Experiment No. 3

Title: Execution of In-memory database queries

Batch: B-4 **Roll No.: 16010422234** **Name: Chandana Ramesh Galgali**

Experiment No.: 3

Aim: To execute In-memory database queries

Resources needed: MySQL

Theory

In-Memory database is a database that uses a system's main memory for data storage rather than the disk-based storage typically utilized by traditional databases. In-memory databases, or IMDBs, are frequently employed in high-volume environments where response time is critical, as access times and database requests are typically considerably faster when system memory is used as opposed to hard disk storage.

The traditional databases and in-memory databases can be used together and referred to as hybrid databases, which support both in-memory and disk-based storage in order to maximize performance as well as reliability of the system. All most all RDBMS systems available in market supports In-Memory databases

MySQL In-Memory database:

In MySQL DB, the MEMORY storage engine creates special-purpose tables with contents that are stored in memory. Because the data is vulnerable to crashes, hardware issues, or power outages, use of these tables are limited to temporary work areas or read-only caches for data pulled from other tables.

A typical use case for the MEMORY engine involves these characteristics:

- Operations involving transient, non-critical data such as session management or caching. When the MySQL server halts or restarts, the data in MEMORY tables is lost.
- In-memory storage for fast access and low latency. Data volume can fit entirely in memory without causing the operating system to swap out virtual memory pages.
- A read-only or read-mostly data access pattern (limited updates).
- MEMORY tables cannot contain BLOB or TEXT columns.

To create a MEMORY table, specify the clause ENGINE=MEMORY on the CREATE TABLE statement

CREATE TABLE EMP (emp_Id INT, name CHAR (30)) ENGINE = MEMORY;

As indicated by the engine name, MEMORY tables are stored in memory. They use hash indexes by default, which makes them very fast for single-value lookups, and very useful for creating temporary tables. However, when the server shuts down, all rows stored in MEMORY tables are lost. The tables themselves continue to exist because their definitions are stored in .frm files on disk, but they are empty when the server restarts.

To load the data in memory from other existing table use,

> **CREATE TABLE EMP (emp_Id INT, name CHAR (30))) ENGINE=MEMORY as SELECT * FROM EMP;**

To move the data from In-Memory table to hard drive (using any text file) use the following syntax,

> **SELECT * INTO OUTFILE ''emp_data.txt' FROM EMP;**

To populate a MEMORY table when the MySQL server starts, use the INFILE option. For example,

> **LOAD DATA INFILE 'emp_data.txt' INTO TABLE EMP;**

Where, emp_data.txt is a data file.

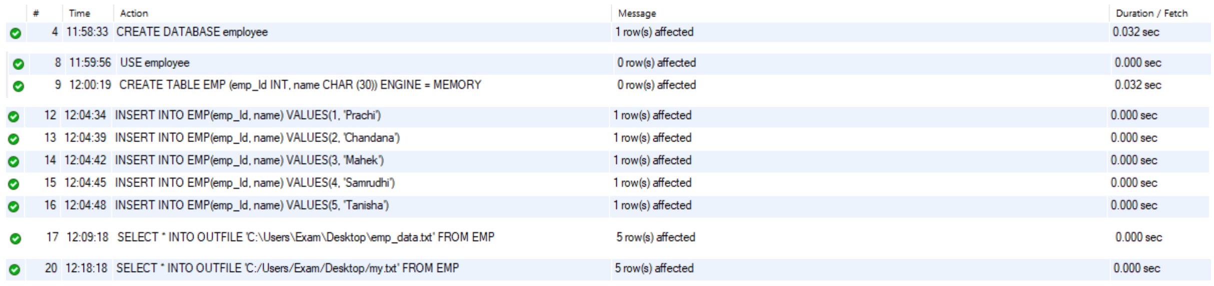
Procedure:

Perform following tasks:

