FULL NAME: Santhoshi Priya Sunchu
Student ID #: 02131881

CIS 552 Database Design – Fall 2024

Introduction:

This phase is an extended version of Phase 1, where the primary focus was on loading CSV datasets into an SQLite database and analyzing query performance based on dataset sizes. In Phase 1, the approach was centered around querying raw data without a formal schema.

The datasets used for this project are structured in CSV format with the following file sizes:

- salary_tracker_1MB.csv (1 MB)
- salary_tracker_10MB.csv (10 MB)
- salary_tracker_100MB.csv (100 MB)

Each CSV file contains salary information with fields such as PersonName, SchoolName, DepartmentName, BirthDate, JobTitle, Earnings, and other attributes related to individuals' employment details.

In Phase 2, we introduced a properly designed and normalized schema to improve data integrity. This phase includes refining the schema, incorporating relationships between tables through foreign keys, and expanding SQL queries. The performance of these queries will be evaluated, with a focus on how well the system handles more complex queries and larger datasets.

Schema Design

The database schema for this project is designed to ensure data integrity and optimize query performance. The schema is normalized to eliminate redundancy and ensure consistency by organizing the data into logical, related tables. We have created 5 tables.

- Person Table: This table stores information about individuals, which contains their ID(PersonID), name(PersonName), birthdate(BirthDate), and employment status(StillWorking).

```
CREATE TABLE IF NOT EXISTS Person (
    PersonID INTEGER PRIMARY KEY,
    PersonName TEXT NOT NULL,
    BirthDate TEXT,
    StillWorking TEXT )
```

This way, we have created the Person table by adding all the necessary columns. Here, PersonID is set as the primary key.

- School Table: This table contains information about the educational institutions, such as the school's name (SchoolName), Id(SchoolID) and location(SchoolCampus).Here, SchoolID is set as the primary key.

- Department Table: The Department Table stores information about departments, including their unique identifiers (DepartmentID, primary key)

- Job Table: The Job Table stores job-related information, including job titles and their associated departments. It contains columns Job (primary key), JobTitle, and DepartmentID, with a foreign key constraint ensuring that DepartmentID in the Job table references DepartmentID in the Department table.

- Salary Table: The Salary Table stores earnings information by linking individuals (PersonID), schools (SchoolID), and jobs (JobID). It includes columns for Earnings and EarningsYear to record income details over specific years. The table uses a composite primary key (PersonID, JobID, EarningsYear) to uniquely identify each record and track changes in earnings over time. Foreign key constraints ensure referential integrity with the Person, School, and Job tables.

The schema is normalized to 3NF, ensuring that all data is logically structured and reducing potential data anomalies. The relationships between the tables are established using foreign keys, ensuring referential integrity. By organizing the data this way, queries can be executed efficiently, and data redundancy is minimized.

Details of Executed Queries:

1. SELECT DISTINCT p.PersonName FROM Person p, Salary sWHERE s.PersonID = p.PersonID AND p.BirthDate < '1975-01-01' AND s.Earnings > 130000 AND s.EarningsYear = (SELECT MAX(s2.EarningsYear) FROM Salary s2 WHERE s2.PersonID = s.PersonID);

   This query retrieves the names of individuals born before 1975, whose most recent earnings exceed $130,000. We have structured it using a subquery to find the maximum earnings year for each person, ensuring the query only considers their latest earnings. Also used conditional statements to filter the results based on the BirthDate and Earnings.We joined the Person and Salary tables on PersonID to combine the relevant data from both tables

2. SELECT DISTINCT p.PersonName, sc.SchoolName FROM Person p, Salary s, School sc WHERE p.PersonID = s.PersonID AND sc.SchoolID = s.SchoolID AND s.Earnings > 400000 AND p.StillWorking = 'no'

   This query retrieves the names of individuals who earned more than $400,000 at any point in their careers and are no longer working. It joins the Person, Salary, and School tables based on matching IDs, and filters the results by earnings and the "StillWorking" status. The DISTINCT keyword we used ensures unique combinations of person and school names in the results.

