Name : Ch Mubashir .

SAP : 56892.
Course: DataWareHouse.

Instructer: Junaid khan

Lab 02 :

Lab: Microsoft SQL Server Implementation.

Lab Objectives

• Reinforce the understanding of SQL categories: DDL, DML, DCL, and TCL.

• Design and implement a moderately complex relational database schema.

• Work with constraints, keys, and relationships between multiple entities.

• Write SQL queries to perform data retrieval with advanced clauses (JOINs, GROUP BY, HAVING, subqueries).

• Apply DCL commands to manage user privileges effectively.

• Use transaction control to ensure database consistency in multi-step operations.

• Gain practical experience in handling realistic enterprise-like problems in SQL Server.

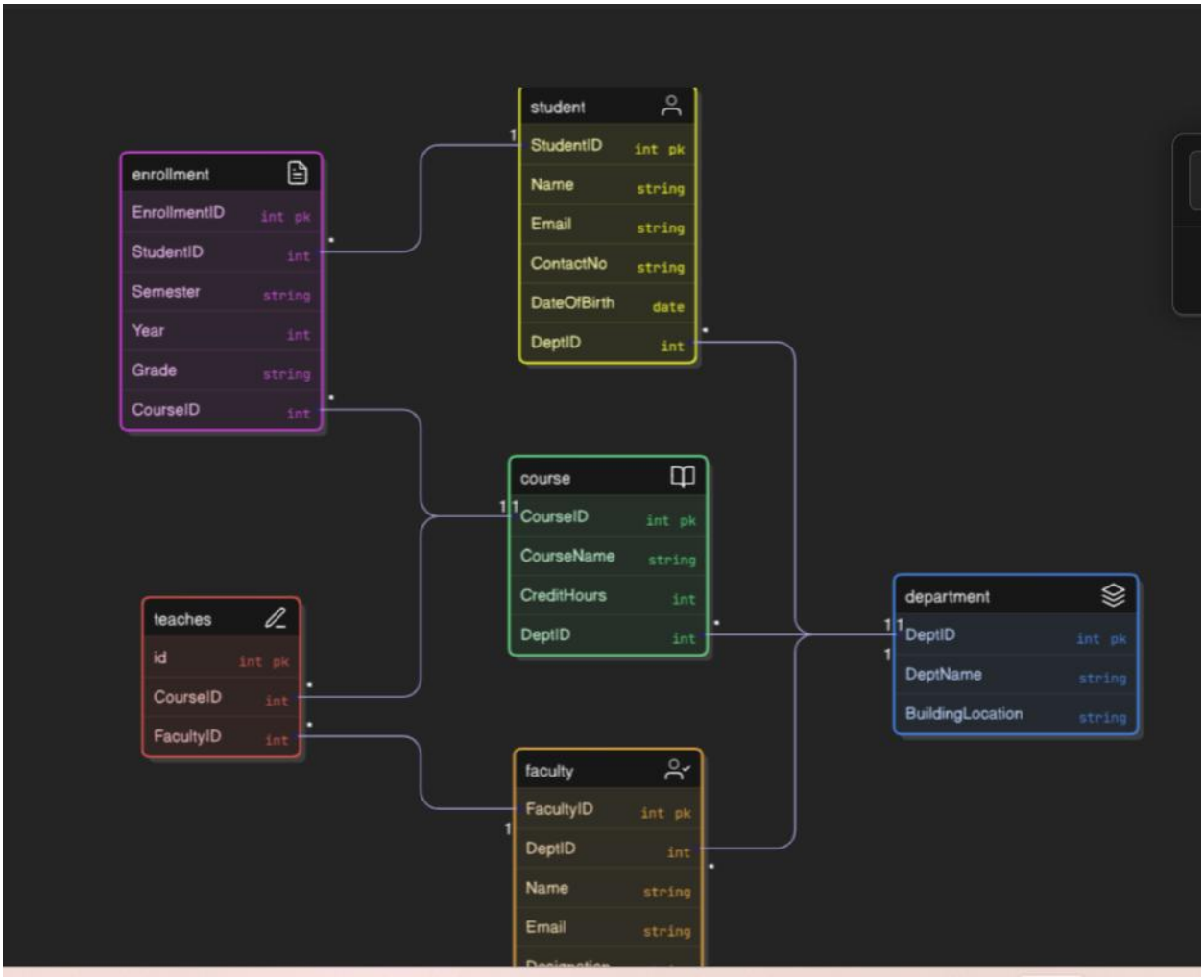
Scenario :

Entities and Attributes :

