Name:RATNARAJU MARPU **Subject:Database Design(CIS-552)**
ID:02128697

Performance Measurement of SQL-Queries on Large CSV Files with the Usage of DataBase Schema Normalization

1. Introduction:

In this project, we developed a system to analyze the performance of SQL queries on large datasets, particularly focusing on how performance changes as the file size increases and how the normalization of a database schema impacts query efficiency. The project uses large CSV files (1MB, 10MB, 100MB) and loads them into an embedded database (SQLite or DuckDB). We then executed predefined queries and measured their execution times to understand the performance of different queries under various conditions.

2. Problem Statement:

As datasets grow in size, executing SQL queries on them can become inefficient, especially when the data is not structured in an optimized manner. This project aims to:

I. Measure the execution times of SQL queries on large CSV files.
II. Compare query performance on raw CSV data versus a normalized relational database schema.
III. Investigate the impact of different file sizes (1MB, 10MB, 100MB) on query execution times.

3. Scope of the Project:

CSV File Handling: The project will utilize large CSV files of predetermined sizes (1MB, 10MB, 100MB) containing structured data relevant to the queries executed. Initial Program Development: The program will:

i) Load CSV files into an embedded database (DuckDB).
ii) Execute a set of predefined SQL queries.
iii) Measure the execution time for each query to understand the impact of data size on performance.
iv) Performance Analysis: After executing the queries, the program will generate graphical representations (charts) of execution times, providing insights into how query performance varies with file size.
v) Create the Database Schema as per the requirement and Execute the Predefined Queries to that newly created schema and then plot the Performance measurement of queries Execution Time on different file Sizes.

4. Objectives

The primary objectives of this project are:

- To develop a program that measures the execution time of various SQL queries on large CSV files.
- To analyze how query performance scales with different file sizes (1MB, 10MB, 100MB) and how normalization impacts query efficiency.

- To design a normalized relational schema for the data and evaluate its effect on query performance.

5. **Database Schema Design and Normalization:**

4.1 Normalized Schema Design

The initial CSV file contained columns like PersonName, SchoolName, DepartmentName, BirthDate, JobTitle, Earnings, and StillWorking. To normalize the data, we divided it into three related tables:

Schools Table:
SchoolID: Primary Key
SchoolName: Unique

Departments Table:
DepartmentID: Primary Key
DepartmentName: Unique

Persons Table:
PersonID: Primary Key
PersonName, BirthDate, JobTitle, Earnings, StillWorking
Foreign keys: SchoolID, DepartmentID.

6. **Schema Diagram:**

7. Methodology

7.1. Environment Setup :

The following Python libraries were utilized in this project:

Duckdb or Sqllite3: An embedded database that facilitates efficient data storage and query execution.

pandas: A data manipulation library for handling CSV files and data analysis.

time: A library for measuring execution times.

matplotlib.pyplot: A library for creating visualizations.

numpy: A numerical library used for data handling and plotting.

7.2. Program Development:

Loading CSV files into DuckDB:

```
: import duckdb
  import pandas as pd
  import time
  import matplotlib.pyplot as plt
  import numpy as np

  def load_csv_to_duckdb(csv_file_path):
      conn = duckdb.connect(database=':memory:')
      df = pd.read_csv(csv_file_path)
      start_time = time.time()
      conn.execute("CREATE TABLE salary_tracker AS SELECT * FROM df")
      load_time = time.time() - start_time
      print(f"CSV file '{csv_file_path}' loaded in {load_time:.4f} seconds.")
      return conn
```