3. SELECT DISTINCT p.PersonName FROM Person p, Salary s, Job j, School sc WHERE p.PersonID = s.PersonID AND s.JobID = j.JobID AND s.SchoolID = sc.SchoolID AND sc.SchoolName = 'University of Texas' AND j.JobTitle = 'Lecturer' AND p.StillWorking = 'no'

This query lists names of individuals who have worked as Lecturers at the University of Texas and are currently not active. It joins the Person, Salary, Job, and School tables based on matching IDs and filters the results by the school name, job title, and the "StillWorking" status. We have used the DISTINCT keyword to avoid duplicates and filters results specifically for the SchoolName and JobTitle.

4. SELECT sc.SchoolName, sc.SchoolCampus, COUNT(DISTINCT p.PersonID) AS faculty_count FROM Person p, Salary s, Job j, School sc WHERE p.PersonID = s.PersonID AND s.JobID = j.JobID AND s.SchoolID = sc.SchoolID AND p.StillWorking = 'yes' GROUP BY sc.SchoolName, sc.SchoolCampus ORDER BY faculty_count DESC LIMIT 1

   This query determines the university and campus pair with the highest number of currently active faculty members. It joins the Person, Salary, Job, and School tables and filters the results to include only those individuals who are still working . We used COUNT(DISTINCT p.PersonID) which counts the unique active faculty members for each school and campus, and the results are grouped by school name and campus, ordered by the count in descending order to return the top result.

5. SELECT p.PersonName, j.JobTitle, d.DepartmentName, sc.SchoolName, s.Earnings FROM Person p, Salary s, Department d, Job j, School sc WHERE p.PersonID = s.PersonID AND s.JobID = j.JobID AND d.DepartmentID = j.DepartmentID AND s.SchoolID = sc.SchoolID AND p.PersonName = 'Madison Harrison' AND s.EarningsYear = (SELECT MAX(EarningsYear) FROM Salary WHERE PersonID = p.PersonID);

   This query retrieves the most recent earnings, job title, department name, and school name for a specific individual named “Madison Harrison”. It joins the Person, Salary, Department, Job, and School tables to gather the necessary details. We have a subquery to ensure that it only retrieves the latest year of earnings for that “Madison Harrison”.

6. SELECT d.DepartmentName FROM Department d,Job J, Salary S WHERE J.JobID = S.JobID AND d.DepartmentID = J.DepartmentID GROUP BY d.DepartmentName ORDER BY AVG(S.Earnings) DESC Limit 1;

   This query identifies the department with the highest average earnings compared to other departments. It joins the Department, Job, and Salary tables using their respective foreign key relationships We have used AVG(S.Earnings) for average calculations and groups the results by DepartmentName, ordering them in descending order to find the top department.

Calculating Execution time

• Python's time library was used to measure the Query execution time.

Query Execution Time: For each query, the start and end times were recorded, and the execution time was calculated in milliseconds to understand how dataset size impacts retrieval and computation times. execution_times[size][query_name] = (end_time - start_time) * 1000
The measured times are presented in milliseconds to provide a detailed comparison.

Results:

Query Execution time for 1MB file:

q1 execution time: 3.6359 ms q2
execution time: 0.9229 ms q3
execution time: 1.3070 ms q4
execution time: 1.8110 ms q5
execution time: 1.1761 ms q6
execution time: 2.9581 ms