- **Student**
  - Attributes: StudentID (PK), Name, Email, ContactNo, DateOfBirth, DeptID (FK)
  - Constraints:
    - Email must be unique
    - DeptID references Department(DeptID)
- **Faculty**
  - Attributes: FacultyID (PK), Name, Email, Designation, DeptID (FK)
  - Constraints:
    - Faculty email should be unique (recommended)
    - DeptID references Department(DeptID)
- **Department**
  - Attributes: DeptID (PK), DeptName, BuildingLocation
  - Constraints:
    - DeptName must be unique
- **Course**
  - Attributes: CourseID (PK), CourseName, CreditHours, DeptID (FK)
  - Constraints:
    - Each Course must belong to a Department
    - DeptID references Department(DeptID)
- **Enrollment**
  - Attributes: EnrollmentID (PK), StudentID (FK), CourseID (FK), Semester, Year, Grade
  - Constraints:
    - StudentID must exist in Student(StudentID)
    - CourseID must exist in Course(CourseID)
    - Grade restricted to {A, B, C, D, F}
- **Teaches**
  - Attributes: FacultyID (FK), CourseID (FK) — Composite Primary Key (FacultyID, CourseID)
  - Constraints:
    - FacultyID references Faculty(FacultyID)
    - CourseID references Course(CourseID)
    - Many-to-many relationship (Faculty ↔ Course).

Database design :

ER Diagram :



SQL Implementation :

University Management Database

-1. Departments

```
CREATE TABLE Departments (
   dept_id INTEGER PRIMARY KEY,
   dept_name TEXT NOT NULL UNIQUE,
   building_location TEXT
);
```

2. Students

```
CREATE TABLE Students (
   student_id INTEGER PRIMARY KEY,
   name TEXT NOT NULL,
   email TEXT NOT NULL UNIQUE,
   contact_number TEXT,
   date_of_birth DATE,
   dept_id INTEGER,
   FOREIGN KEY (dept_id) REFERENCES Departments(dept_id)
);
```

3. Faculty

```
CREATE TABLE Faculty (
   faculty_id INTEGER PRIMARY KEY,
   name TEXT NOT NULL,
   email TEXT NOT NULL UNIQUE,
   designation TEXT,
   dept_id INTEGER,
   FOREIGN KEY (dept_id) REFERENCES Departments(dept_id)
);
```

4. Courses

```
CREATE TABLE Courses (
    course_id INTEGER PRIMARY KEY,
    course_name TEXT NOT NULL,
    credit_hours INTEGER NOT NULL CHECK (credit_hours > 0),
    dept_id INTEGER,
    FOREIGN KEY (dept_id) REFERENCES Departments(dept_id)
);
```

5. Enrollments

```
CREATE TABLE Enrollments (
    enrollment_id INTEGER PRIMARY KEY,
    student_id INTEGER,
    course_id INTEGER,
    semester TEXT NOT NULL,
    year INTEGER NOT NULL,
    grade TEXT CHECK (grade IN ('A', 'B', 'C', 'D', 'F')),
    FOREIGN KEY (student_id) REFERENCES Students(student_id),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);
```

6. Faculty-Course relationship (many-to-many)

```
CREATE TABLE Faculty_Course (
    faculty_id INTEGER,
    course_id INTEGER,
    PRIMARY KEY (faculty_id, course_id),
```

```
    FOREIGN KEY (faculty_id) REFERENCES Faculty(faculty_id),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);
```

-3. GPA Calculation Query

```
WITH GradePoints AS (
    SELECT
        e.enrollment_id,
        e.student_id,
        e.course_id,
        e.semester,
        e.year,
        CASE e.grade
            WHEN 'A' THEN 4.0
            WHEN 'B' THEN 3.0
            WHEN 'C' THEN 2.0
            WHEN 'D' THEN 1.0
            WHEN 'F' THEN 0.0
        END AS grade_point
    FROM Enrollments e
),
StudentGPA AS (
    SELECT
        s.student_id,
        s.name AS student_name,
        d.dept_name,
```

```
    ROUND(AVG(gp.grade_point), 2) AS gpa
  FROM Students s
  JOIN Departments d ON s.dept_id = d.dept_id
  JOIN GradePoints gp ON s.student_id = gp.student_id
  GROUP BY s.student_id, s.name, d.dept_name
),
RankedGPA AS (
  SELECT
    student_name,
    dept_name,
    gpa,
    ROW_NUMBER() OVER (PARTITION BY dept_name ORDER BY gpa DESC) AS rank_in_dept
  FROM StudentGPA
)
SELECT
  student_name,
  dept_name,
  gpa
FROM RankedGPA
WHERE rank_in_dept = 1;
```

Insert Data (DML) :

Insert Departments

```
INSERT INTO Departments (dept_id, dept_name, building_location) VALUES
(1, 'Computer Science', 'Block A'),
(2, 'Electrical Engineering', 'Block B'),
```

(3, 'Business Administration', 'Block C');

Insert Students

INSERT INTO Students (student_id, name, email, contact_number, date_of_birth, dept_id) VALUES

(101, 'Ali Khan', 'ali.khan@example.com', '03001234567', '2002-05-15', 1),

(102, 'Sara Ahmed', 'sara.ahmed@example.com', '03007654321', '2001-08-20', 1),

(103, 'Hamza Malik', 'hamza.malik@example.com', '03123456789', '2003-02-10', 2);