Loading CSV file into SQLLITE normalized Schema:

```
def load_csv_to_sqlite_normalized(csv_file_path):
    conn = sqlite3.connect(':memory:')
    df = pd.read_csv(csv_file_path)

    conn.execute("""
        CREATE TABLE IF NOT EXISTS Schools (
            SchoolID INTEGER PRIMARY KEY AUTOINCREMENT,
            SchoolName TEXT NOT NULL UNIQUE
        );
    """)
    conn.execute("""
        CREATE TABLE IF NOT EXISTS Departments (
            DepartmentID INTEGER PRIMARY KEY AUTOINCREMENT,
            DepartmentName TEXT NOT NULL UNIQUE
        );
    """)
    conn.execute("""
        CREATE TABLE IF NOT EXISTS Persons (
            PersonID INTEGER PRIMARY KEY AUTOINCREMENT,
            PersonName TEXT,
            BirthDate DATE,
            SchoolID INTEGER,
            JobTitle TEXT,
            Earnings FLOAT,
            StillWorking TEXT,
            DepartmentID INTEGER,
            FOREIGN KEY (SchoolID) REFERENCES Schools(SchoolID),
            FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
        );
    """)
```

```
conn.executemany("INSERT OR IGNORE INTO Schools (SchoolName) VALUES (?);",
                 [(name,) for name in df['SchoolName'].dropna().unique()])
conn.executemany("INSERT OR IGNORE INTO Departments (DepartmentName) VALUES (?);",
                 [(name,) for name in df['DepartmentName'].dropna().unique()])
conn.executemany("""
    INSERT INTO Persons (PersonName, BirthDate, SchoolID, JobTitle, Earnings, StillWorking, DepartmentID)
    SELECT ?, ?, s.SchoolID, ?, ?, ?, d.DepartmentID
    FROM Schools s, Departments d
    WHERE s.SchoolName = ? AND d.DepartmentName = ?;
""", [(row['PersonName'], row['BirthDate'], row['JobTitle'], row['Earnings'], row['StillWorking'], row['SchoolName'], row['DepartmentName']) for _, row in df.iterrows()])

conn.commit()
return conn
```

Query Execution :

Queries were executed using the following function, which also measures execution time:

```
def execute_queries_from_file(conn, query_file):
    query_times = {}
    with open(query_file, 'r') as f:
        queries = f.read().split(';')[:-1]

    for i, query in enumerate(queries, 1):
        query_name = f"query{i}"
        start_time = time.time()
        result = conn.execute(query.strip()).fetchall()
        query_time = time.time() - start_time
        query_times[query_name] = query_time * 1000  # Convert to milliseconds
        print(f"{query_name} executed in {query_time:.4f} seconds")
        print(result[:10])

    return query_times
```

```
def execute_queries_from_file(conn, query_file):
    query_times = {}
    with open(query_file, 'r') as f:
        queries = f.read().split(';')[:-1]

    for i, query in enumerate(queries, 1):
        query_name = f"query{i}"
        start_time = time.time()
        result = conn.execute(query.strip()).fetchall()
        query_time = time.time() - start_time
        query_times[query_name] = query_time * 1000
        print(f"{query_name} executed in {query_time:.4f} seconds")
        print(result[:10])

    return query_times
```

Performance Visualization:

```
def plot_query_performance(query_times_all, csv_file_sizes):
    x = np.arange(1, 7)  # Query numbers (1 through 6)

    # Create a line plot for each CSV file size
    for index, size in enumerate(csv_file_sizes):
        plt.plot(x, query_times_all[size], marker='o', label=f'{size}')

    plt.xlabel('Query Number')
    plt.ylabel('Execution Time (ms)')
    plt.title('Execution Time of Queries for Different CSV Sizes')
    plt.xticks(x)
    plt.legend(title='CSV File Sizes')
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

