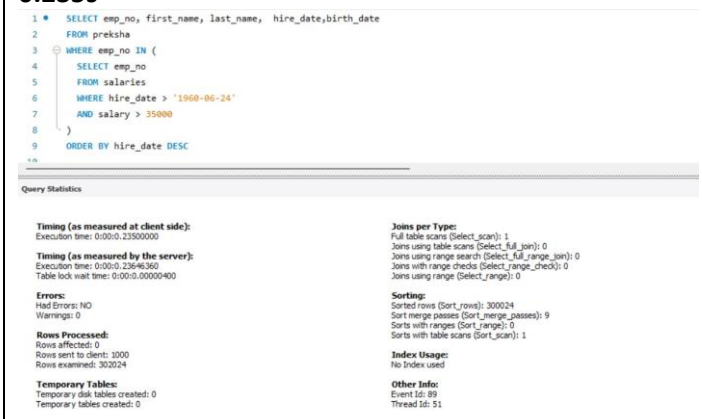
Conclusion :-

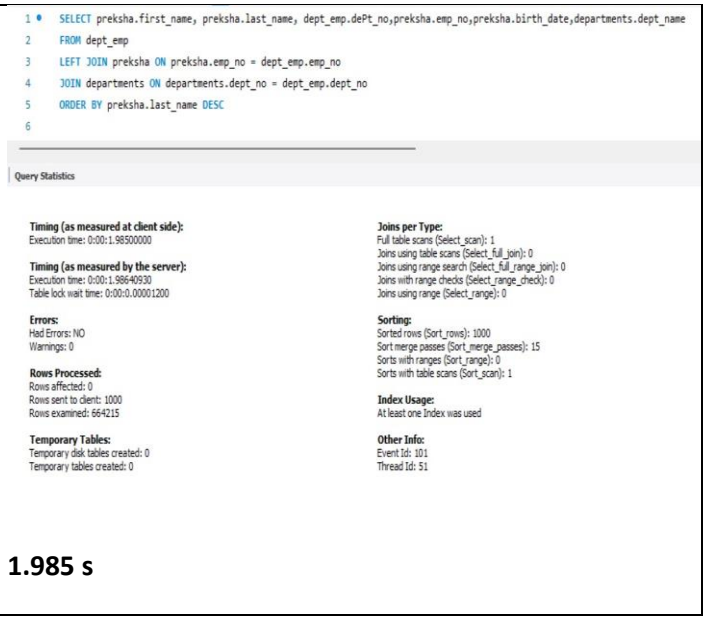
Simulation of queries where performed, firstly

**QUERY:**

**Renamed employee table as preksha**

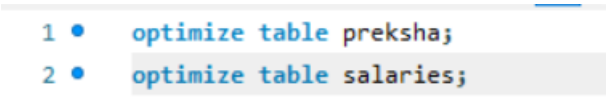
| | QUERY | TIME OF EXECUTION |
|---|---|---|
| **SELECT AND WHERE QUERY** | **select first_name from preksha;**<br>**SELECT first_name,last_name FROM preksha WHERE first_name LIKE 'P%';** | **0.015 s**<br>1 • select first_name from preksha;<br>2 • SELECT first_name,last_name<br>3 FROM preksha<br>4 WHERE first_name LIKE 'P%';<br>Query Statistics<br>Timing (as measured at client side):<br>Execution time: 0:00:0.01500000<br>Query stats can only be fetched when a single statement is executed. |

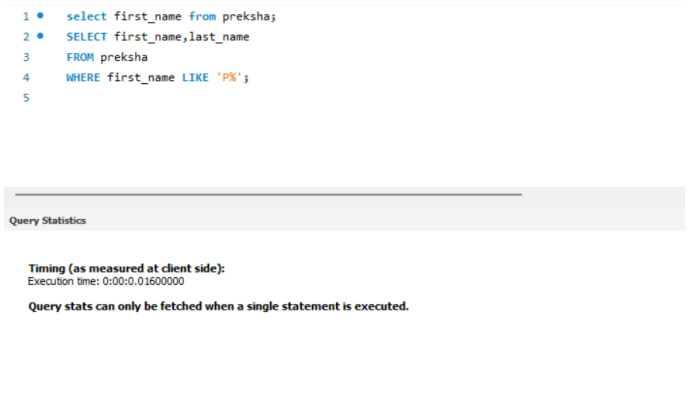
| | | |
|---|---|---|
| **AGGREGATE FUNCTION QUERY** | **SELECT AVG(salary) AS average_salary<br>FROM salaries;** |  <br>**1.234s** |
| **ORDER BY** | **SELECT * FROM SALARIES ORDER BY SALARY ASC;** |  <br>**1.171 s** |
| **NESTED QUERY** | **SELECT emp_no, first_name, last_name, hire_date,birth_date<br>FROM preksha<br>WHERE emp_no IN (<br>SELECT emp_no<br>FROM salaries<br>WHERE hire_date > '1960-06-24'<br>AND salary > 35000<br>)<br>ORDER BY hire_date DESC** | **0.235s**<br>  |
| | | |

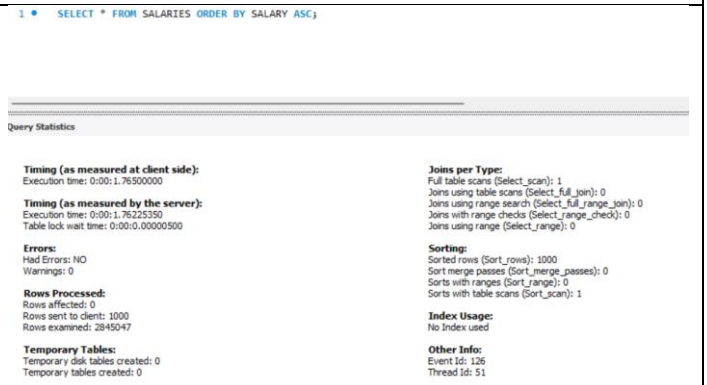
| | | |
|---|---|---|
| **JOIN QUERY** | **SELECT preksha.first_name, preksha.last_name, dept_emp.dePt_no,preksha.emp_no,preksha.birth_date,departments.dept_name FROM dept_emp LEFT JOIN preksha ON preksha.emp_no = dept_emp.emp_no JOIN departments ON departments.dept_no = dept_emp.dept_no ORDER BY preksha.last_name DESC** |  <br>**1.985 s** |

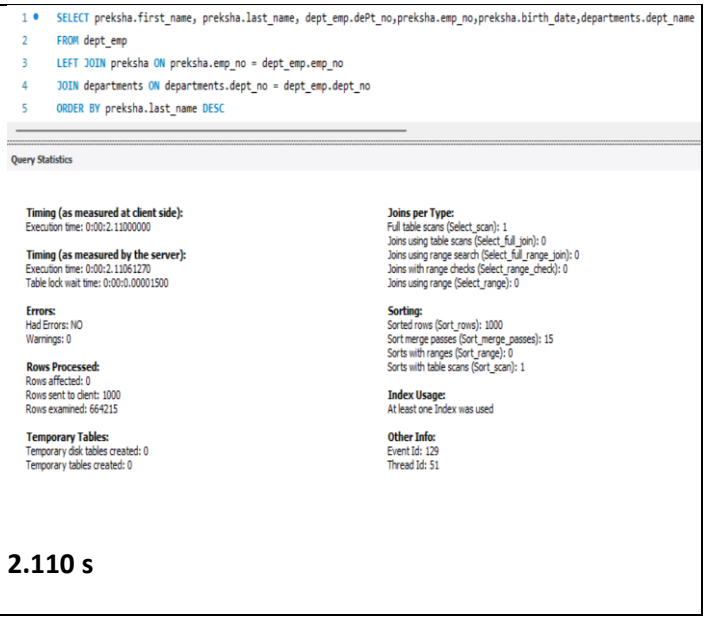
**OPTIMIZE TABLE**

**<u>COMMAND RUN</u>**

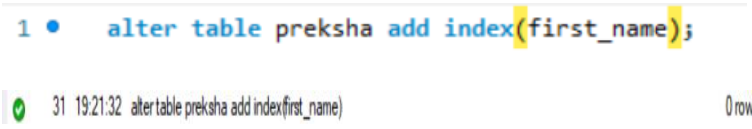


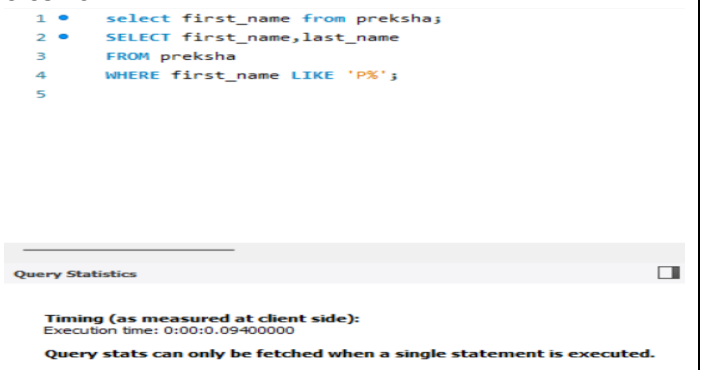

| | QUERY | TIME OF EXECUTION |
|---|---|---|
| **SELECT AND WHERE QUERY** | **select first_name from preksha;<br>SELECT first_name,last_name FROM preksha WHERE first_name LIKE 'P%';** | **0.016 s**<br>  |
| **AGGREGATE FUNCTION QUERY** | **SELECT AVG(salary) AS average_salary FROM salaries;** |  |

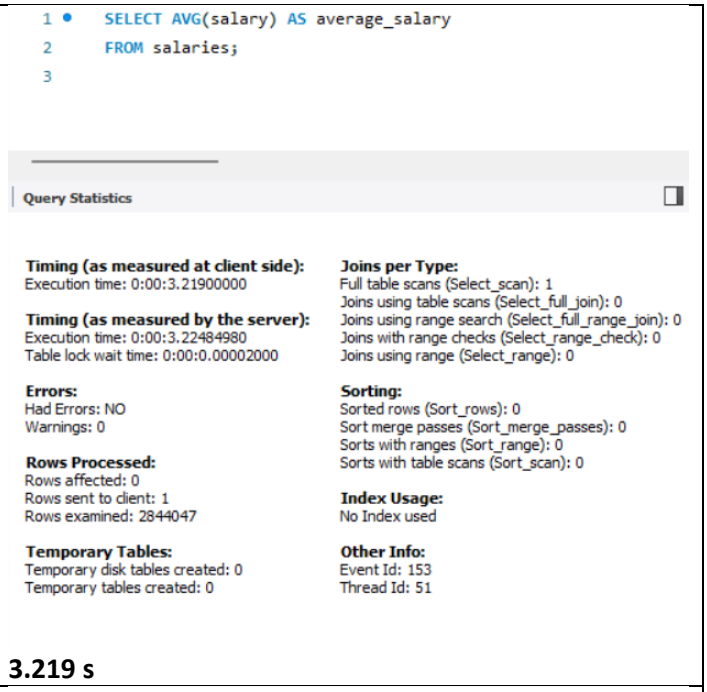
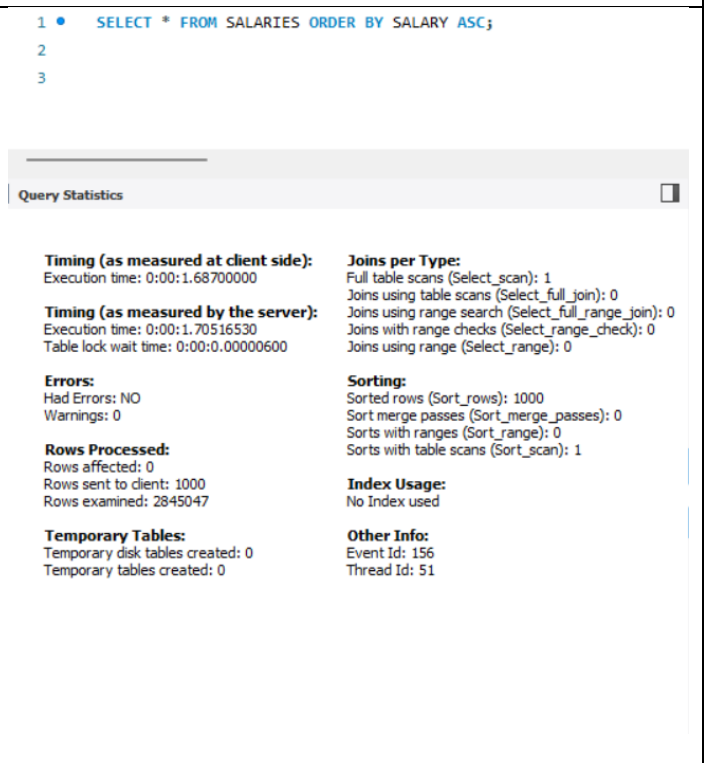
| | | |
|---|---|---|
| | | **2.891** |
| **ORDER BY** | **SELECT * FROM SALARIES ORDER BY SALARY ASC;** |  <br>**1.765 s** |
| **NESTED QUERY** | **SELECT emp_no, first_name, last_name, hire_date,birth_date FROM preksha WHERE emp_no IN ( SELECT emp_no FROM salaries WHERE hire_date > '1960-06-24' AND salary > 35000 ) ORDER BY hire_date DESC** | **0.921s**<br>  |
| | | |