Insert Faculty

INSERT INTO Faculty (faculty_id, name, email, designation, dept_id) VALUES

(201, 'Dr. Ayesha Siddiqui', 'ayesha.siddiqui@example.com', 'Professor', 1),

(202, 'Dr. Imran Qureshi', 'imran.qureshi@example.com', 'Associate Professor', 2),

(203, 'Dr. Nadia Hassan', 'nadia.hassan@example.com', 'Assistant Professor', 3);

Insert Courses

INSERT INTO Courses (course_id, course_name, credit_hours, dept_id) VALUES

(301, 'Database Systems', 3, 1),

(302, 'Data Structures', 4, 1),

(303, 'Circuit Analysis', 3, 2),

(304, 'Marketing 101', 3, 3);

Insert Faculty-Course (many-to-many relationship)

INSERT INTO Faculty_Course (faculty_id, course_id) VALUES

(201, 301), -- Dr. Ayesha teaches Database Systems

(201, 302), -- Dr. Ayesha teaches Data Structures

(202, 303), -- Dr. Imran teaches Circuit Analysis

(203, 304); -- Dr. Nadia teaches Marketing 101

Insert Enrollments (with grades)

INSERT INTO Enrollments (enrollment_id, student_id, course_id, semester, year, grade) VALUES

(401, 101, 301, 'Fall', 2024, 'A'),

(402, 101, 302, 'Fall', 2024, 'B'),

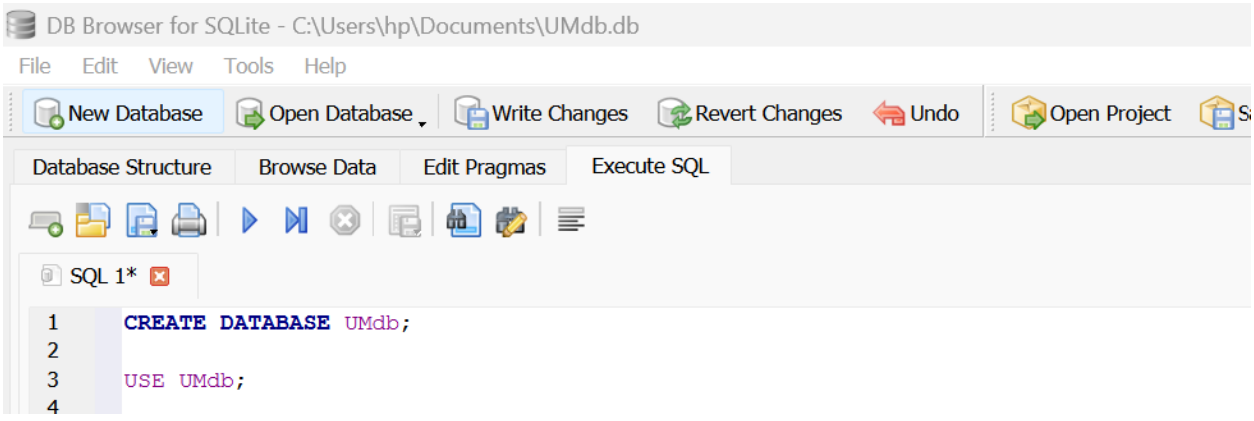
(403, 102, 301, 'Fall', 2024, 'A'),

(404, 103, 303, 'Fall', 2024, 'C'),

(405, 103, 304, 'Fall', 2024, 'B');

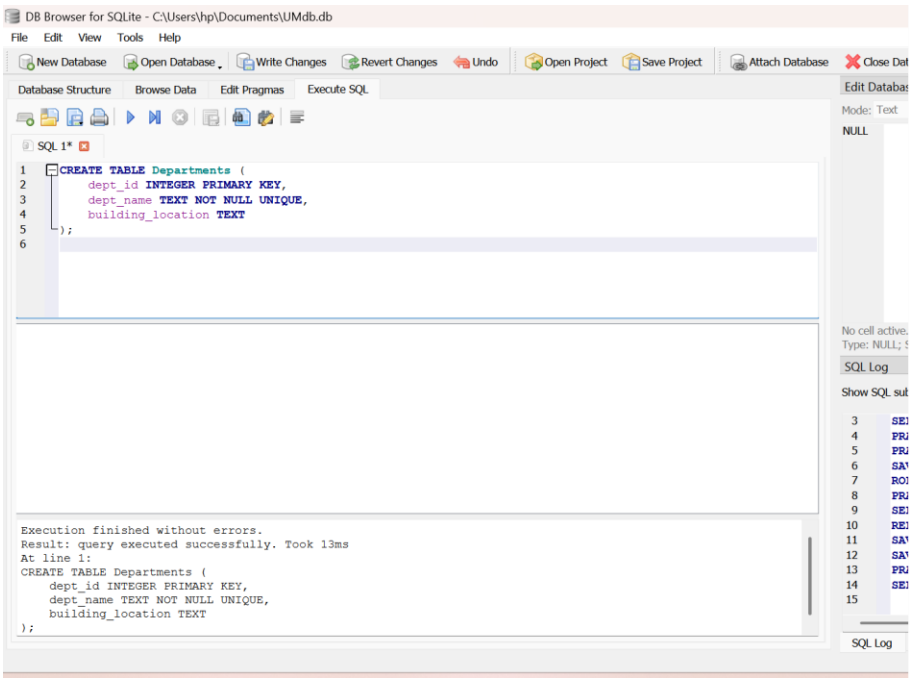
Screeenshots :