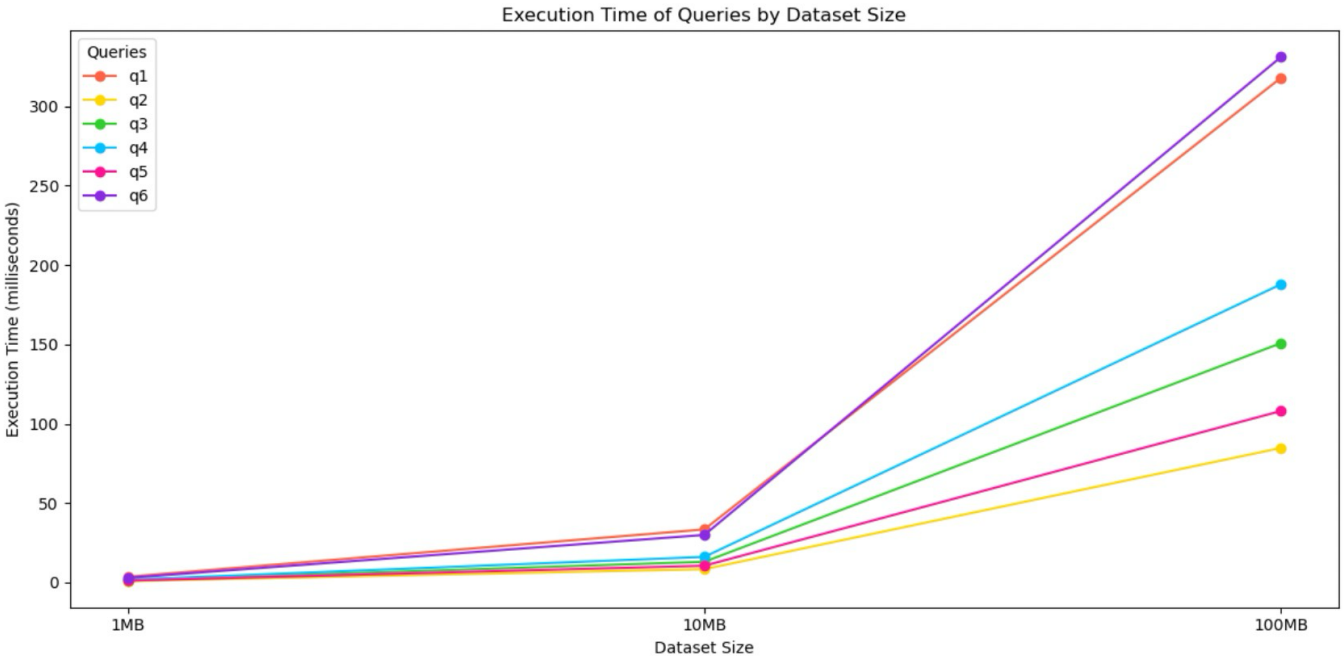
Execution time for 10MB file:

q1 execution time: 33.4373 ms
q2 execution time: 8.3640 ms q3
execution time: 13.0510 ms q4
execution time: 16.1459 ms q5
execution time: 10.5798 ms q6
execution time: 29.9911 ms

Execution time for 100MB file: q1 execution time:
317.8489 ms q2 execution time:
84.7342 ms q3 execution time:
150.7828 ms q4 execution time:
187.8479 ms q5 execution time:
108.0389 ms q6 execution time:
331.0122 ms

Visual Representation:

The line chart below shows the execution times for each query across the three dataset sizes (1MB, 10MB, and 100MB). Each line represents the execution time for a specific query and dataset size, allowing for a visual comparison of performance.



Observations:

- As dataset size increases from 1MB to 100MB, the execution time for each query also increases significantly.
- Queries q1 show relatively higher execution times for larger datasets, due to the complexity of filtering conditions and subquery usage.
- Queries q3, q2 which focus on filters or retrievals without aggregations perform faster in comparison to q1, q5.

Conclusion:

In this phase 2, we enhanced the database schema and updated our queries to evaluate SQLite's performance under varying dataset sizes. The results reveal that

- **Performance and Dataset Size**: Execution time increases with larger datasets, reaffirming SQLite's suitability for smaller-scale applications but highlighting its limitations with larger datasets.
- **Impact of Query Complexity**: Queries involving joins, aggregations, and subqueries are significantly affected by dataset size, demonstrating the need for optimization techniques.