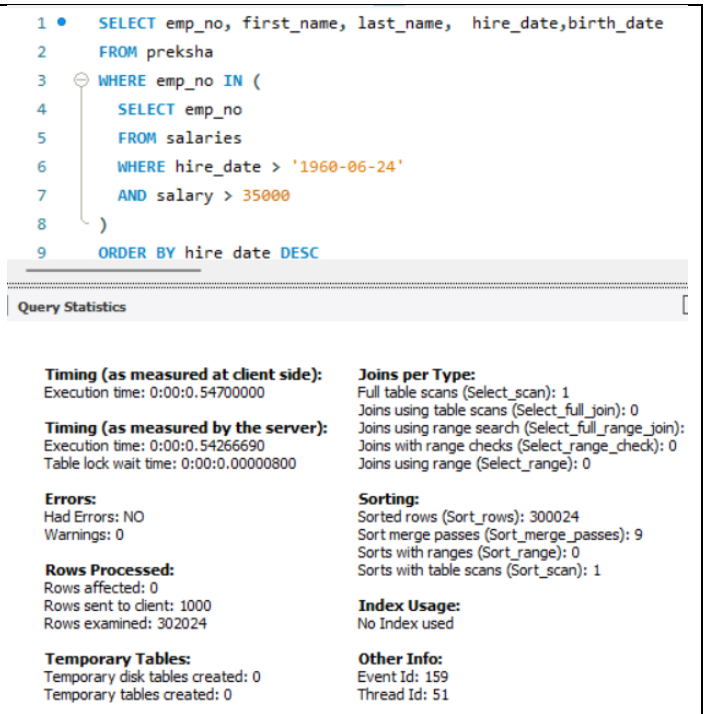
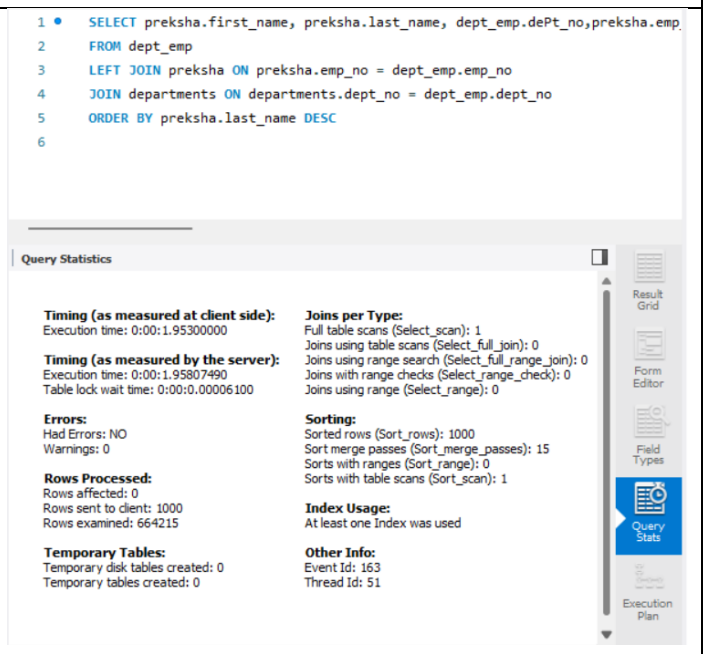
| | | |
|---|---|---|
| **JOIN QUERY** | **SELECT preksha.first_name, preksha.last_name, dept_emp.dePt_no,preksha.emp_no,preksha.birth_date,departments.dept_name FROM dept_emp LEFT JOIN preksha ON preksha.emp_no = dept_emp.emp_no JOIN departments ON departments.dept_no = dept_emp.dept_no ORDER BY preksha.last_name DESC** |  <br>**2.110 s** |

**INDEXING:**




| | QUERY | TIME OF EXECUTION |
|---|---|---|
| **SELECT AND WHERE QUERY** | **select first_name from preksha;<br>SELECT first_name,last_name FROM preksha WHERE first_name LIKE 'P%';** | **0.094 s**<br>  |

| | | |
|---|---|---|
| **AGGREGATE FUNCTION QUERY** | **SELECT AVG(salary) AS average_salary FROM salaries;** |  <br>**3.219 s** |
| **ORDER BY** | **SELECT * FROM SALARIES ORDER BY SALARY ASC;** |  <br>**1.687 s** |
| **NESTED QUERY** | **SELECT emp_no, first_name, last_name, hire_date,birth_date FROM preksha WHERE emp_no IN ( SELECT emp_no FROM salaries WHERE hire_date > '1960-06-24' AND salary > 35000** | **0.547s** |

| | | |
|---|---|---|
| | **)<br>ORDER BY hire_date DESC** |  |
| | | |
| **JOIN QUERY** | **SELECT preksha.first_name, preksha.last_name, dept_emp.dePt_no,preksha.emp_no,preksha.birth_date,departments.dept_name<br>FROM dept_emp<br>LEFT JOIN preksha ON preksha.emp_no = dept_emp.emp_no<br>JOIN departments ON departments.dept_no = dept_emp.dept_no<br>ORDER BY preksha.last_name DESC** |  <br>**1.953 s** |

**CONCLUSION:-** Thus stimulated query optimisation by applying SQL query on seleceted database of employees,by 3 ways unoptimized ,optimised and indexing query.

NAME:- PREKSHA ASHOK PATEL

BRANCH:-COMPUTER ENGINEERING

SAPID:-60004210126

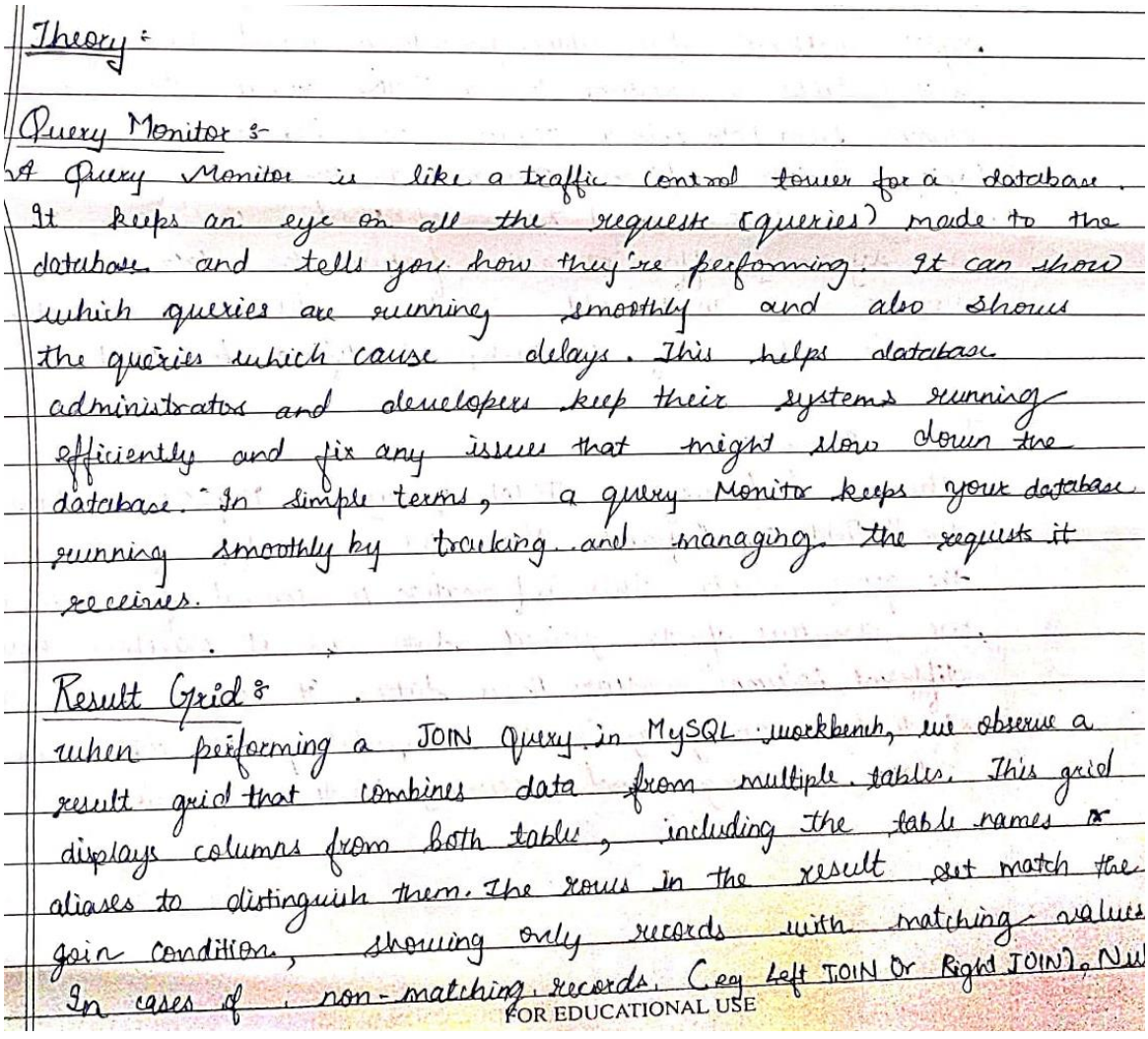
DIV:-C2 ; BATCH:- 1

# ADBMS

# EXPERIMENT-3

**AIM**:- Perform Query Monitor.

**THEORY :-**



Theory :-

Query Monitor :-

A Query Monitor is like a traffic control tower for a database. It keeps an eye on all the requests (queries) made to the database and tells you how they're performing. It can show which queries are running smoothly and also shows the queries which cause delays. This helps database administrators and developers keep their systems running efficiently and fix any issues that might slow down the database. In simple terms, a query Monitor keeps your database running smoothly by tracking and managing the requests it receives.

Result Grid :-

When performing a JOIN query in MySQL workbench, we observe a result grid that combines data from multiple tables. This grid displays columns from both tables, including the table names or aliases to distinguish them. The rows in the result set match the join condition, showing only records with matching values. In cases of non-matching records, (eg Left JOIN or Right JOIN), Nu

values may appear in columns from the non-matching table. The result grid provides a total row count and allows for data export and visualization.

Filtering in MySQL Workbench enables users to refine SQL query results, table data and object listings by specifying criteria to view only the specific information of interest.

Form Editor :

MySQL Workbench's form editor presents a unified view of data from multiple tables in response to a JOIN query. This view includes columns from both joined tables, each labeled for clarity. The displayed data is filtered according to the join condition, showing only records that meet the specified criteria. The form editor typically offers interactive features for data manipulation, allowing you to edit, insert or delete rows when we consider the ~~ref~~ relationships ~~fo~~ defined by JOIN

Field Types :

When performing a JOIN query in MySQL workbench, the "Field Types" panel shows the data types of the columns in the query result. This information is crucial for understanding the structure of the joined data, as it clarifies how different columns store their data. It assists users in effectively analyzing and manipulating the data, ensuring data consistency and accuracy in their queries.

Query Statistics :-

In MySQL Workbench, when performing a JOIN query, the "Query Statistics" panel typically provides essential performance metrics and information about the execution of the query. This includes details on the no. of rows examined, the time it took to complete the query, and any possible optimizations or indexes used. These statistics are valuable for assessing query effecieny and identifying potential areas for query optimization.

Execution Plan :-

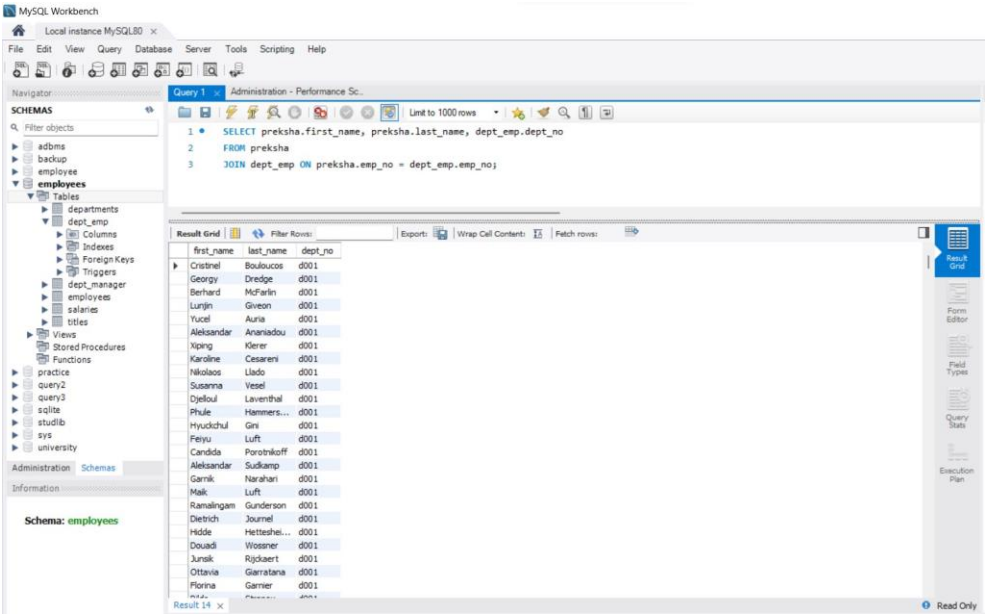
When performing a JOIN query in MySQL Workbench, the "Execution Plan" panel displays a detailed plan outlining how the database engine executes the query. It provides insights into the order in which tables are accessed and how the query is optimized for performance. By efficient JOIN methods, and optimize your queries for better performance.

**COMMAND**:-

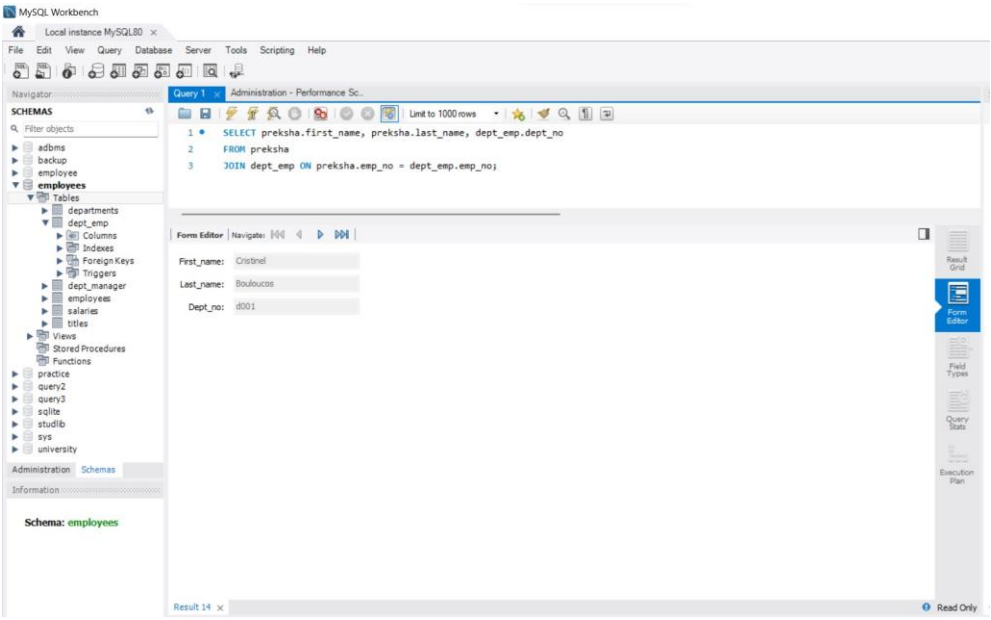
JOIN QUERY :-

```
SELECT preksha.first_name, preksha.last_name, dept_emp.dept_no
FROM preksha
JOIN dept_emp ON preksha.emp_no = dept_emp.emp_no;
```