Creation of DataBase :

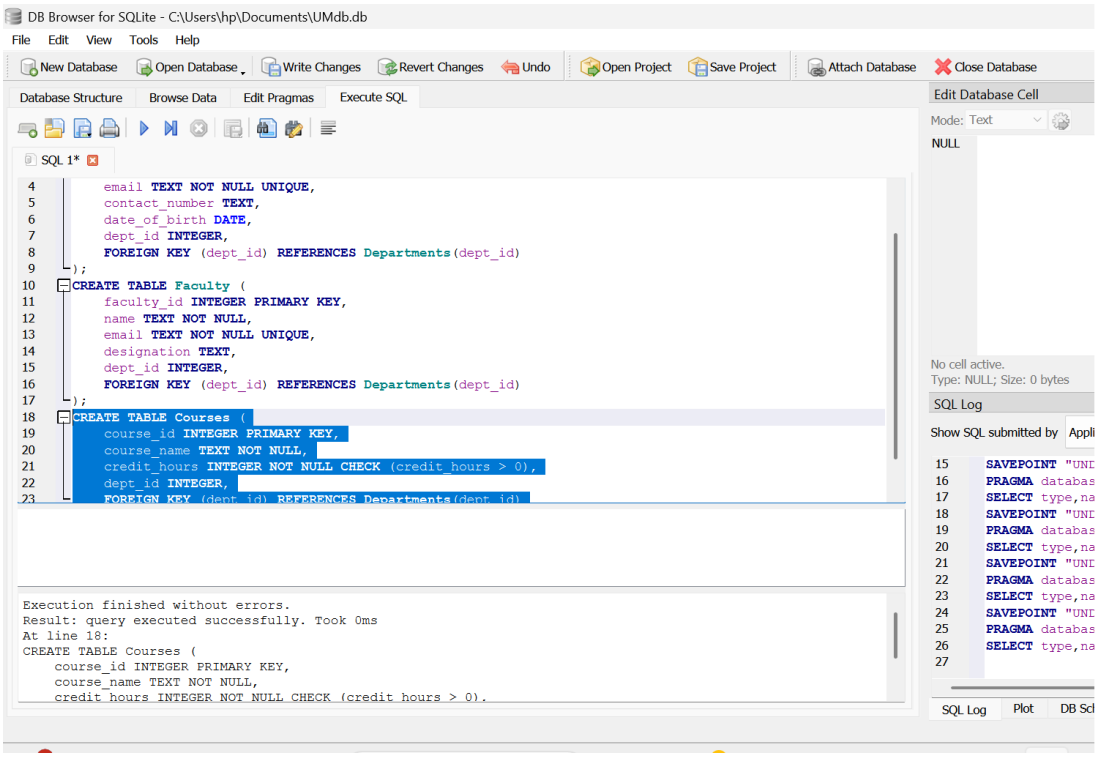


Creation of tables :

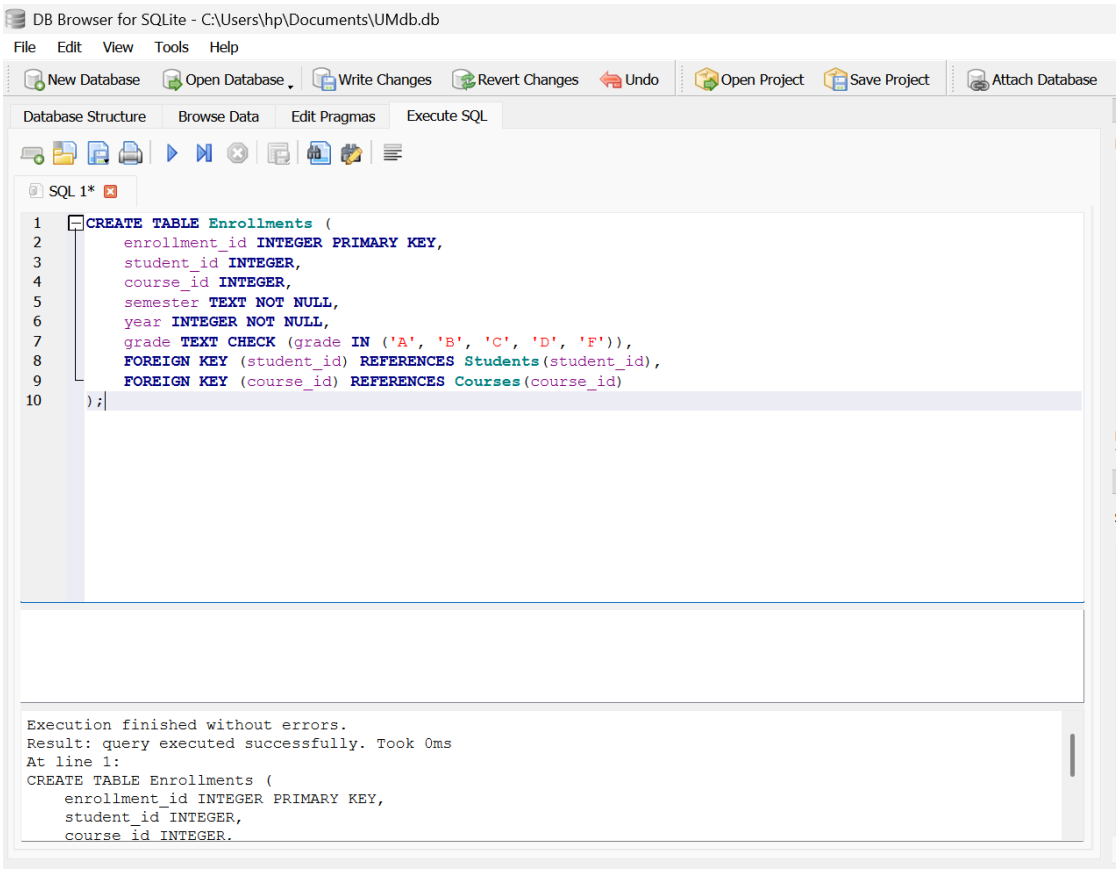
1.