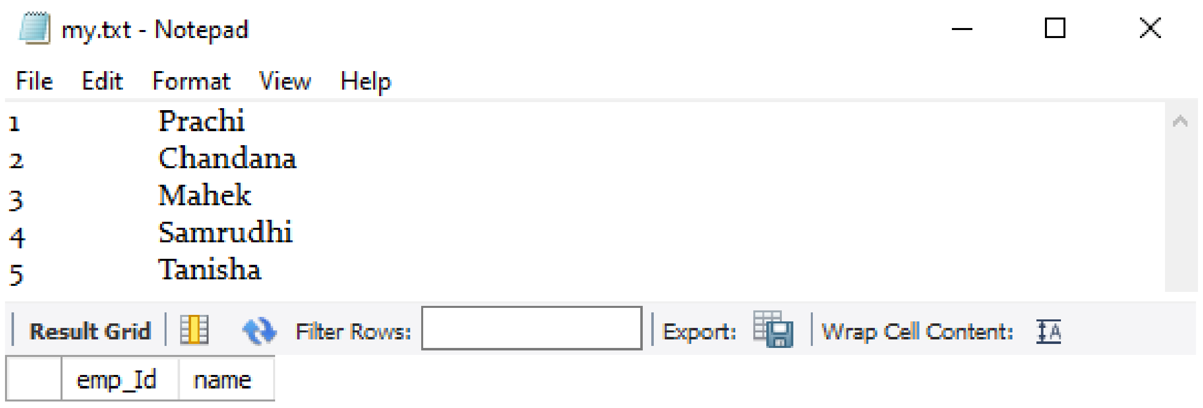
1. Create an In-memory table using Engine as Memory.
2. Insert values in that table.
3. Attempt to retrieve values from the table after restarting the database server.
4. Load the data into the table using file load.

Results: (Program printout with output)

```
CREATE DATABASE employee;
USE employee;
CREATE TABLE EMP (emp_Id INT, name CHAR (30)) ENGINE = MEMORY;
INSERT INTO EMP(emp_Id, name) VALUES(1, 'Prachi');
INSERT INTO EMP(emp_Id, name) VALUES(2, 'Chandana');
INSERT INTO EMP(emp_Id, name) VALUES(3, 'Mahek');
INSERT INTO EMP(emp_Id, name) VALUES(4, 'Samrudhi');
INSERT INTO EMP(emp_Id, name) VALUES(5, 'Tanisha');
SELECT * INTO OUTFILE 'C:/Users/Exam/Desktop/my.txt' FROM EMP;
SELECT * FROM EMP;
LOAD DATA INFILE 'C:/Users/Exam/Desktop/my.txt' INTO TABLE
EMP;
SELECT * FROM EMP;
```