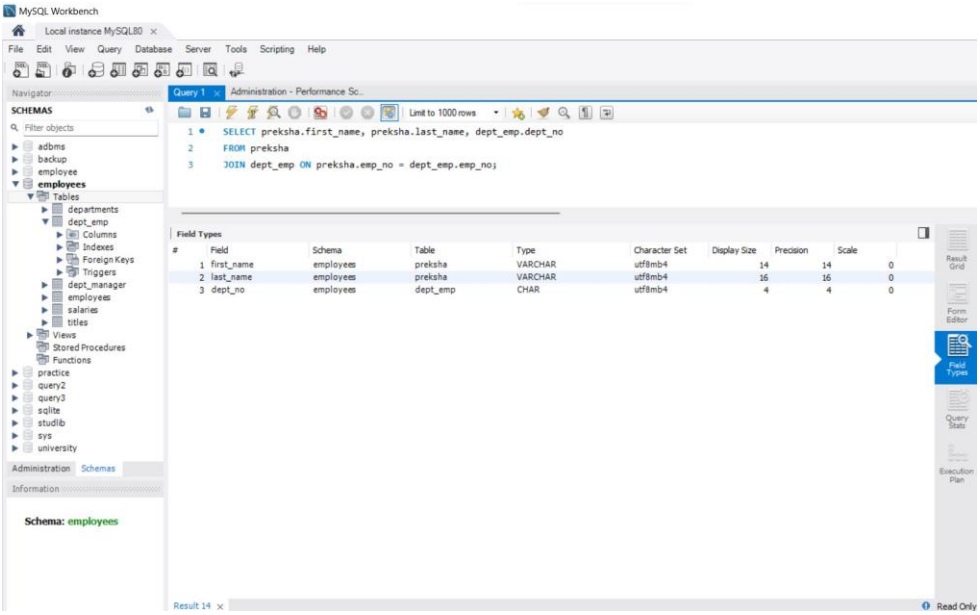
RESULT GRID:



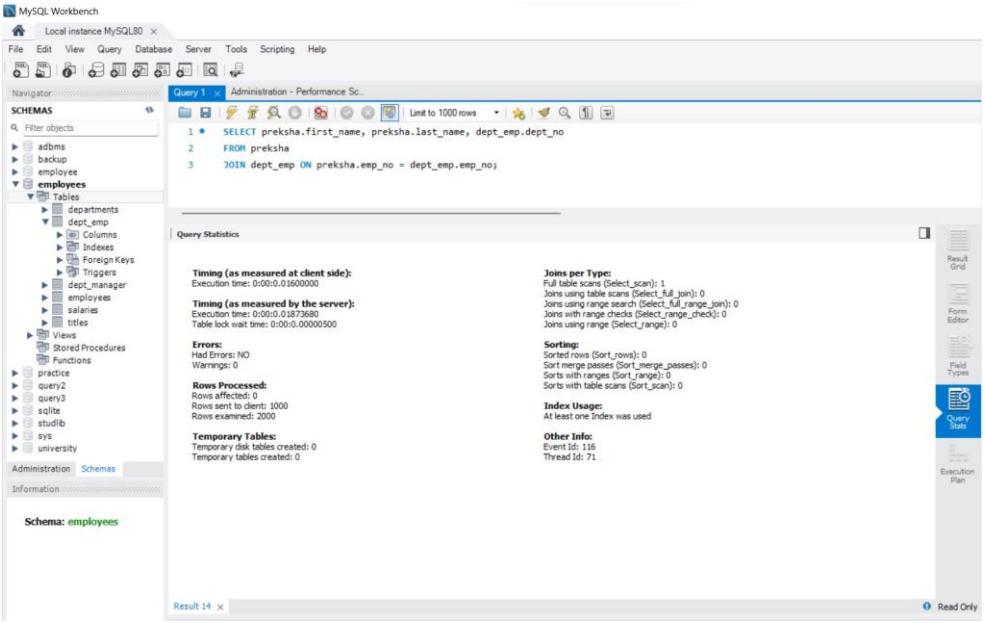

FORM EDITOR:



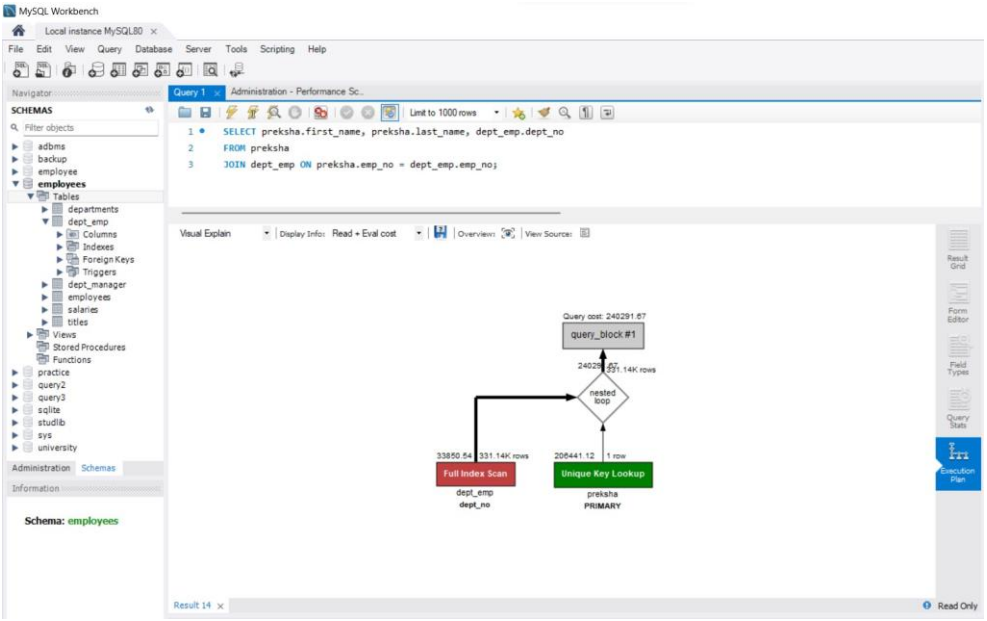

FIELD TYPES:




QUERY STATISTICS:




EXECUTION PLAN:




CONCLUSION:-

In conclusion,an experiment to perform a query Monitor provided valuable insight into how we can keep a close watch on the performance of our database queries. By using this tool, we know how to track,analyze and optimize our database interactions, ensuring our projects to run smoothly and efficiently.

NAME:-PREKSHA ASHOK PATEL

BRANCH:-COMPUTER ENGINEERING

SAPID:-60004210126

DIV:-C2 ; BATCH:-1

# ADBMS

# Experiment 4

**Aim**: Optimize using B and B+ trees.

## Theory:-

**B-Tree:**

A B-Tree (Balanced Tree) is a self-balancing tree data structure that maintains sorted data and allows searches, insertions, and deletions in logarithmic time. It is designed to keep data sorted and balanced, ensuring efficient operations even with large datasets. A B-Tree node contains multiple keys and child pointers, and it is structured in a way that maintains balance by redistributing keys among nodes during insertions and deletions. This balancing property helps in optimizing search operations, making B-Trees suitable for use in databases and file systems.

The main characteristics of a B-Tree include:

- Balanced Structure: All leaf nodes of the tree are at the same level, ensuring a balanced structure that guarantees logarithmic height.
- Sorted Data: Within each node, keys are stored in sorted order, allowing for efficient searching through a binary search-like mechanism.
- Node Capacity: Each node in a B-Tree has a maximum and minimum number of keys it can hold. When a node becomes full, it is split, and the median key is pushed up to the parent node.
- Child Pointers: Non-leaf nodes have child pointers corresponding to ranges of keys. These pointers guide the search process.

B-Trees are commonly used in databases to store and manage indices, allowing for rapid retrieval of records.

**B+ Tree:**

A B+ Tree is an extension of the B-Tree that optimizes certain aspects, especially for use in databases and file systems. The key differences lie in the structure of the nodes and the way keys are stored.

In a B+ Tree:

- Data Storage: Unlike B-Trees, B+ Trees store data only in the leaf nodes. Internal nodes contain only keys and child pointers, leading to a more efficient use of memory.

- Sequential Access: Leaf nodes are linked in a sequential manner, facilitating range queries and sequential access to the data.
- Search Operation: Search operations involve navigating down the tree through internal nodes and finally reaching a leaf node where the desired data is stored.
- Balanced Structure: Similar to B-Trees, B+ Trees maintain a balanced structure, ensuring logarithmic height and efficient operations.
- Key Range Queries: B+ Trees are particularly well-suited for range queries due to their sequential leaf node arrangement.

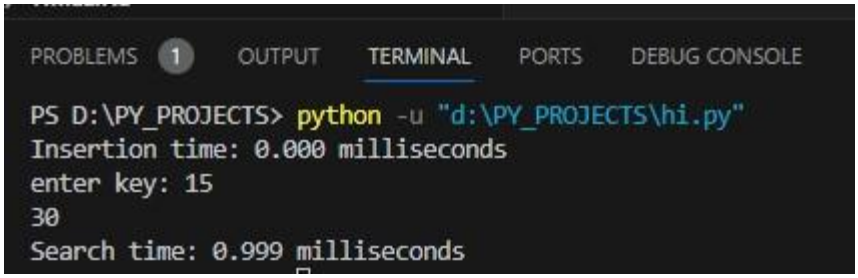
B+ Trees are commonly used in database systems to implement indices, providing efficient support for various query types, including equality searches and range queries.

**Code:**

## B Tree

```
from BTrees import IIBTree

import time

t = IIBTree()

insertion_start_time= time.time()

for i in range(1000):

t.update({i: 2*i})

insertion_end_time=time.time()

print("Insertion time:", format((insertion_end_time-insertion_start_time)*1000, ".3f"),

"milliseconds")

key = int(input("enter key: "))

search_start_time = time.time()

if t.has_key(key):

print(t[key])

search_end_time = time.time()

print("Search time:", format((search_end_time-search_start_time)*1000, ".3f"),

"milliseconds")
```

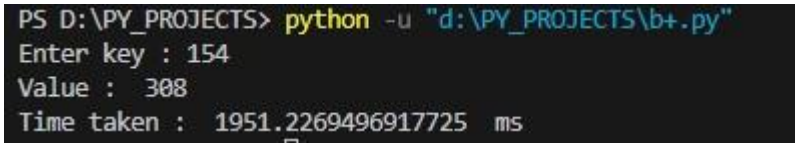
**Output:**



B+ Tree:

```
from bplustree import BPlusTree

import time

tree = BPlusTree('D:/PY_PROJECTS/tmp/bplustree.db',order=50)

for i in range(1000):

data=(2*i).to_bytes(10,'big')

tree[i]=data

start_time=time.time()

for i in range(5):

data=int(input("Enter key : "))

byte_data=tree.get(data)

int_data=int.from_bytes(byte_data, 'big')

print("Value : ",int_data)

end_time=time.time()

print("Time taken : ",(end_time-start_time)*1000," ms")

tree.close()
```

**Output**




**Observation:**

Runtime for a single search in B Tree was 0.999 ms while that in B+ Tree was 0.195 ms which proves that

B+ Tree optimize the query further B Trees.

**CONCLUSION:-**

The B-Tree and B+ Tree data structures are powerful tools for managing sorted data efficiently. B-Trees excel in scenarios where a balanced structure and sorted data are crucial, providing logarithmic time complexity for operations. On the other hand, B+ Trees optimize memory usage and enhance support for range queries, making them particularly well-suited for database applications. Both tree types play a significant role in improving the performance of search, insertion, and deletion operations, with B+ Trees offering specific advantages in certain database-related scenarios.

NAME:-PREKSHA ASHOK PATEL

SAPID:-60004210126

BRANCH:-COMPUTER ENGINEERING

DIV:-C2 ; BATCH:-1

# ADBMS

## EXPERIMENT-5

**AIM:-** Implement various types of fragmentation and partitioning (list,keys,hash and range)

**THEORY:**

Theory :-

Fragmentation :-

Fragmentation is the inefficient allocation of memory or storage, resulting in non-contiguous blocks that hinder resource utilization and system performance. It can be categorized as vertical, horizontal and mixed.

Vertical fragmentation :-

Vertical fragmentation is a database design technique where a single table is split into smaller, related tables by columns. Each fragmented table contains a subset of the original table's columns, with shared keys to link them together. This method can improve data retrieval efficiency and security by reducing the amount of redundant data accessed, especially in distributed databases or scenarios where certain users or applications should only access specific columns of the data.

Horizontal fragmentation :-

Horizontal fragmentation is a database design approach in which a single table is divided into smaller, related tables by rows. Each fragmented table contains a subset of the original table's rows, typically based on a defined condition or attribute value. This method is used to enhance data distribution and access control, especially in distributed database systems, as it allows for data partitioning based on specific criteria, improving performance

and ensuring that certain users or applications can only access relevant data subsets.

Mixed fragmentation :-

Mixed fragmentation is a database design technique that combines both horizontal and vertical fragmentation. In this approach, a single table is divided into smaller tables by both rows and columns, resulting in a more granular distribution of data. This allows for fine-grained control over data access and storage optimization, making it useful in scenarios where a combination of row-based & column-based segmentation is required to meet specific performance & security objectives in a distributed database environment.

Partitioning :-

Partitioning is a database management technique that involves dividing large tables into smaller, more manageable segments to improve performance & manageability. Four common types of partitioning, including key-based partitioning, are :-

Range Partitioning :-

Data is divided based on a specified range of values for a chosen column, such as dates or numerical values. This simplifies data maintenance and retrieval for a specific range.

List partitioning :-

Data is grouped into partitions based on predefined lists of values within a designated column. This method is useful when data can be categorized into discrete categories.

Hash partitioning :-

Data is distributed across partitions using a hash function, which aims to evenly distribute rows across partitions. It's beneficial for load balancing in distributed systems.

Key partitioning :-

It is similar to hash partitioning where the hash partitioning uses user-specified expression & mysql server supplied hashing functions for key. If we use internal hashing function that is performed by using partitioning by key clause

**How can we partition the table in MySQL?**

We can create a partition in MySQL using the CREATE TABLE or ALTER TABLE statement. Below is the syntax of creating partition using CREATE TABLE command:

```
CREATE TABLE [IF NOT EXISTS] table_name
(column_definitions)
[table_options]
[partition_options]
```

The below is the syntax of creating partition using ALTER TABLE command:

```
ALTER TABLE [IF EXISTS] tab_name
(colm_definitions)
[tab_options]
[partition_options]
```

## Types of MySQL Partitioning

MySQL has mainly six types of partitioning, which are given below:

- RANGE Partitioning
- LIST Partitioning
- COLUMNS Partitioning
- HASH Partitioning
- KEY Partitioning
- Subpartitioning

### MySQL RANGE Partitioning

This partitioning allows us to partition the rows of a table based on column values that fall within a specified range. The given range is always in a contiguous form but should not overlap each other, and also uses the VALUES LESS THAN operator to define the ranges.