DB Browser for SQLite - C:\Users\hp\Documents\UMdb.db
File Edit View Tools Help
New Database Open Database Write Changes Revert Changes Undo Open Project Save Project Attach Database Close Dat
Database Structure Browse Data Edit Pragmas Execute SQL
SQL 1*
CREATE TABLE Departments (
dept_id INTEGER PRIMARY KEY,
dept_name TEXT NOT NULL UNIQUE,
building_location TEXT
);
Execution finished without errors.
Result: query executed successfully. Took 13ms
At line 1:
CREATE TABLE Departments (
dept_id INTEGER PRIMARY KEY,
dept_name TEXT NOT NULL UNIQUE,
building_location TEXT
);
Edit Databas
Mode: Text
NULL
SQL Log

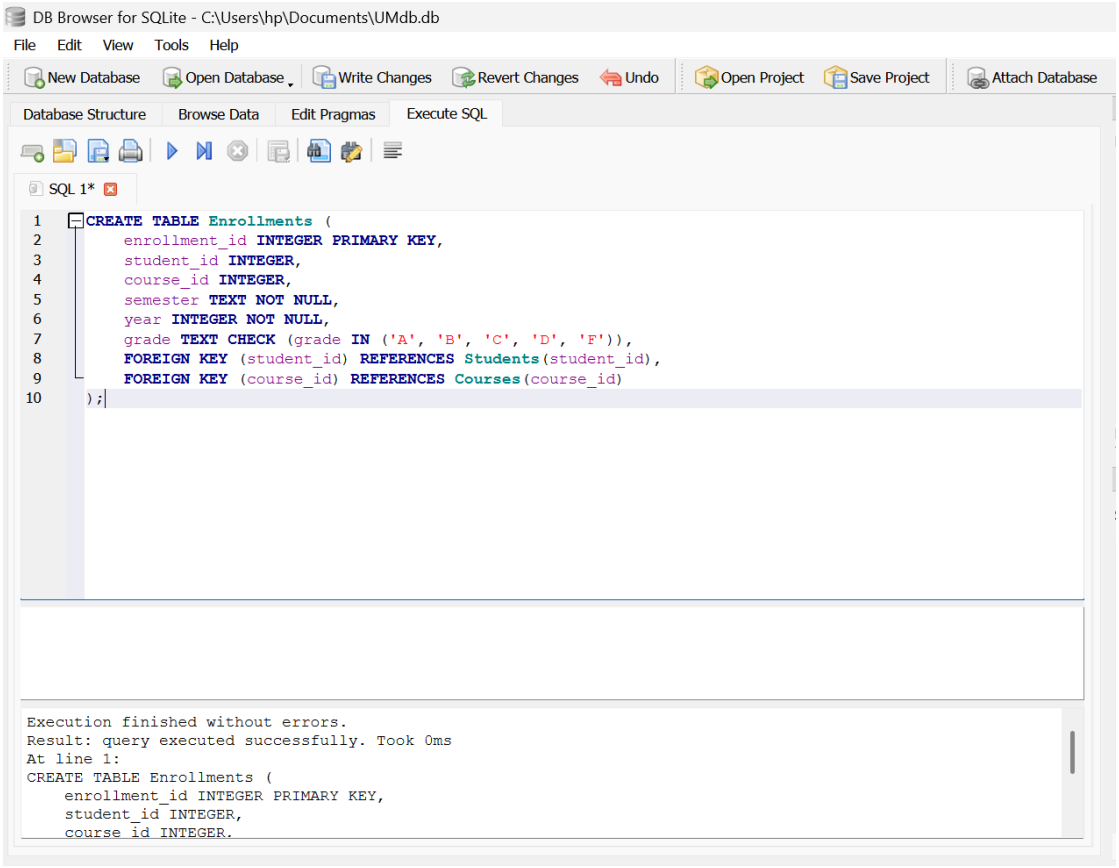
2.


