Lab 7: SQLite Database Performance and Indexing

CSL303 - Database Management Systems Lab

October 7, 2025

Objective

The objective of this lab is to understand the impact of database indexing on query and insert performance in SQLite. You will create a database, populate it with a large dataset, run queries, analyze their execution plans, and use a programming language to measure the time difference with and without indexes.

Prerequisites

- Basic knowledge of SQL (CREATE TABLE, INSERT, SELECT, JOIN).
- SQLite3 command-line tool installed.
- A programming language with SQLite support installed (Python 3 is recommended and used in examples).

Instructions

This lab is divided into three parts. First, you will set up the database and run queries to establish a baseline. Second, you will add indexes and observe how the query plan changes. Finally, you will write a script to measure the performance changes quantitatively.

Part 1: Database Setup and Baseline Analysis

1. **Create the Database:** Open your terminal and create a new SQLite database file named `university.db`.

   ```
   sqlite3 university.db
   ```

2. **Create the Tables:** Inside the SQLite prompt, define the schema for three tables: `Students`, `Courses`, and `Enrollments`.

   ```sql
   CREATE TABLE Students (
       student_id INTEGER PRIMARY KEY,
       first_name TEXT NOT NULL,
       last_name TEXT NOT NULL,
       email TEXT NOT NULL UNIQUE,
       major TEXT,
       enrollment_date DATE
   );

   CREATE TABLE Courses (
       course_id INTEGER PRIMARY KEY,
       course_name TEXT NOT NULL,
       department TEXT NOT NULL,
       credits INTEGER
   );
   ```

```sql
CREATE TABLE Enrollments (
    enrollment_id INTEGER PRIMARY KEY,
    student_id INTEGER,
    course_id INTEGER,
    grade REAL,
    FOREIGN KEY(student_id) REFERENCES Students(student_id),
    FOREIGN KEY(course_id) REFERENCES Courses(course_id)
);
```

3. **Populate the Database:** Exit the SQLite prompt (`.quit`). A data file named `lab7_data.sql` has been provided, which will generate 2000 students, 100 courses, and 10000 enrollments. Load this data into your database using the following command:

   ```
   sqlite3 university.db < lab7_data.sql
   ```

   Verify that the data has been loaded by running a `COUNT(*)` query on each table.

4. **Simple Query:** Write a SQL query to find all students majoring in 'Computer Science'.

5. **Analyze the Simple Query:** Use `EXPLAIN QUERY PLAN` before your query from question 4. What does the output tell you about how SQLite is retrieving the data? Note down the result.

6. **Complex Join Query:** Write a SQL query to find the first name, last name, and course name for all students enrolled in a course from the 'Humanities' department.

7. **Analyze the Join Query:** Use `EXPLAIN QUERY PLAN` for the query from question 6. How many tables are being scanned? Note down the result.

Part 2: Adding Indexes

8. **Create an Index for a Column:** The query in question 4 was slow because it had to scan the entire `Students` table. Create an index on the `major` column.

   ```sql
   CREATE INDEX idx_students_major ON Students(major);
   ```

9. **Re-analyze the Simple Query:** Run `EXPLAIN QUERY PLAN` again for the query from question 4. What has changed in the execution plan? How does the new plan improve performance?

10. **Create Indexes for Joins:** The query in question 6 was slow because of the join operations. Create indexes on the foreign key columns in the `Enrollments` table.

    ```sql
    CREATE INDEX idx_enrollments_student_id ON Enrollments(student_id);
    CREATE INDEX idx_enrollments_course_id ON Enrollments(course_id);
    ```

11. **Re-analyze the Join Query:** Run `EXPLAIN QUERY PLAN` again for the query from question 6. How has the plan changed for the `Enrollments` table?

Part 3: Quantitative Performance Measurement

12. **Measure SELECT Time (Without Index):** Write a Python script to measure the execution time of the query from question 4. Make sure to **drop the index** `idx_students_major` first. Run the query in a loop (e.g., 100 times) to get an average time.

    Hint: Use the `time` module.

    ```python
    import sqlite3
    import time

    DB_FILE = "university.db"
    QUERY = "SELECT * FROM Students WHERE major = 'Computer Science';"
    ITERATIONS = 100

    con = sqlite3.connect(DB_FILE)
    cur = con.cursor()
    ```

```python
# Drop index if it exists
try:
    cur.execute("DROP INDEX idx_students_major")
    print("Index dropped.")
except sqlite3.OperationalError:
    print("Index did not exist.")

total_time = 0
for _ in range(ITERATIONS):
    start_time = time.time()
    cur.execute(QUERY).fetchall()
    end_time = time.time()
    total_time += (end_time - start_time)

print(f"Avg. time without index: {total_time / ITERATIONS:.6f} seconds")
con.close()
```

13. **Measure SELECT Time (With Index):** Modify your script to re-create the index `idx_students_major` and run the same timing measurement. Compare the average execution time with the result from question 12.

14. **Measure INSERT Time (With Indexes):** Indexes speed up reads but can slow down writes. Measure the time it takes to insert 500 new, random enrollments into the `Enrollments` table while the foreign key indexes (`idx_enrollments_student_id`, `idx_enrollments_course_id`) exist.

15. **Measure INSERT Time (Without Indexes):** Modify your script to first drop the two indexes on the `Enrollments` table and then measure the time for the same 500 insertions. Compare the results. What does this tell you about the trade-offs of using indexes?

Submission

Submit a report containing:

- The SQL queries for questions 4 and 6.
- The `EXPLAIN QUERY PLAN` outputs for questions 5, 7, 9, and 11, along with your analysis.
- Your Python script(s) used for Part 3.
- The timing results from questions 12, 13, 14, and 15, with a concluding paragraph explaining what the results demonstrate about database indexing.