CODE:

```
CREATE TABLE Sales ( cust_id INT NOT NULL, name VARCHAR(40),
store_id VARCHAR(20) NOT NULL, bill_no INT NOT NULL,
bill_date DATE PRIMARY KEY NOT NULL, amount DECIMAL(8,2) NOT
NULL)
PARTITION BY RANGE (year(bill_date))(
PARTITION p0 VALUES LESS THAN (2016),
```

```
PARTITION p1 VALUES LESS THAN (2017),
PARTITION p2 VALUES LESS THAN (2018),
PARTITION p3 VALUES LESS THAN (2020));

INSERT INTO Sales VALUES
(1, 'Mike', 'S001', 101, '2015-01-02', 125.56),
(2, 'Robert', 'S003', 103, '2015-01-25', 476.50),
(3, 'Peter', 'S012', 122, '2016-02-15', 335.00),
(4, 'Joseph', 'S345', 121, '2016-03-26', 787.00),
(5, 'Harry', 'S234', 132, '2017-04-19', 678.00),
(6, 'Stephen', 'S743', 111, '2017-05-31', 864.00),
(7, 'Jacson', 'S234', 115, '2018-06-11', 762.00),
(8, 'Smith', 'S012', 125, '2019-07-24', 300.00),
(9, 'Adam', 'S456', 119, '2019-08-02', 492.20);

SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH,
DATA_LENGTH
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_SCHEMA = 'partitioning' AND TABLE_NAME = 'Sales';
```

OUTPUT:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
|---|---|---|---|---|
| sales | p0 | 2 | 8192 | 16384 |
| sales | p1 | 2 | 8192 | 16384 |
| sales | p2 | 2 | 8192 | 16384 |
| sales | p3 | 3 | 5461 | 16384 |

Query Statistics

**Timing (as measured at client side):**
Execution time: 0:00:0.01600000

**Timing (as measured by the server):**
Execution time: 0:00:0.00683610
Table lock wait time: 0:00:0.00396400

**Errors:**
Had Errors: NO
Warnings: 0

**Rows Processed:**
Rows affected: 0
Rows sent to client: 4
Rows examined: 7

**Temporary Tables:**
Temporary disk tables created: 0
Temporary tables created: 0

**Joins per Type:**
Full table scans (Select_scan): 1
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

**Sorting:**
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

**Index Usage:**
At least one Index was used

**Other Info:**
Event Id: 160
Thread Id: 49

**MySQL LIST Partitioning**

It is the same as Range Partitioning. Here, the partition is defined and selected based on columns matching one of a set of discrete value lists rather than a set of a contiguous range of values. It is performed by the PARTITION BY LIST(exp) clause. The exp is an expression or column value that returns an integer value. The VALUES IN(value_lists) statement will be used to define each partition.

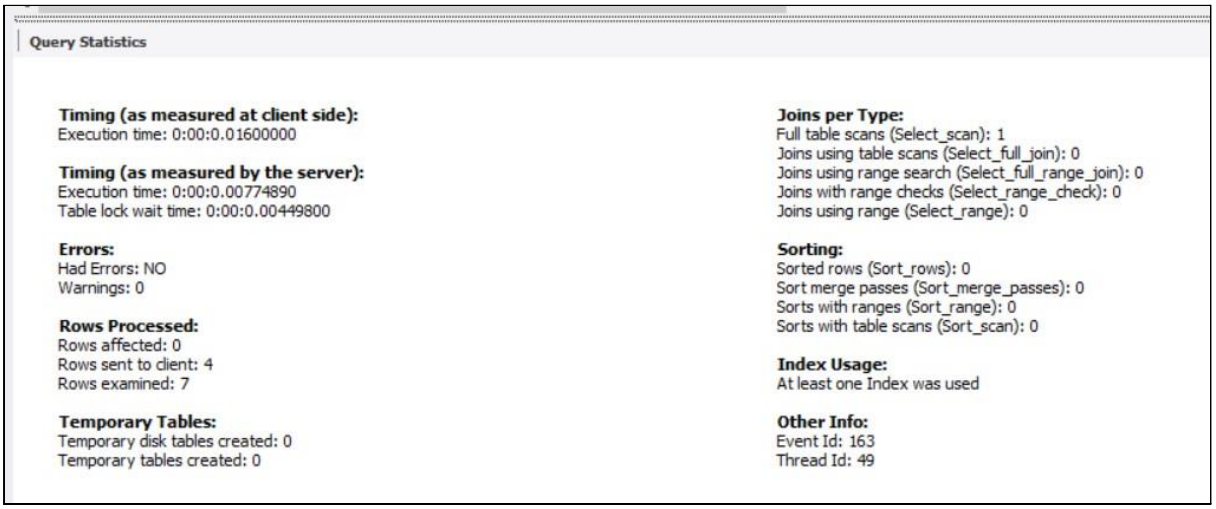
CODE:

```
CREATE TABLE Stores ( cust_name
    VARCHAR(40), bill_no VARCHAR(20)
    NOT NULL, store_id INT PRIMARY
    KEY NOT NULL, bill_date DATE NOT
    NULL, amount DECIMAL(8,2) NOT
    NULL
)
PARTITION BY LIST(store_id) (
PARTITION pEast VALUES IN (101, 103, 105),
PARTITION pWest VALUES IN (102, 104, 106),
PARTITION pNorth VALUES IN (107, 109, 111),
PARTITION pSouth VALUES IN (108, 110, 112));

INSERT INTO Stores VALUES
("Mike", "1", 101, "2015-01-25", 100.00),
("Joseph", "2", 102, "2015-01-25", 100.00),
("Robert", "3", 103, "2015-01-25", 100.00),
("Peter", "4", 104, "2015-01-25", 100.00),
("Joseph", "5", 105, "2015-01-25",100.00),
("Harry", "6", 106, "2015-01-25", 100.00),
("Jacson", "7", 107, "2015-01-25", 100.00),
("Smith", "8", 108, "2015-01-25", 100.00),
("Adam", "9", 110, "2015-01-25", 100.00);

SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH,
DATA_LENGTH
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_SCHEMA = 'partitioning' AND TABLE_NAME = 'Stores';
```

OUTPUT:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
|---|---|---|---|---|
| stores | pEast | 3 | 5461 | 16384 |
| stores | pNorth | 1 | 16384 | 16384 |
| stores | pSouth | 2 | 8192 | 16384 |
| stores | pWest | 3 | 5461 | 16384 |



Query Statistics

**Timing (as measured at client side):**
Execution time: 0:00:0.01600000

**Timing (as measured by the server):**
Execution time: 0:00:0.00774890
Table lock wait time: 0:00:0.00449800

**Errors:**
Had Errors: NO
Warnings: 0

**Rows Processed:**
Rows affected: 0
Rows sent to client: 4
Rows examined: 7

**Temporary Tables:**
Temporary disk tables created: 0
Temporary tables created: 0

**Joins per Type:**
Full table scans (Select_scan): 1
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

**Sorting:**
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

**Index Usage:**
At least one Index was used

**Other Info:**
Event Id: 163
Thread Id: 49

---

### MySQL HASH Partitioning

This partitioning is used to distribute data based on a predefined number of partitions. In other words, it splits the table as of the value returned by the user-defined expression. It is mainly used to distribute data evenly into the partition. It is performed with the PARTITION BY HASH(expr) clause. Here, we can specify a column value based on the column_name to be hashed and the number of partitions into which the table is divided.

CODE:

```
CREATE TABLE Stores2 ( cust_name VARCHAR(40), bill_no
    VARCHAR(20) NOT NULL, store_id INT PRIMARY KEY NOT NULL,
    bill_date DATE NOT NULL, amount DECIMAL(8,2) NOT NULL
)
PARTITION BY HASH(store_id)
PARTITIONS 4;

INSERT INTO Stores2 VALUES
```

```
("Mike", "1", 101, "2015-01-25", 100.00),
("Joseph", "2", 102, "2015-01-25", 100.00),
("Robert", "3", 103, "2015-01-25", 100.00),
("Peter", "4", 104, "2015-01-25", 100.00),
("Joseph", "5", 105, "2015-01-25",100.00),
("Harry", "6", 106, "2015-01-25", 100.00),
("Jacson", "7", 107, "2015-01-25", 100.00),
("Smith", "8", 108, "2015-01-25", 100.00),
("Adam", "9", 110, "2015-01-25", 100.00);

SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH,
DATA_LENGTH
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_SCHEMA = 'partitioning' AND TABLE_NAME = 'Stores2';
```

OUTPUT:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
|---|---|---|---|---|
| stores2 | p0 | 2 | 8192 | 16384 |
| stores2 | p1 | 2 | 8192 | 16384 |
| stores2 | p2 | 3 | 5461 | 16384 |
| stores2 | p3 | 2 | 8192 | 16384 |

Query Statistics

**Timing (as measured at client side):**
Execution time: 0:00:0.00000000

**Timing (as measured by the server):**
Execution time: 0:00:0.00489300
Table lock wait time: 0:00:0.00344600

**Errors:**
Had Errors: NO
Warnings: 0

**Rows Processed:**
Rows affected: 0
Rows sent to client: 4
Rows examined: 7

**Temporary Tables:**
Temporary disk tables created: 0
Temporary tables created: 0

**Joins per Type:**
Full table scans (Select_scan): 1
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

**Sorting:**
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

**Index Usage:**
At least one Index was used

**Other Info:**
Event Id: 187
Thread Id: 49

**MySQL COLUMN Partitioning**

This partitioning allows us to use the multiple columns in partitioning keys. The purpose of these columns is to place the rows in partitions and determine which partition will be validated for matching rows. It is mainly divided into two types:

**RANGE Columns Partitioning**

LIST Columns Partitioning

They provide supports for the use of non-integer columns to define the ranges or value lists. They support the following data types:

All Integer Types: TINYINT, SMALLINT, MEDIUMINT, INT (INTEGER), and BIGINT.
String Types: CHAR, VARCHAR, BINARY, and VARBINARY.
DATE and DATETIME data types.
Range Column Partitioning: It is similar to the range partitioning with one difference. It defines partitions using ranges based on various columns as partition keys. The defined ranges are of column types other than an integer type.

CODE:

```
CREATE TABLE test_part (A INT, B CHAR(5), C INT, D INT)
PARTITION BY RANGE COLUMNS(A, B, C)
(PARTITION p0 VALUES LESS THAN (50, 'test1', 100),
PARTITION p1 VALUES LESS THAN (100, 'test2', 200),
PARTITION p2 VALUES LESS THAN (150, 'test3', 300), PARTITION p3
VALUES LESS THAN (MAXVALUE, MAXVALUE, MAXVALUE));

INSERT INTO test_part VALUES
(10, 'a', 50, 3),
(30, 'test1', 150, 3),
(55, 'test1', 175, 3),
(123, 'test2', 233, 3),
(160, 'test4', 333, 3);

SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH,
DATA_LENGTH
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_SCHEMA = 'partitioning' AND TABLE_NAME = 'test_part';
```

OUTPUT:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
|---|---|---|---|---|
| test_part | p0 | 2 | 8192 | 16384 |
| test_part | p1 | 1 | 16384 | 16384 |
| test_part | p2 | 1 | 16384 | 16384 |
| test_part | p3 | 1 | 16384 | 16384 |


---

**LIST Columns Partitioning**

It takes a list of single or multiple columns as partition keys. It enables us to use various columns of types other than integer types as partitioning columns. In this partitioning, we can use String data types, DATE, and DATETIME columns.

CODE:

```
CREATE TABLE AgentDetail (
agent_id VARCHAR(10),
agent_name VARCHAR(40),
city VARCHAR(10))
PARTITION BY LIST COLUMNS(agent_id) (
PARTITION pNewyork VALUES IN('A1', 'A2', 'A3'),
PARTITION pTexas VALUES IN('B1', 'B2', 'B3'),
PARTITION pCalifornia VALUES IN ('C1', 'C2', 'C3'));

INSERT INTO AgentDetail VALUES
('A1', 'DummyName', 'CityName'),
('A2', 'DummyName', 'CityName'),
('A3', 'DummyName', 'CityName'),
```

```
('B1', 'DummyName', 'CityName'),
('B2', 'DummyName', 'CityName'),
('B3', 'DummyName', 'CityName'),
('C1', 'DummyName', 'CityName'),
('C2', 'DummyName', 'CityName'),
('C3', 'DummyName', 'CityName'),
('A1', 'DummyName', 'CityName'),
('C2', 'DummyName', 'CityName'),
('B1', 'DummyName', 'CityName');

SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH,
DATA_LENGTH
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_SCHEMA = 'partitioning' AND TABLE_NAME = 'AgentDetail';
```

OUTPUT:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
|---|---|---|---|---|
| agentdetail | pCalifornia | 4 | 4096 | 16384 |
| agentdetail | pNewyork | 4 | 4096 | 16384 |
| agentdetail | pTexas | 4 | 4096 | 16384 |

Query Statistics

**Timing (as measured at client side):**
Execution time: 0:00:0.01600000

**Timing (as measured by the server):**
Execution time: 0:00:0.01013060
Table lock wait time: 0:00:0.00426000

**Errors:**
Had Errors: NO
Warnings: 0

**Rows Processed:**
Rows affected: 0
Rows sent to client: 3
Rows examined: 6

**Temporary Tables:**
Temporary disk tables created: 0
Temporary tables created: 0

**Joins per Type:**
Full table scans (Select_scan): 1
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

**Sorting:**
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

**Index Usage:**
At least one Index was used

**Other Info:**
Event Id: 207
Thread Id: 49

---

**MySQL KEY Partitioning**

It is similar to the HASH partitioning where the hash partitioning uses the user-specified expression, and MySQL server supplied the hashing function for key. If we use other storage engines, the MySQL server employs its own internal hashing function that is performed by using the PARTITION BY KEY

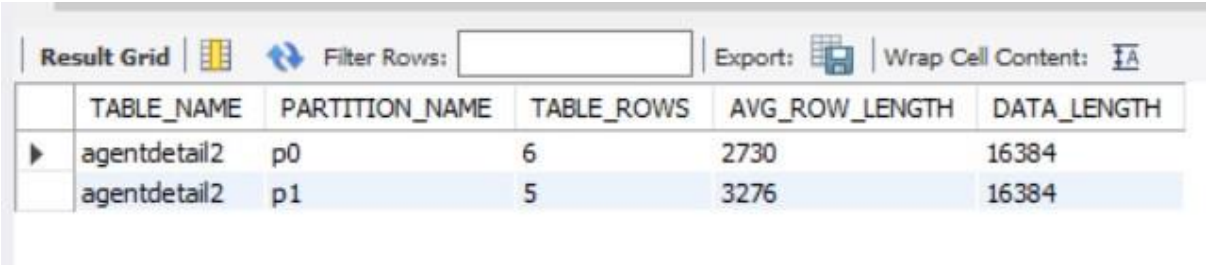
clause. Here, we will use KEY rather than HASH that can accept only a list of zero or more column names.