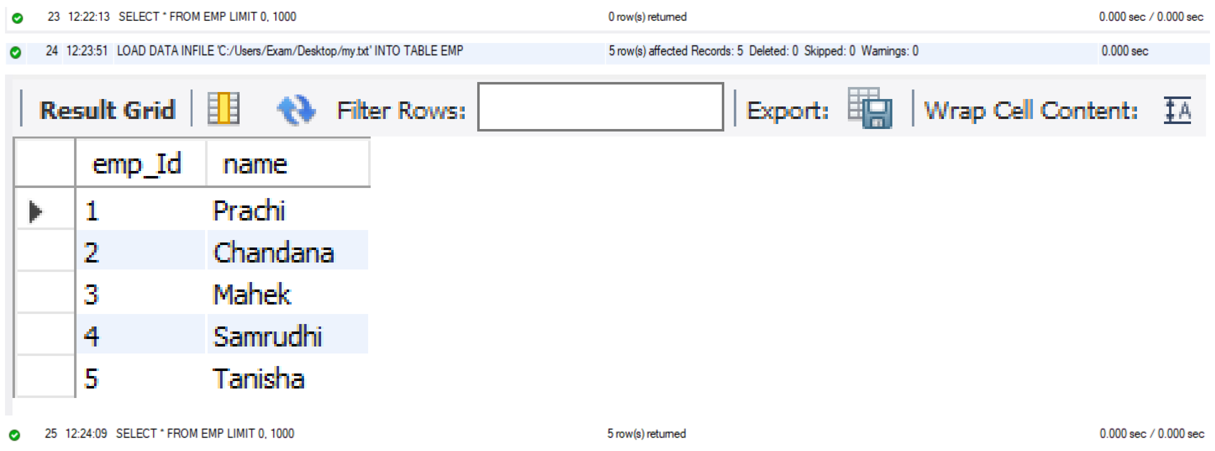


	#	Time	Action	Message	Duration / Fetch
✓	4	11:58:33	CREATE DATABASE employee	1 row(s) affected	0.032 sec
✓	8	11:59:56	USE employee	0 row(s) affected	0.000 sec
✓	9	12:00:19	CREATE TABLE EMP (emp_Id INT, name CHAR (30)) ENGINE = MEMORY	0 row(s) affected	0.032 sec
✓	12	12:04:34	INSERT INTO EMP(emp_Id, name) VALUES(1, 'Prachi')	1 row(s) affected	0.000 sec
✓	13	12:04:39	INSERT INTO EMP(emp_Id, name) VALUES(2, 'Chandana')	1 row(s) affected	0.000 sec
✓	14	12:04:42	INSERT INTO EMP(emp_Id, name) VALUES(3, 'Mahek')	1 row(s) affected	0.000 sec
✓	15	12:04:45	INSERT INTO EMP(emp_Id, name) VALUES(4, 'Samrudhi')	1 row(s) affected	0.000 sec
✓	16	12:04:48	INSERT INTO EMP(emp_Id, name) VALUES(5, 'Tanisha')	1 row(s) affected	0.000 sec
✓	17	12:09:18	SELECT * INTO OUTFILE 'C:\Users\Exam\Desktop\emp_data.txt' FROM EMP	5 row(s) affected	0.000 sec
✓	20	12:18:18	SELECT * INTO OUTFILE 'C:/Users/Exam/Desktop/my.txt' FROM EMP	5 row(s) affected	0.000 sec



```
1	Prachi
2	Chandana
3	Mahek
4	Samrudhi
5	Tanisha
```

emp_Id	name



```
23  12:22:13  SELECT * FROM EMP LIMIT 0, 1000                                  0 row(s) returned                                                0.000 sec / 0.000 sec
24  12:23:51  LOAD DATA INFILE 'C:/Users/Exam/Desktop/my.txt' INTO TABLE EMP   5 row(s) affected Records: 5  Deleted: 0  Skipped: 0  Warnings: 0   0.000 sec
```

emp_Id	name
1	Prachi
2	Chandana
3	Mahek
4	Samrudhi
5	Tanisha

```
25  12:24:09  SELECT * FROM EMP LIMIT 0, 1000                                  5 row(s) returned                                                0.000 sec / 0.000 sec
```

Questions:

1. **What is the difference between traditional and In-memory databases?**

Ans: Data Storage:

- Traditional databases typically store data on disk, which involves reading and writing data to and from disk during operations.
- In-memory databases, as the name suggests, store data in the system's main memory (RAM), allowing for faster data access.

Performance:

- Traditional databases may experience slower performance due to the need to access data from disk, resulting in higher latency.
- In-memory databases provide significantly faster performance since they can access data directly from RAM, eliminating the disk I/O bottleneck.

Processing Speed:

- Traditional databases may have slower processing speeds due to the time required for disk operations.
- In-memory databases can achieve faster processing speeds by keeping data in RAM, enabling quicker read and write operations.

Data Retrieval:

- Traditional databases rely on indexing and caching mechanisms to enhance data retrieval speed.
- In-memory databases, with data already in RAM, can retrieve information without the need for disk seeks, resulting in near-instantaneous data access.

Use Cases:

- Traditional databases are well-suited for scenarios with large datasets that do not require real-time processing.
- In-memory databases are ideal for applications demanding real-time analytics, high-performance transactions, and rapid data processing.

2. **List applications using in-memory databases. Explain any one of it stressing upon the advantage of using in-memory database.**

Ans: Real-Time Analytics:

- In-memory databases find extensive use in applications requiring real-time analytics, such as business intelligence, data warehousing, and reporting systems.
- One notable application is in the financial sector, where real-time analytics play a crucial role in making split-second decisions for trading and risk management.

Advantage - Real-Time Decision-Making:

- In the context of real-time analytics in finance, the advantage of using in-memory databases lies in the ability to make instant decisions based on the latest, up-to-the-moment data.
- Traditional databases might introduce latency due to disk I/O, which is not acceptable in scenarios where market conditions can change rapidly. In-memory databases ensure that data is readily available in RAM, allowing financial institutions to analyze market trends, assess risks, and execute trades in real-time without delays.

Example:

- Imagine a trading platform that needs to process a large volume of market data, perform complex analytics, and execute trades within fractions of a second. An in-memory database enables the platform to keep the most relevant market data in RAM, ensuring low-latency access and facilitating swift decision-making for traders.

Outcomes: Design advanced database systems using Parallel, Distributed and In-memory databases and its implementation.

Conclusion: (Conclusion to be based on outcomes achieved)

The experiment underscores the transformative impact of in-memory databases on query performance, throughput, and real-time data access. The outcomes support the growing trend of adopting in-memory databases for applications demanding rapid and efficient processing of data, positioning them as a valuable technology for enhancing overall system performance.

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

1. https://dev.mysql.com/doc/refman/5.5/en/memory-storage-engine.html
2. http://opensourceforu.efytimes.com/2012/01/importance-of-in-memory-databases/
3. http://pages.cs.wisc.edu/~jhuang/qual/main-memory-db-overview.pdf
4. http://docs.memsql.com