DB Browser for SQLite - C:\Users\hp\Documents\UMdb.db
File Edit View Tools Help
New Database Open Database Write Changes Revert Changes Undo Open Project Save Project Attach Database Close Database
Database Structure Browse Data Edit Pragmas Execute SQL
SQL 1*
email TEXT NOT NULL UNIQUE,
contact_number TEXT,
date_of_birth DATE,
dept_id INTEGER,
FOREIGN KEY (dept_id) REFERENCES Departments(dept_id)
);
CREATE TABLE Faculty (
faculty_id INTEGER PRIMARY KEY,
name TEXT NOT NULL,
email TEXT NOT NULL UNIQUE,
designation TEXT,
dept_id INTEGER,
FOREIGN KEY (dept_id) REFERENCES Departments(dept_id)
);
CREATE TABLE Courses (
course_id INTEGER PRIMARY KEY,
course_name TEXT NOT NULL,
credit_hours INTEGER NOT NULL CHECK (credit_hours > 0),
dept_id INTEGER,
FOREIGN KEY (dept_id) REFERENCES Departments(dept_id)
Execution finished without errors.
Result: query executed successfully. Took 0ms
At line 18:
CREATE TABLE Courses (
course_id INTEGER PRIMARY KEY,
course_name TEXT NOT NULL,
credit_hours INTEGER NOT NULL CHECK (credit_hours > 0),
Edit Database Cell
Mode: Text
NULL
No cell active.
Type: NULL; Size: 0 bytes
SQL Log
Show SQL submitted by
SAVEPOINT "UNI
PRAGMA databas
SELECT type,na
SAVEPOINT "UNI
PRAGMA databas
SELECT type,na
SAVEPOINT "UNI
PRAGMA databas
SELECT type,na
SAVEPOINT "UNI
PRAGMA databas
SELECT type,na
SQL Log
Plot
DB Sc

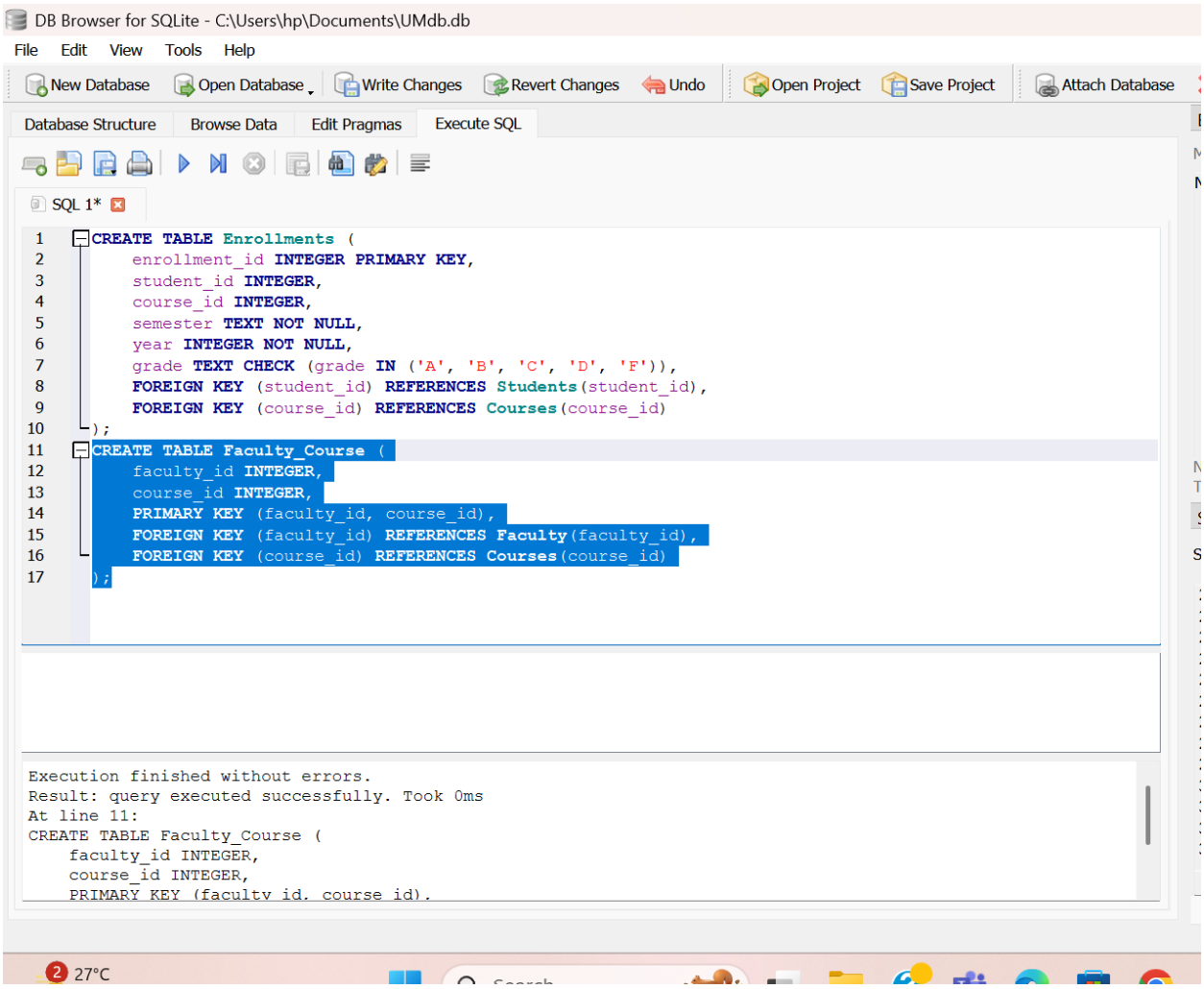
3.