If the table contains a PRIMARY KEY and we have not specified any column for partition, then the primary key is used as partitioning key.

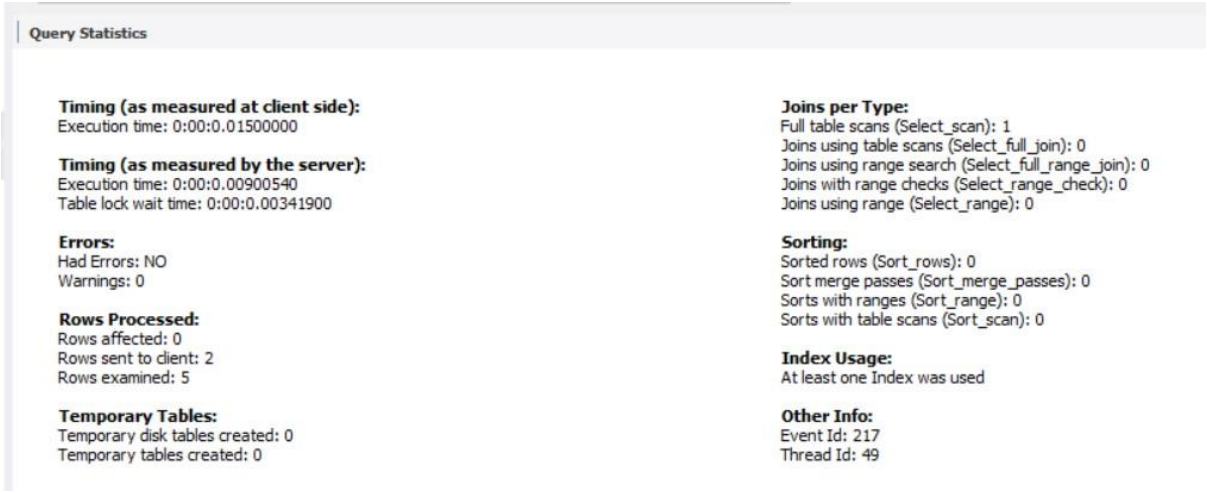
CODE:

```
CREATE TABLE AgentDetail2 ( agent_id
    INT NOT NULL PRIMARY KEY,
    agent_name VARCHAR(40)
)
PARTITION BY KEY()
PARTITIONS 2;

INSERT INTO AgentDetail2 VALUES
(1, "Name"),
(2, "Name"),
(3, "Name"),
(4, "Name"),
(5, "Name"),
(6, "Name"),
(7, "Name"),
(8, "Name"),
(9, "Name"),
(10, "Name"),
(11, "Name");

SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH,
DATA_LENGTH
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_SCHEMA = 'partitioning' AND TABLE_NAME = 'AgentDetail2';
```

OUTPUT:




| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
|---|---|---|---|---|
| agentdetail2 | p0 | 6 | 2730 | 16384 |
| agentdetail2 | p1 | 5 | 3276 | 16384 |




---

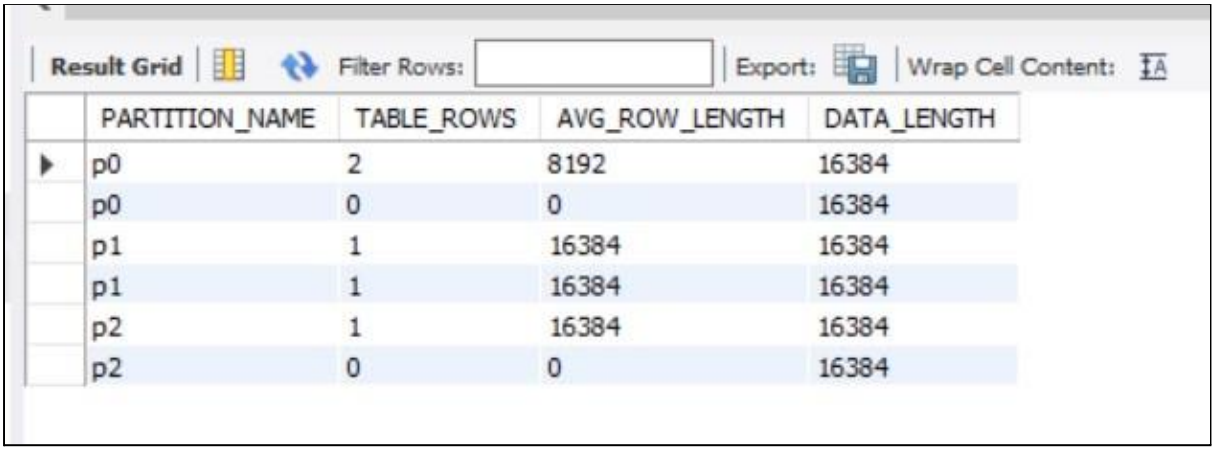
**SUBPARTITIONING**

It is a composite partitioning that further splits each partition in a partition table.

CODE:

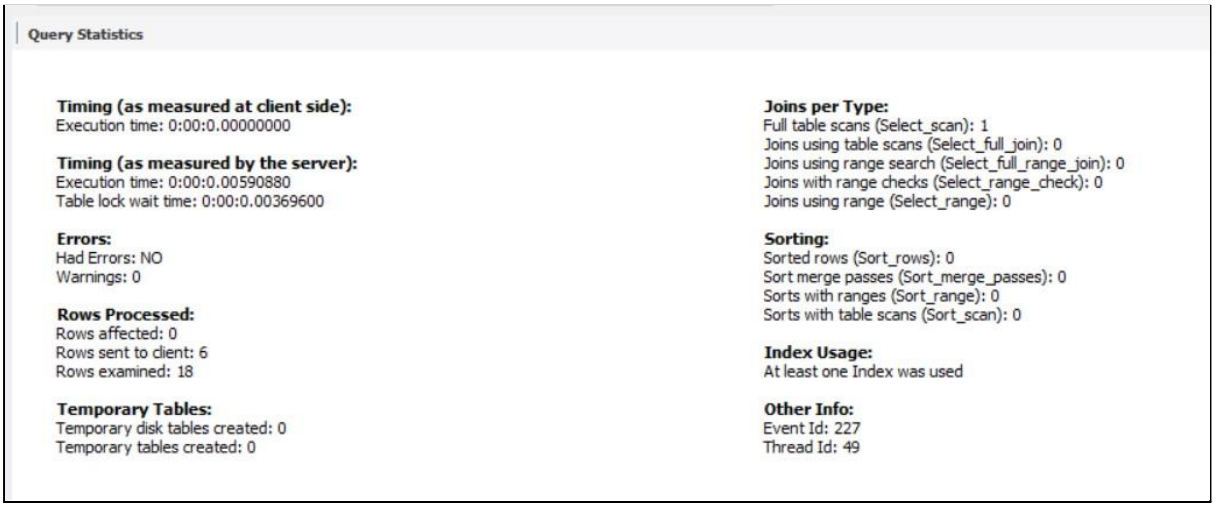
```
CREATE TABLE Person (
    id INT NOT NULL,
    name VARCHAR(40),
    purchased DATE,
    PRIMARY KEY (`id`, `purchased`)
)
 PARTITION BY RANGE( YEAR(purchased) )
    SUBPARTITION BY HASH( TO_DAYS(purchased) )
    SUBPARTITIONS 2 (
        PARTITION p0 VALUES LESS THAN (2015),
        PARTITION p1 VALUES LESS THAN (2020),
        PARTITION p2 VALUES LESS THAN MAXVALUE
    );

INSERT INTO Person VALUES
(1, "Name", "2013-01-13"),
(2, "Name2", "2014-04-22"),
(3, "Name3", "2015-02-25"),
(4, "Name4", "2018-05-05"),
(5, "Name5", "2020-02-04");

SELECT PARTITION_NAME, TABLE_ROWS
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_SCHEMA = 'partitioning' AND TABLE_NAME = 'Person';
```

OUTPUT:



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
|---|---|---|---|---|
| ▶ | p0 | 2 | 8192 | 16384 |
| | p0 | 0 | 0 | 16384 |
| | p1 | 1 | 16384 | 16384 |
| | p1 | 1 | 16384 | 16384 |
| | p2 | 1 | 16384 | 16384 |
| | p2 | 0 | 0 | 16384 |



Query Statistics

**Timing (as measured at client side):**
Execution time: 0:00:0.00000000

**Timing (as measured by the server):**
Execution time: 0:00:0.00590880
Table lock wait time: 0:00:0.00369600

**Errors:**
Had Errors: NO
Warnings: 0

**Rows Processed:**
Rows affected: 0
Rows sent to client: 6
Rows examined: 18

**Temporary Tables:**
Temporary disk tables created: 0
Temporary tables created: 0

**Joins per Type:**
Full table scans (Select_scan): 1
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

**Sorting:**
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

**Index Usage:**
At least one Index was used

**Other Info:**
Event Id: 227
Thread Id: 49

---

**Conclusion**: In conclusion, the implementation of diverse fragmentation and partitioning strategies, including list, key-based, hash, and range methods, serves to optimize data management in databases, enhancing retrieval efficiency and scalability. Tailoring these strategies to the nature of the data and anticipated query patterns is essential for achieving balanced workloads and maximizing the performance of database systems.

NAME:-PREKSHA ASHOK PATEL

SAPID:-60004210126

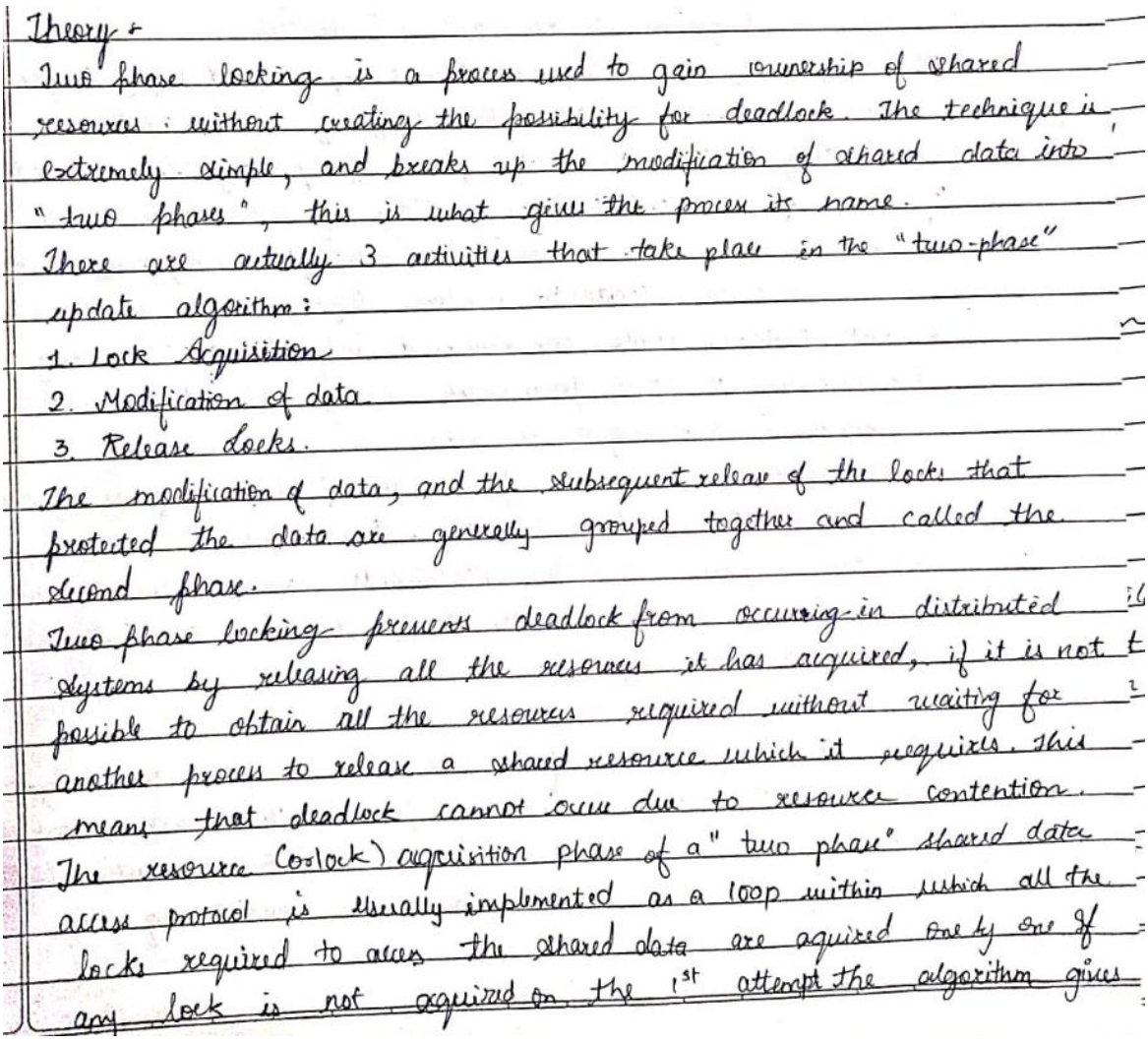
BRANCH:-COMPUTER ENGINEERING

DIV:-C2 ; BATCH:-1

# ADBMS EXPERIMENT-6

**AIM:- Implementation of 2 phase commit protocol in distributed system**

**THEORY:-**



Theory :-

Two phase locking is a process used to gain ownership of shared resources without creating the possibility for deadlock. The technique is extremely simple, and breaks up the modification of shared data into "two phases", this is what gives the process its name.

There are actually 3 activities that take place in the "two-phase" update algorithm :

1. Lock Acquisition
2. Modification of data
3. Release Locks.

The modification of data, and the subsequent release of the locks that protected the data are generally grouped together and called the second phase.

Two phase locking prevents deadlock from occuring in distributed systems by releasing all the resources it has acquired, if it is not possible to obtain all the resources required without waiting for another process to release a shared resource which it requires. This means that deadlock cannot occur due to resource contention.

The resource (Co)lock) acquisition phase of a "two phase" shared data access protocol is usually implemented as a loop within which all the locks required to access the shared data are aquired one by one. If any lock is not aquired on the 1st attempt the algorithm gives

up all the locks it had previously been able to get, and starts to try to get all the locks again.