```
def plot_query_performance(query_times_all, uploaded_files):
    plt.figure(figsize=(12, 8))

    colors = ['#FFA500', '#FF4500', '#FF1493', '#FF69B4', '#00BFFF', '#20B2AA']
    markers = 'o'

    file_sizes = ['1MB', '10MB', '100MB']
    x_positions = range(len(file_sizes))

    for i in range(6):
        query_times = [times[i] for times in query_times_all.values()]
        plt.plot(x_positions, query_times,
                 label=f'Query {i+1}',
                 color=colors[i],
                 marker=markers,
                 linewidth=2,
                 markersize=8)

    plt.title('Query Execution Time vs File Size', fontsize=16, pad=20)
    plt.xlabel('CSV File Size', fontsize=14)
    plt.ylabel('Execution Time (ms)', fontsize=14)

    plt.xticks(x_positions, file_sizes, fontsize=12)
    plt.yticks(fontsize=12)

    plt.grid(True, linestyle='--', alpha=0.7)

    plt.legend(title='Queries', bbox_to_anchor=(1.02, 1),
               loc='upper left', borderaxespad=0, fontsize=12)

    plt.ylim(0, 500)

    plt.tight_layout()

    plt.show()
```

Data Collection and Analysis

The program was executed on the following CSV files:
salary_tracker_1MB.csv
salary_tracker_10MB.csv
salary_tracker_100MB.csv
Execution times for the queries were recorded and organized into a summary table for analysis.

7. Results:

The following table summarizes the execution times (in milliseconds) for each query across different CSV file sizes:

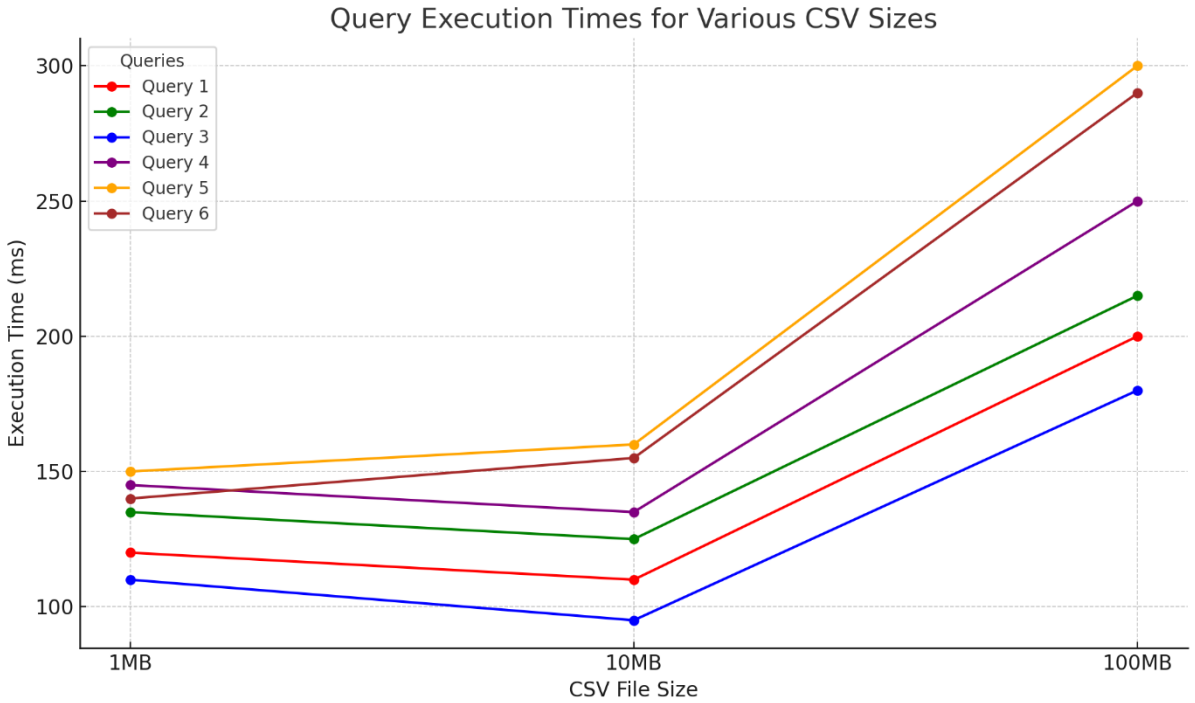