DB Browser for SQLite - C:\Users\hp\Documents\UMdb.db
File Edit View Tools Help
New Database
Open Database
Write Changes
Revert Changes
Undo
Open Project
Save Project
Attach Database
Database Structure
Browse Data
Edit Pragmas
Execute SQL
SQL 1*
1 CREATE TABLE Enrollments (
2 enrollment_id INTEGER PRIMARY KEY,
3 student_id INTEGER,
4 course_id INTEGER,
5 semester TEXT NOT NULL,
6 year INTEGER NOT NULL,
7 grade TEXT CHECK (grade IN ('A', 'B', 'C', 'D', 'F')),
8 FOREIGN KEY (student_id) REFERENCES Students(student_id),
9 FOREIGN KEY (course_id) REFERENCES Courses(course_id)
10);
Execution finished without errors.
Result: query executed successfully. Took 0ms
At line 1:
CREATE TABLE Enrollments (
enrollment_id INTEGER PRIMARY KEY,
student_id INTEGER,
course_id INTEGER,

4.

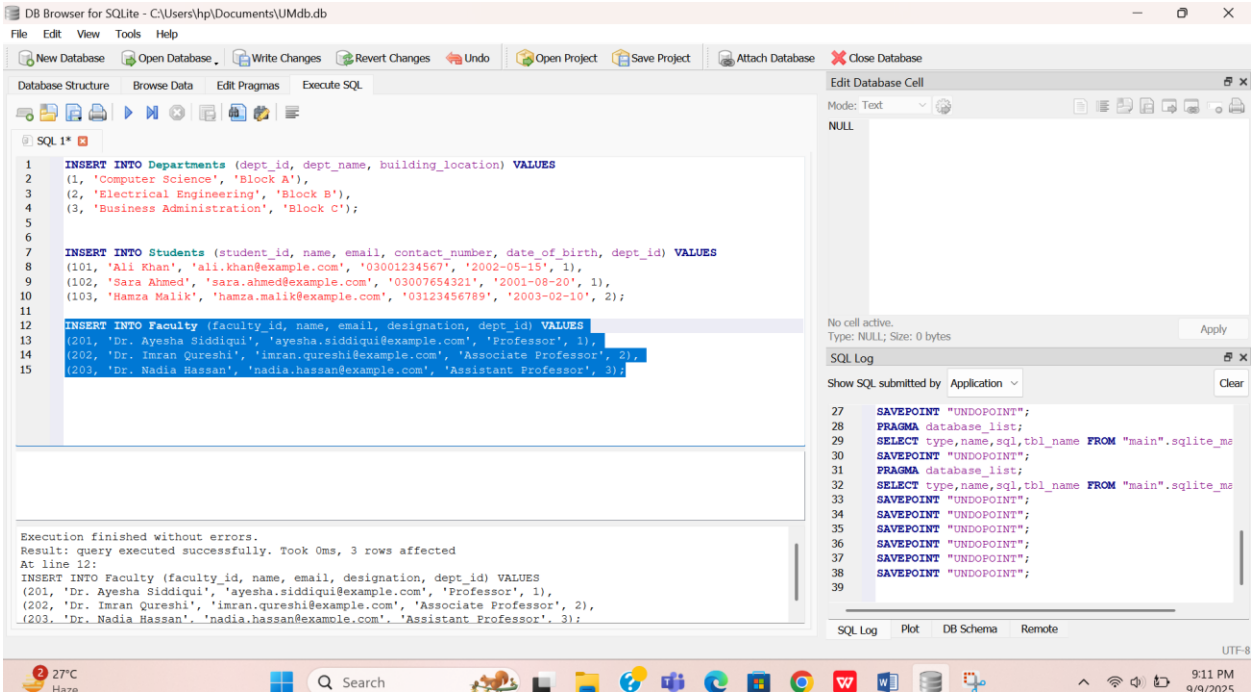

DB Browser for SQLite - C:\Users\hp\Documents\UMdb.db
File Edit View Tools Help
New Database Open Database Write Changes Revert Changes Undo Open Project Save Project Attach Database
Database Structure Browse Data Edit Pragmas Execute SQL
SQL 1*