Two phase commit :

The intention of all locking and transaction protocols is to produce an atomic update to distributed shared data, or to replicas of a shared data item. In the case of 2 phase commit algorithms for transactions on distributed databases, the intention is to prevent an update being carried out on only one of several replicas, since this would make the replicas inconsistent with each other. Therefore, the operation is either "successful on all replicas" or "aborted"

The only way that a 2 phase can be implemented safely is for each of the replicas to have some knowledge of where all the participating entities are in the process of updating the shared information. If each of the entities keeps a diary of what it has been doing then they can crash & re-start without becoming confused or allowing inconsistent data states to develop. Again 2 phases are used,

- Start Protocol, write commencement information into log, send transaction to other participants asking them to "commit" that transaction & requesting a response.
- Collect and log responses from other participants, if everyone is ready write "commit" to the log, then send a "commit message to all other participants. The participants write commit to their logs, commit the transaction locally, & send a "finished" message to the originator.

When collecting responses from the other participants the originator of a transaction cannot wait forever for all the other databases to respond, this will require a timeout to be implemented in the section of the protocol that initiates a commit request.

**IMPLEMENTATION:**

**COORDINATOR**

```
import java.io.IOException;
import
java.net.ServerSocket;
import java.net.Socket;
```

```
public class Coordinator {
   private static final int COORDINATOR_PORT = 9001;

   public static void main(String[] args) throws IOException {
 ServerSocket serverSocket = new ServerSocket(COORDINATOR_PORT);

      while (true) {
         Socket clientSocket = serverSocket.accept(); new
         CoordinatorThread(clientSocket).start();
      }
   }
}
```

**COORDINATOR.THREAD**

```
import java.io.BufferedReader; import
          java.io.IOException; import
java.io.InputStreamReader; import
java.io.PrintWriter;
import java.net.Socket;

public class CoordinatorThread extends Thread {
   private Socket socket;

   public CoordinatorThread(Socket socket) {
      this.socket = socket;
   }

   public void run() {
      try {
         BufferedReader in = new BufferedReader(new
         InputStreamReader(socket.getInputStream()));
         PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
```

```
            // Receive votes from participants
            String participantVote = in.readLine();
            System.out.println("Received vote from participant: " + participantVote);

            // Make a decision based on votes
            boolean commitDecision = decideCommitOrAbort(participantVote);

            // Inform participants about the decision
            out.println(commitDecision ? "COMMIT" : "ABORT");

            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private boolean decideCommitOrAbort(String participantVote) { // Simplified decision logic, can be enhanced based on requirements
        return participantVote.equals("VOTE_COMMIT");
    }
}
```

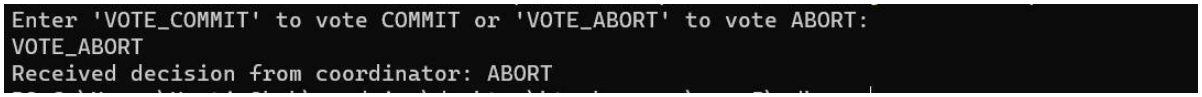
**PARTICIPANT**

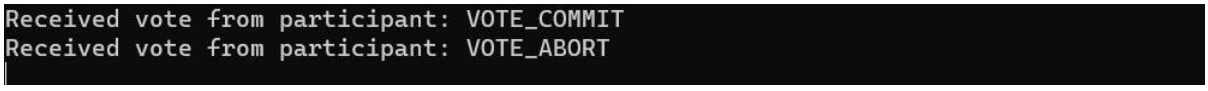
```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket; import
java.util.Scanner;

public class Participant { private static final int PARTICIPANT_PORT =
    8001; private static final
    int COORDINATOR_PORT = 9001;
```

```
    public static void main(String[] args) throws IOException {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter 'VOTE_COMMIT' to vote COMMIT or 'VOTE_ABORT' to vote ABORT:");

        String vote = scanner.nextLine();
        scanner.close();

        Socket socket = new Socket("localhost", COORDINATOR_PORT); PrintWriter
        out = new PrintWriter(socket.getOutputStream(),
        true); out.println(vote);

        BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        String decision = in.readLine();
    System.out.println("Received decision from coordinator: " + decision);

        socket.close();
    }
}
```

**OUTPUT:**





**CONCLUSION:-**

Conclusion :-

Therefore, using 2-Phase Commit (2PC) in advanced databases is like making sure everyone agrees before finalizing a big decision. It works in 2 steps: first, everyone involved says whether they're ready (Prepare phase) & then, if everyone's on board, the decision is made, or if someone has a problem, the plan is cancelled (Commit or Abort Phase). This helps keep things in order when dealing with complex tasks spread across different parts of a system. Although it's good for ensuring everyone is on same page, there can be some delays & challenges, but overall it's a crucial tool for making sure important tasks in distributed systems are completed correctly.

**NAME:-PREKSHA ASHOK PATEL**
**SAPID:-60004210126**
**BRANCH:COMPUTER ENGINEERING**
**DIV:-C2 ; BATCH:-1**

# ADBMS
# EXPERIMENT NO.-07

**AIM: Query execution on XML Database**

**THEORY:-**

Theory :-

XML database is used to store huge amount of information in XML format. As the use of XML is increasing in every field, it is required to have a secured place to store the XML document. The data stored in the database can be queried using XQuery, serialized & exported into a desired format.

They are 2 major types of XML database :-
1) XML enabled the extension provided for the conversion of XML document. It is relational database.
2) Native XML based on the container rather than the table format. It can be stored large amount of XML document & data.

XML stands for extensible markup language. It is similar to HTML & was designed to store & transport data. It was designed to be self descriptive. Its file extension is ".XML". It does not use predefined tags.

**IMPLEMENTATION:**

1. Creating an XML Table:

```
postgres=# create table users as select xml $$<users>
postgres$#     <user id="NoTechnician1">
postgres$#         <username>Star</username>
postgres$#         <email>star@gmail.com</email>
postgres$#         <duration>1y</duration>
postgres$#         <karma>60</karma>
postgres$#         <followers>127</followers>
postgres$#         <posts>20</posts>
postgres$#         <comments>9</comments>
postgres$#     </user>
postgres$#     <user id="NoTechnician2">
postgres$#         <username>Daisy</username>
postgres$#         <email>daisy@gmail.com</email>
postgres$#         <duration>2m</duration>
postgres$#         <karma>2</karma>
postgres$#         <followers>7</followers>
postgres$#         <posts>5</posts>
postgres$#     </user>
postgres$#     <user id="NoTechnician3">
postgres$#         <username>Lora</username>
postgres$#         <email>lora@gmail.com</email>
postgres$#         <duration>5y2m</duration>
postgres$#         <karma>700</karma>
postgres$#         <followers>555</followers>
postgres$#         <posts>70</posts>
postgres$#         <comments>89</comments>
postgres$#     </user>
postgres$#     <user id="NoTechnician4">
postgres$#         <username>Sam</username>
postgres$#         <email>sam@gmail.com</email>
postgres$#         <duration>6m</duration>
postgres$#         <karma>100</karma>
postgres$#         <followers>27</followers>
postgres$#         <posts>60</posts>
postgres$#     </user>
postgres$#     <user id="NoTechnician5">
postgres$#         <username>Roma</username>
postgres$#         <email>roma@gmail.com</email>
postgres$#         <duration>8y</duration>
postgres$#         <karma>500</karma>
postgres$#         <followers>327</followers>
postgres$#         <posts>80</posts>
postgres$#         <comments>90</comments>
postgres$#     </user>
postgres$#     <user id="NoTechnician6">
postgres$#         <username>Mike</username>
postgres$#         <email>mike@gmail.com</email>
postgres$#         <duration>1m</duration>
postgres$#         <karma>5</karma>
postgres$#         <followers>21</followers>
postgres$#         <posts>4</posts>
postgres$#     </user>
postgres$# </users>$$;
```

2. Selecting from Users Table:

```
postgres=# select * from users;
                  xml
-----------------------------------------
 <users>                                +
     <user id="NoTechnician1">          +
         <username>Star</username>      +
         <email>star@gmail.com</email>  +
         <duration>1y</duration>        +
         <karma>60</karma>              +
         <followers>127</followers>     +
         <posts>20</posts>              +
         <comments>9</comments>         +
     </user>                            +
     <user id="NoTechnician2">          +
         <username>Daisy</username>     +
         <email>daisy@gmail.com</email>+
         <duration>2m</duration>        +
         <karma>2</karma>               +
         <followers>7</followers>       +
         <posts>5</posts>               +
     </user>                            +
     <user id="NoTechnician3">          +
         <username>Lora</username>      +
         <email>lora@gmail.com</email>  +
         <duration>5y2m</duration>      +
         <karma>700</karma>             +
         <followers>555</followers>     +
         <posts>70</posts>              +
         <comments>89</comments>        +
     </user>                            +
     <user id="NoTechnician4">          +
         <username>Sam</username>       +
         <email>sam@gmail.com</email>   +
         <duration>6m</duration>        +
         <karma>100</karma>             +
         <followers>27</followers>      +
         <posts>60</posts>              +
     </user>                            +
     <user id="NoTechnician5">          +
         <username>Roma</username>      +
         <email>roma@gmail.com</email>  +
         <duration>8y</duration>        +
         <karma>500</karma>             +
         <followers>327</followers>     +
         <posts>80</posts>              +
         <comments>90</comments>        +
     </user>                            +
     <user id="NoTechnician6">          +
         <username>Mike</username>      +
         <email>mike@gmail.com</email>  +
         <duration>1m</duration>        +
         <karma>5</karma>               +
         <followers>21</followers>      +
         <posts>4</posts>               +
     </user>                            +
 </users>
(1 row)
```

3. Using xpath to extract data from XML

```
ostgres=# With tbl2(p_xml) as (SELECT
ostgres(# '<users>
ostgres'#     <user id="NoTechnician1">
ostgres'#         <username>Star</username>
ostgres'#         <email>star@gmail.com</email>
ostgres'#         <duration>1y</duration>
ostgres'#         <karma>60</karma>
ostgres'#         <followers>127</followers>
ostgres'#         <posts>20</posts>
ostgres'#         <comments>9</comments>
ostgres'#     </user>
ostgres'#     <user id="NoTechnician6">
ostgres'#         <username>Mike</username>
ostgres'#         <email>mike@gmail.com</email>
ostgres'#         <duration>1m</duration>
ostgres'#         <karma>5</karma>
ostgres'#         <followers>21</followers>
ostgres'#         <posts>4</posts>
ostgres'#     </user>
ostgres'# </users>'::xml)
ostgres-# select xpath('/users/user/username/text()',p_xml) from tbl2;
   xpath
------------
{Star,Mike}
1 row)
```

CONCLUSION:-

Conclusion :-

A book catalog, consisting of information about books was initially in an XML file, and using XML Request (), it was displayed in an HTML table. The data was further converted to JSON & then search queries were performed on the data. Separate data entities were represented in XML databases using XML tags -

**NAME:-PREKSHA ASHOK PATEL**
**SAPID:-60004210126**
**BRANCH:COMPUTER ENGINEERING**
**DIV:-C2 ; BATCH:-1**

# ADBMS
# EXPERIMENT NO.-08

**AIM:- Data Handling using Json .**

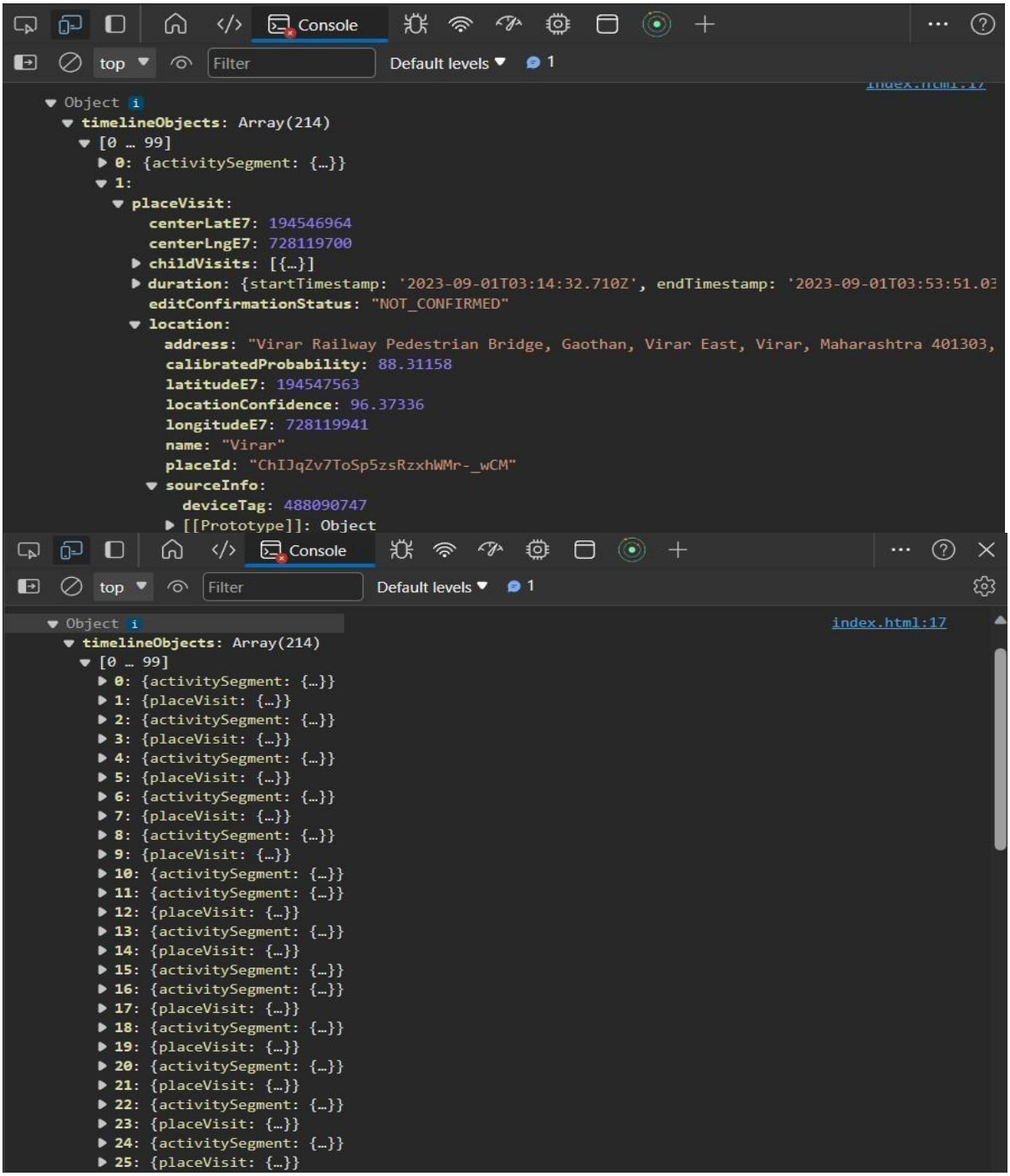
**THEORY:-**

Theory :-

JSON ( Javascript object notation), is a human readable data interchange format specified in early 2000's. Even though JSON is based on a subset of the Javascript Programming language standard, it is completely language independent. JSON objects are associative containers, where in a string key is mapped to a value. Almost any programming language has an implementation for this abstract data structure object in JS, dictionaries in Python, hash table in JAVA, & C#, associative arrays in C++ & so on JSON objects are easy for humans to understand and for machines to parse & generate.