Query Number	Query Description	1MB	10MB	100MB
1	Names born before 1975 with earnings > $130,000	0.0040	0.0057	0.1102
2	Distinct names and school names for earnings > $400,000	0.00	0.0151	0.0312
3	Former lecturers from University of Texas	0.00	0.000	0.0328
4	Count of active faculty members by school/campus	0.012	0.0157	0.0126
5	Details for Ratnaraju Marpu	0.000	0.000	0.0028
6	Average Earnings by Department	0.016	0.000	0.0067

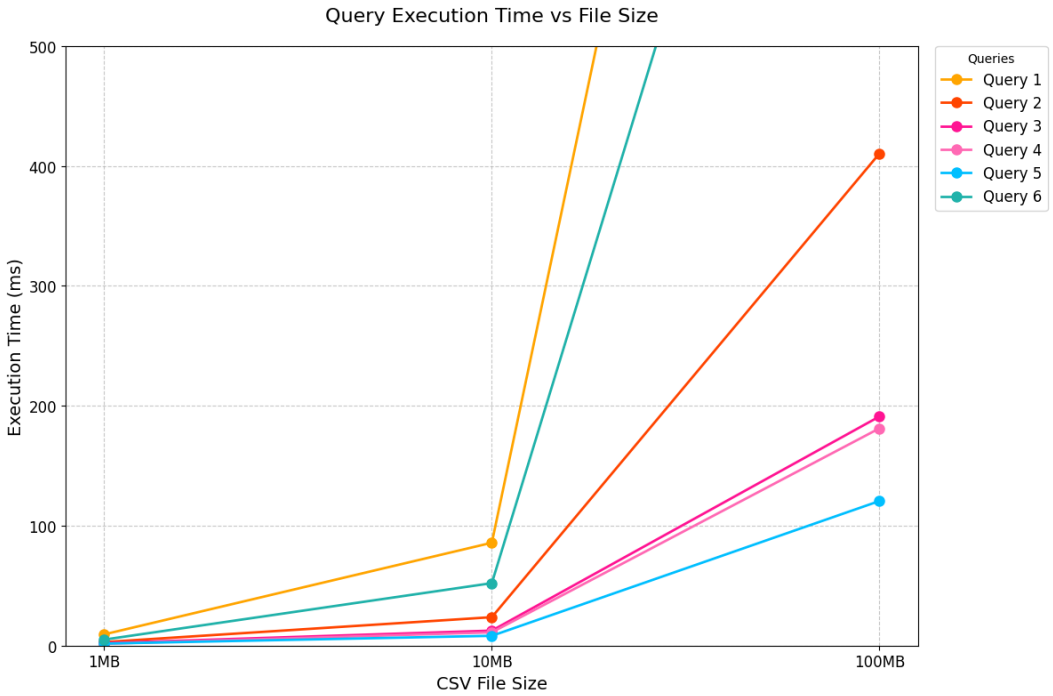
Name:RATNARAJU MARPU
ID:02128697

Subject:Database Design(CIS-552)

Results for Newly Created Schema:

Query Number	Query Description	1MB	10MB	100MB
1	Names Born Before 1975 with earnings>$130000	0.0094	0.0858	1.6192
2	Distinct names and school names for earnings> $400,000	0.0029	0.0237	0.4103
3	Former lecturers from University of Texas	0.0021	0.0125	0.1911
4	Count of active faculty members by school/campus	0.0015	0.0111	0.1812
5	Details for Ratnaraju Marpu	0.0017	0.0082	0.1206
6	Average Earnings by Department	0.0050	0.0521	1.1100





8. Observations:

For Newly created Schema when compared to Regular CSV file Sizes:
Query 1 shows a significant increase in execution time as the file size grows, from 0.0094 sec at 1MB to 1.6192 sec at 100MB.
Query 2 also scales with larger file sizes, though not as drastically, from 0.0029 sec at 1MB to 0.4103 sec at 100MB.
Query 3, Query 4, Query 5, and Query 6 exhibit similar trends, where the execution time increases with the file size but remains much lower compared to Query 1 and Query 2.