1 CREATE TABLE Enrollments (
2 enrollment_id INTEGER PRIMARY KEY,
3 student_id INTEGER,
4 course_id INTEGER,
5 semester TEXT NOT NULL,
6 year INTEGER NOT NULL,
7 grade TEXT CHECK (grade IN ('A', 'B', 'C', 'D', 'F')),
8 FOREIGN KEY (student_id) REFERENCES Students(student_id),
9 FOREIGN KEY (course_id) REFERENCES Courses(course_id)
10);
Execution finished without errors.
Result: query executed successfully. Took 0ms
At line 1:
CREATE TABLE Enrollments (
enrollment_id INTEGER PRIMARY KEY,
student_id INTEGER,
course_id INTEGER,

5.

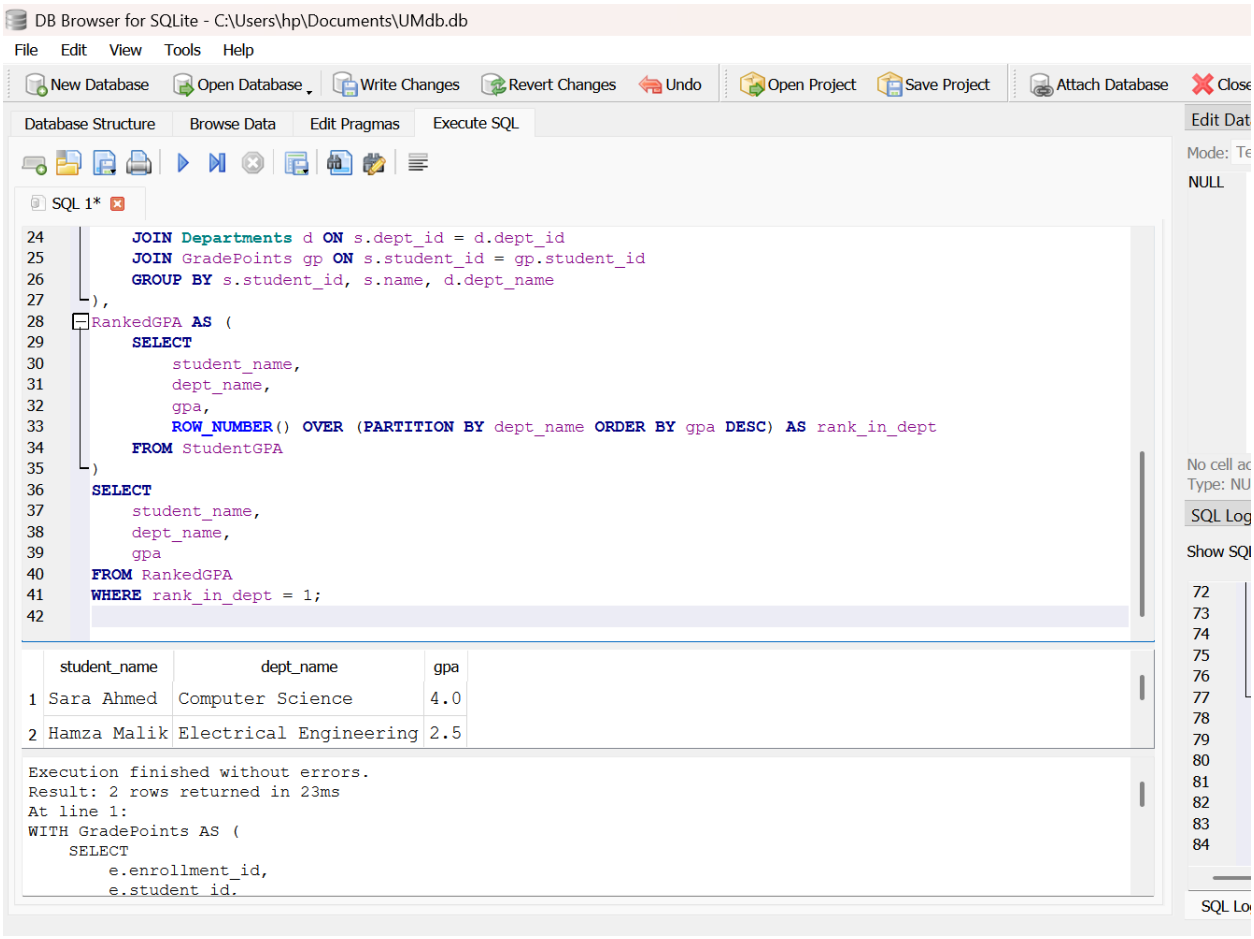


Insertion of Data :

1.



Sample Output :



Conclusion :

In this lab, we successfully designed and implemented a University Management Database System in MySQL Lite. We applied various SQL concepts including DDL, DML, constraints, relationships, and advanced queries. The GPA calculation and ranking query helped identify the top-performing student(s) per department, demonstrating practical use of SQL joins, aggregate functions, and window functions within a lightweight database environment.