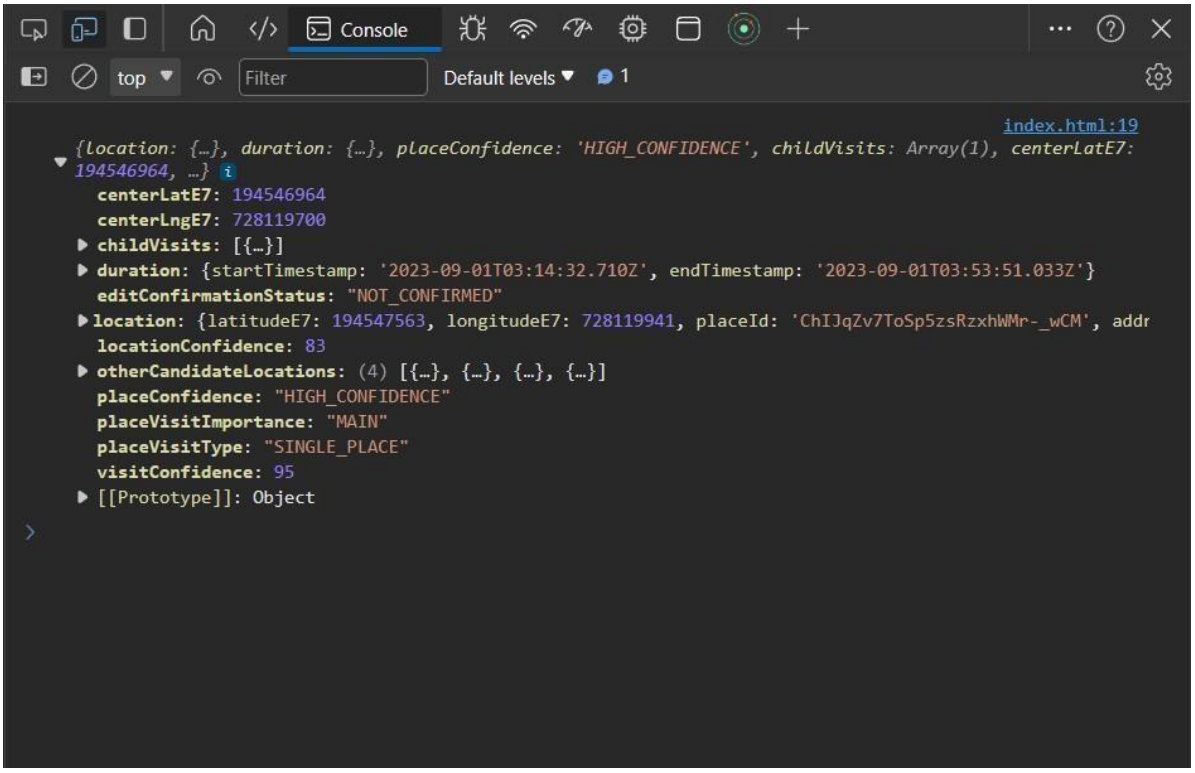
Handling JSON data :-
It is often observed that two terms Serialize & deserialize are associative with JSON objects with the context of working with JS & JSON in an nutshell to get JSON value from Javascript is Serialization & where is the other way, its deserialization. This is done by 2 methods, JSON stringify & JSON Parse

**IMPLEMENTATION:**



index.html

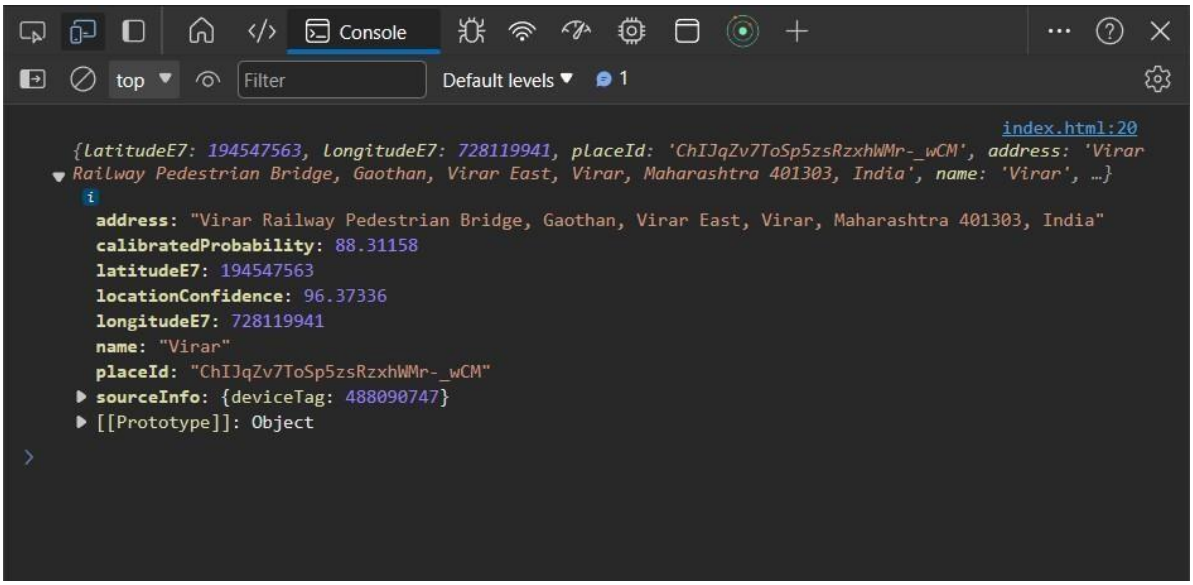
```
index.html > html > body > script > Func > then() callback > obj
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-width, initial-scale=1.0">
6       <title>Document</title>
7   </head>
8   <body>
9       <button onClick="Func()">Show Details</button>
10      <script>
11          function Func() {
12              var data_loc;
13              fetch(  (parameter) res: Response
14              .then(res => {
15                  return res.json();
16              })
17              .then((data) => {
18                  const obj = data.timelineObjects
19                  console.log(obj[1].placeVisit)
20              })
21          }
22      </script>
23  </body>
24  </html>
```

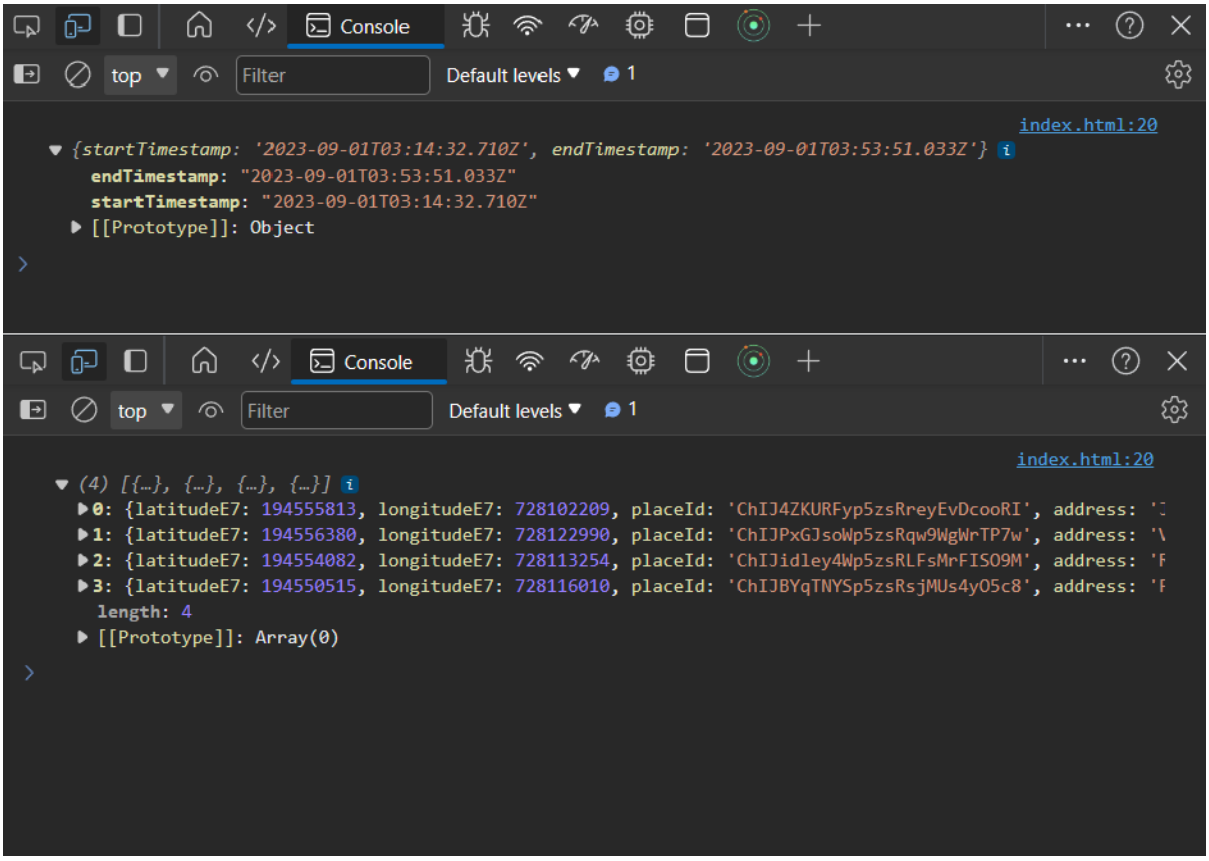


index.html

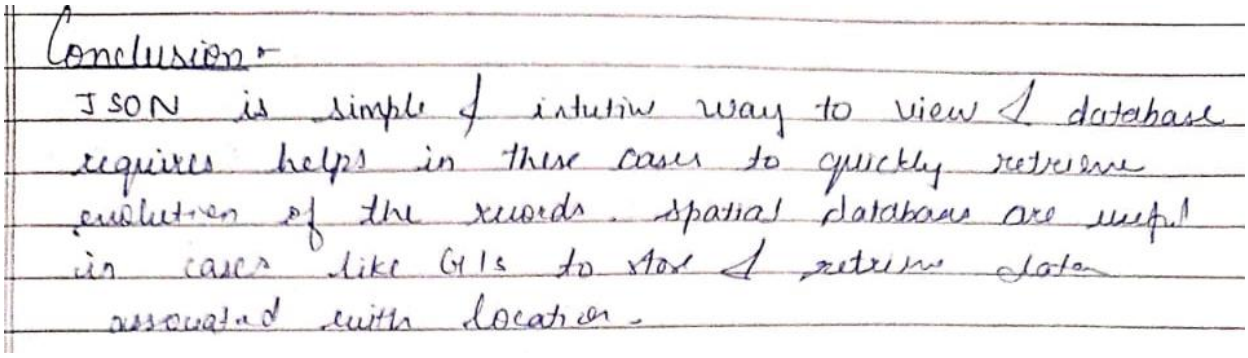
```
index.html > html > body > script > Func > then() callback
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <button onClick="Func()">Show Details</button>
    <script>
        function Func() {
            var data_loc;
            fetch("./2023_SEPTember.json")
            .then(res => {
                return res.json();
            })
            .then((data) => {
                const obj = data.timelineObjects
                const place = obj[1].placeVisit
                console.log(place.location)
            })
        }
    </script>
</body>
</html>
```




Console
top
Filter
Default levels
1
index.html:20
{startTimestamp: '2023-09-01T03:14:32.710Z', endTimestamp: '2023-09-01T03:53:51.033Z'}
endTimestamp: "2023-09-01T03:53:51.033Z"
startTimestamp: "2023-09-01T03:14:32.710Z"
[[Prototype]]: Object
Console
top
Filter
Default levels
1
index.html:20
(4) [{…}, {…}, {…}, {…}]
0: {latitudeE7: 194555813, longitudeE7: 728102209, placeId: 'ChIJ4ZKURFyp5zsRreyEvDcooRI', address:
1: {latitudeE7: 194556380, longitudeE7: 728122990, placeId: 'ChIJPxGJsoWp5zsRqw9WgWrTP7w', address:
2: {latitudeE7: 194554082, longitudeE7: 728113254, placeId: 'ChIJidley4Wp5zsRLFsMrFISO9M', address:
3: {latitudeE7: 194550515, longitudeE7: 728116010, placeId: 'ChIJBYqTNYSp5zsRsjMUs4yO5c8', address:
length: 4
[[Prototype]]: Array(0)

CONCLUSION:-


Conclusion:-
JSON is simple & intuitive way to view & database
requires helps in these cases to quickly retrieve
evaluation of the records. Spatial databases are useful
in cases like GIS to store & retrieve data
associated with location.

NAME:-PREKSHA ASHOK PATEL

SAPID:-60004210126

BRANCH:-COMPUTER ENGINEERING

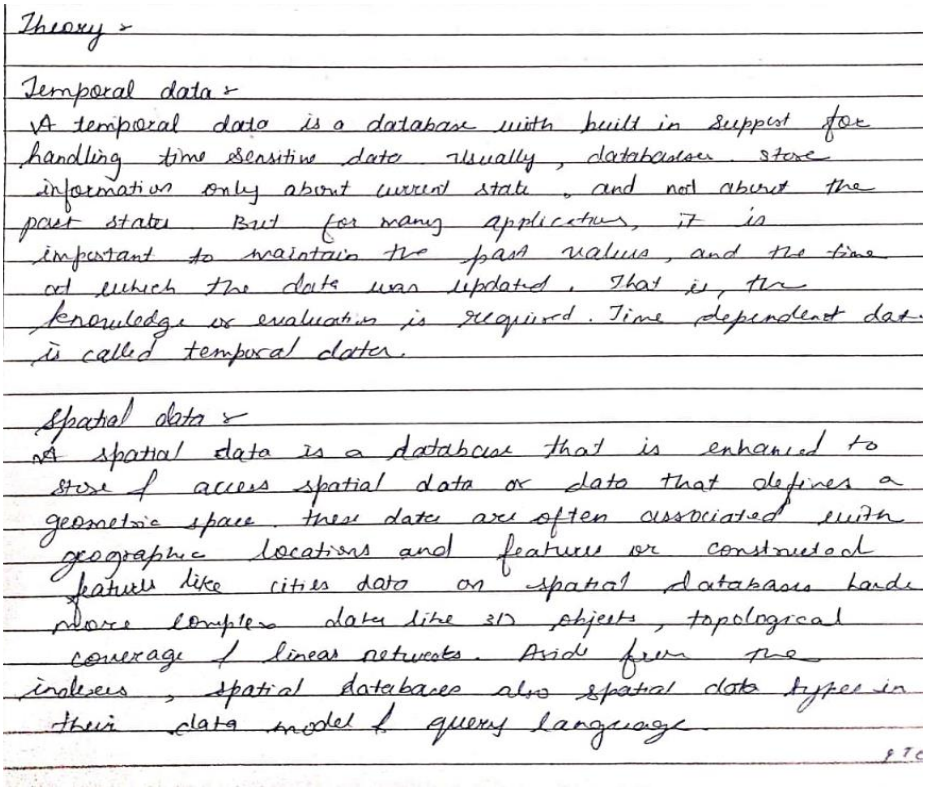
DIV:-C2 ; BATCH:-1

# ADBMS

## EXPERIMENT NO.09

**AIM: Procesing spatial and temporal data**

**THEORY:-**

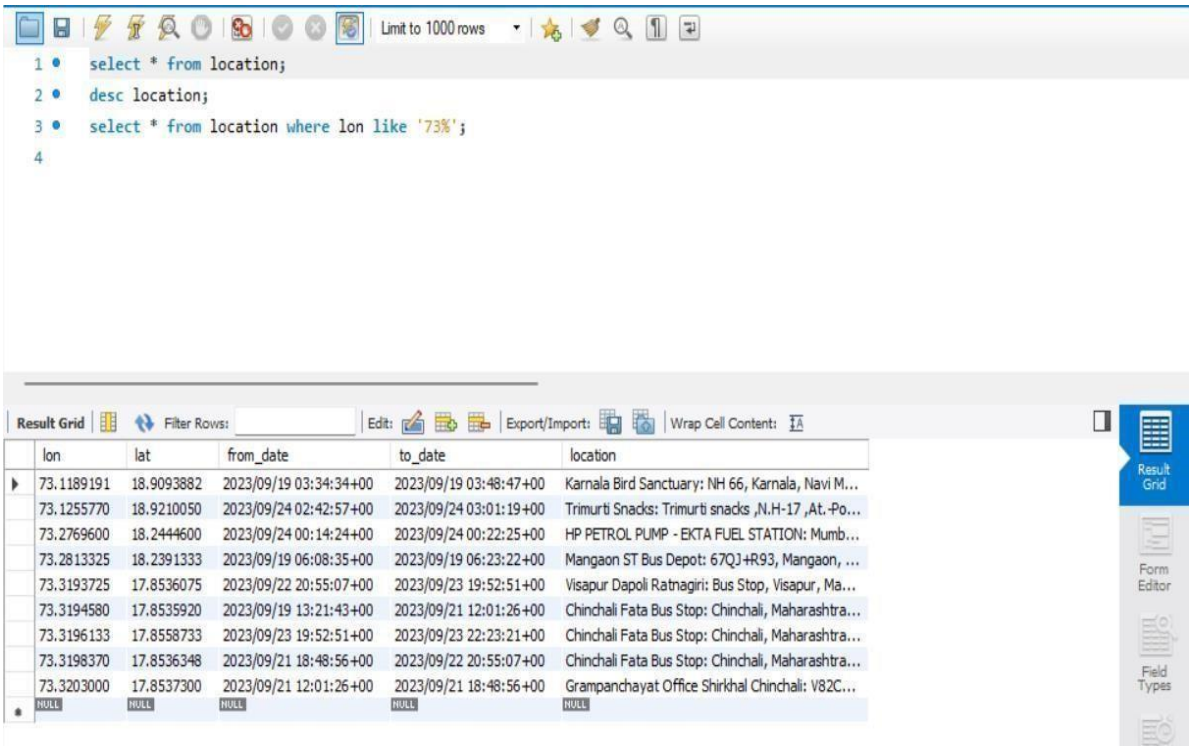


Theory :-

Temporal data :-
A temporal data is a database with built in support for handling time sensitive data. Usually, databases store information only about current state, and not about the past states. But for many applications, it is important to maintain the past values, and the time at which the data was updated. That is, the knowledge or evaluation is required. Time dependent data is called temporal data.

Spatial data :-
A spatial data is a database that is enhanced to store & access spatial data or data that defines a geometric space. These data are often associated with geographic locations and features or constructed features like cities data on spatial database handle more complex data like 3D objects, topological coverage & linear networks. Aside from the indexes, spatial databases also spatial data types in their data model & query language.

**IMPLEMENTATION:**



```
1 • select * from location;
2 • desc location;
3 • select * from location where lon like '73%';
4
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| lon | lat | from_date | to_date | location |
|---|---|---|---|---|
| 73.1189191 | 18.9093882 | 2023/09/19 03:34:34+00 | 2023/09/19 03:48:47+00 | Karnala Bird Sanctuary: NH 66, Karnala, Navi M... |
| 73.1255770 | 18.9210050 | 2023/09/24 02:42:57+00 | 2023/09/24 03:01:19+00 | Trimurti Snacks: Trimurti snacks ,N.H-17 ,At.-Po... |
| 73.2769600 | 18.2444600 | 2023/09/24 00:14:24+00 | 2023/09/24 00:22:25+00 | HP PETROL PUMP - EKTA FUEL STATION: Mumb... |
| 73.2813325 | 18.2391333 | 2023/09/19 06:08:35+00 | 2023/09/19 06:23:22+00 | Mangaon ST Bus Depot: 67QJ+R93, Mangaon, ... |
| 73.3193725 | 17.8536075 | 2023/09/22 20:55:07+00 | 2023/09/23 19:52:51+00 | Visapur Dapoli Ratnagiri: Bus Stop, Visapur, Ma... |
| 73.3194580 | 17.8535920 | 2023/09/19 13:21:43+00 | 2023/09/21 12:01:26+00 | Chinchali Fata Bus Stop: Chinchali, Maharashtra... |
| 73.3196133 | 17.8558733 | 2023/09/23 19:52:51+00 | 2023/09/23 22:23:21+00 | Chinchali Fata Bus Stop: Chinchali, Maharashtra... |
| 73.3198370 | 17.8536348 | 2023/09/21 18:48:56+00 | 2023/09/22 20:55:07+00 | Chinchali Fata Bus Stop: Chinchali, Maharashtra... |
| 73.3203000 | 17.8537300 | 2023/09/21 12:01:26+00 | 2023/09/21 18:48:56+00 | Grampanchayat Office Shirkhal Chinchali: V82C... |
| NULL | NULL | NULL | NULL | NULL |

Result Grid | Form Editor | Field Types

```
1 • select * from location;
2 • desc location;
3 • select * from location where lon like '73%';
4 • select * from location where location = "SVKM's Dwarkadas J. Sanghvi College of Engineering: No. U, 15, Bhaktive
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| lon | lat | from_date | to_date | location |
|---|---|---|---|---|
| 72.8370532 | 19.1070402 | 2023/09/10 03:23:29+00 | 2023/09/10 08:20:09+00 | SVKM's Dwarkadas J. Sanghvi College of Engine... |
| 72.8370757 | 19.1070180 | 2023/09/12 05:03:11+00 | 2023/09/12 11:50:38+00 | SVKM's Dwarkadas J. Sanghvi College of Engine... |
| 72.8370772 | 19.1071268 | 2023/09/10 08:58:37+00 | 2023/09/10 11:45:30+00 | SVKM's Dwarkadas J. Sanghvi College of Engine... |
| 72.8370886 | 19.1071036 | 2023/09/11 02:33:29+00 | 2023/09/11 10:39:31+00 | SVKM's Dwarkadas J. Sanghvi College of Engine... |
| 72.8370928 | 19.1071473 | 2023/09/25 04:53:23+00 | 2023/09/25 11:32:39+00 | SVKM's Dwarkadas J. Sanghvi College of Engine... |
| 72.8371343 | 19.1071759 | 2023/09/05 04:56:31+00 | 2023/09/05 11:36:28+00 | SVKM's Dwarkadas J. Sanghvi College of Engine... |
| 72.8371424 | 19.1071852 | 2023/09/15 02:31:55+00 | 2023/09/15 07:07:42+00 | SVKM's Dwarkadas J. Sanghvi College of Engine... |
| 72.8371462 | 19.1071622 | 2023/09/08 02:26:04+00 | 2023/09/08 11:13:20+00 | SVKM's Dwarkadas J. Sanghvi College of Engine... |
| 72.8371464 | 19.1070913 | 2023/09/26 05:11:06+00 | 2023/09/26 11:53:43+00 | SVKM's Dwarkadas J. Sanghvi College of Engine... |
| 72.8371542 | 19.1071930 | 2023/09/09 02:22:17+00 | 2023/09/09 07:56:21+00 | SVKM's Dwarkadas J. Sanghvi College of Engine... |
| 72.8371709 | 19.1073204 | 2023/09/06 04:41:37+00 | 2023/09/06 10:14:05+00 | SVKM's Dwarkadas J. Sanghvi College of Engine... |
| 72.8371823 | 19.1074310 | 2023/09/27 06:55:52+00 | 2023/09/27 09:27:40+00 | SVKM's Dwarkadas J. Sanghvi College of Engine... |

```
1 • select * from location;
2 • desc location;
3 • select * from location where lon like '73%';
4 • select * from location where location = "SVKM's Dwarkadas J. Sanghvi College of Engineering: No. U
5 • select * from location;
6 • select location, from_date from location where from_date like '2023/09/24%';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

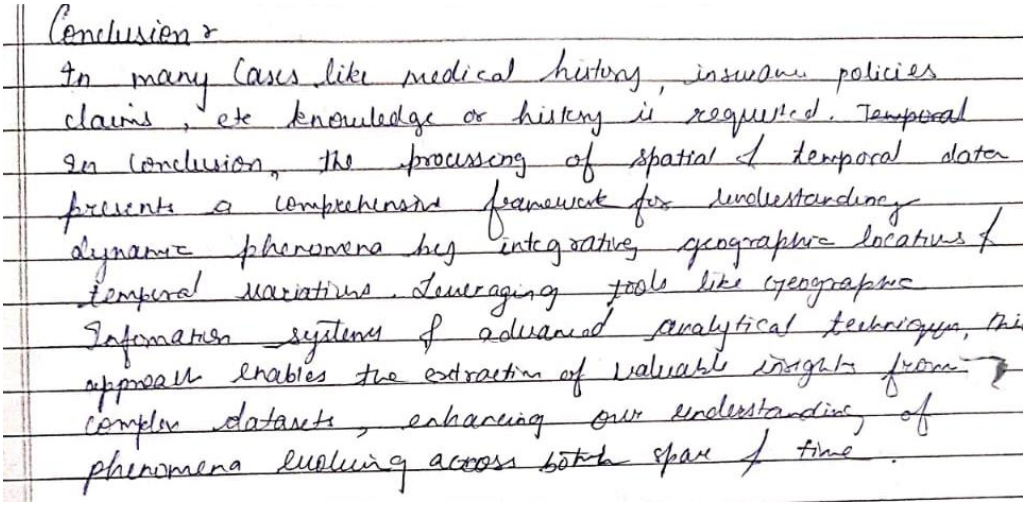
| location | from_date |
|---|---|
| Shakti CHS Ltd.: Datt Mandir Rd, Doghar Pada, ... | 2023/09/24 05:25:11+00 |
| Trimurti Snacks: Trimurti snacks ,N.H-17 ,At.-Po... | 2023/09/24 02:42:57+00 |
| HP PETROL PUMP - EKTA FUEL STATION: Mumb... | 2023/09/24 00:14:24+00 |

```
Limit to 1000 rows
1  select * from location;
2  desc location;
3  select * from location where lon like '73%';
4  select * from location where location = "SVKM's Dwarkadas J. Sanghvi College of Engineering: No. U, 15, Bha
5  select * from location;
6  select location, from_date from location where from_date like '2023/09/24%';
7
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| lon | lat | from_date | to_date | location |
|---|---|---|---|---|
| 73.1189191 | 18.9093882 | 2023/09/19 03:34:34+00 | 2023/09/19 03:48:47+00 | Karnala Bird Sanctuary: NH 66, Karnala, Navi M... |
| 73.1255770 | 18.9210050 | 2023/09/24 02:42:57+00 | 2023/09/24 03:01:19+00 | Trimurti Snacks: Trimurti snacks ,N.H-17 ,At.-Po... |
| 73.2769600 | 18.2444600 | 2023/09/24 00:14:24+00 | 2023/09/24 00:22:25+00 | HP PETROL PUMP - EKTA FUEL STATION: Mumb... |
| 73.2813325 | 18.2391333 | 2023/09/19 06:08:35+00 | 2023/09/19 06:23:22+00 | Mangaon ST Bus Depot: 67QJ+R93, Mangaon, ... |
| 73.3193725 | 17.8536075 | 2023/09/22 20:55:07+00 | 2023/09/23 19:52:51+00 | Visapur Dapoli Ratnagiri: Bus Stop, Visapur, Ma... |
| 73.3194580 | 17.8535920 | 2023/09/19 13:21:43+00 | 2023/09/21 12:01:26+00 | Chinchali Fata Bus Stop: Chinchali, Maharashtra... |
| 73.3196133 | 17.8558733 | 2023/09/23 19:52:51+00 | 2023/09/23 22:23:21+00 | Chinchali Fata Bus Stop: Chinchali, Maharashtra... |
| 73.3198370 | 17.8536348 | 2023/09/21 18:48:56+00 | 2023/09/22 20:55:07+00 | Chinchali Fata Bus Stop: Chinchali, Maharashtra... |
| 73.3203000 | 17.8537300 | 2023/09/21 12:01:26+00 | 2023/09/21 18:48:56+00 | Grampanchayat Office Shirkhal Chinchali: V82C... |
| NULL | NULL | NULL | NULL | NULL |

CONCLUSION:-



Conclusion :-

In many cases like medical history, insurance policies claims, etc knowledge or history is required. Temporal

In conclusion, the processing of spatial & temporal data presents a comprehensive framework for understanding dynamic phenomena by integrating geographic locations & temporal variations. Leveraging tools like Geographic Information systems & advanced analytical techniques, this approach enables the extraction of valuable insights from complex datasets, enhancing our understanding of phenomena evolving across both space & time.