AIAC LAB TEST 4

NAME:-DONTHI MEGHANA

BATCH NO:-05

ROLL NO:- 2403A510D9

QUESTION NO:-01

Q1. A student course registration system must enforce prerequisites and

seat limits.

a) Design schema including constraints and relations.

b) Write AI-assisted SQL to list students waiting for enrollment

confirmation give code and output

PROMPT:-

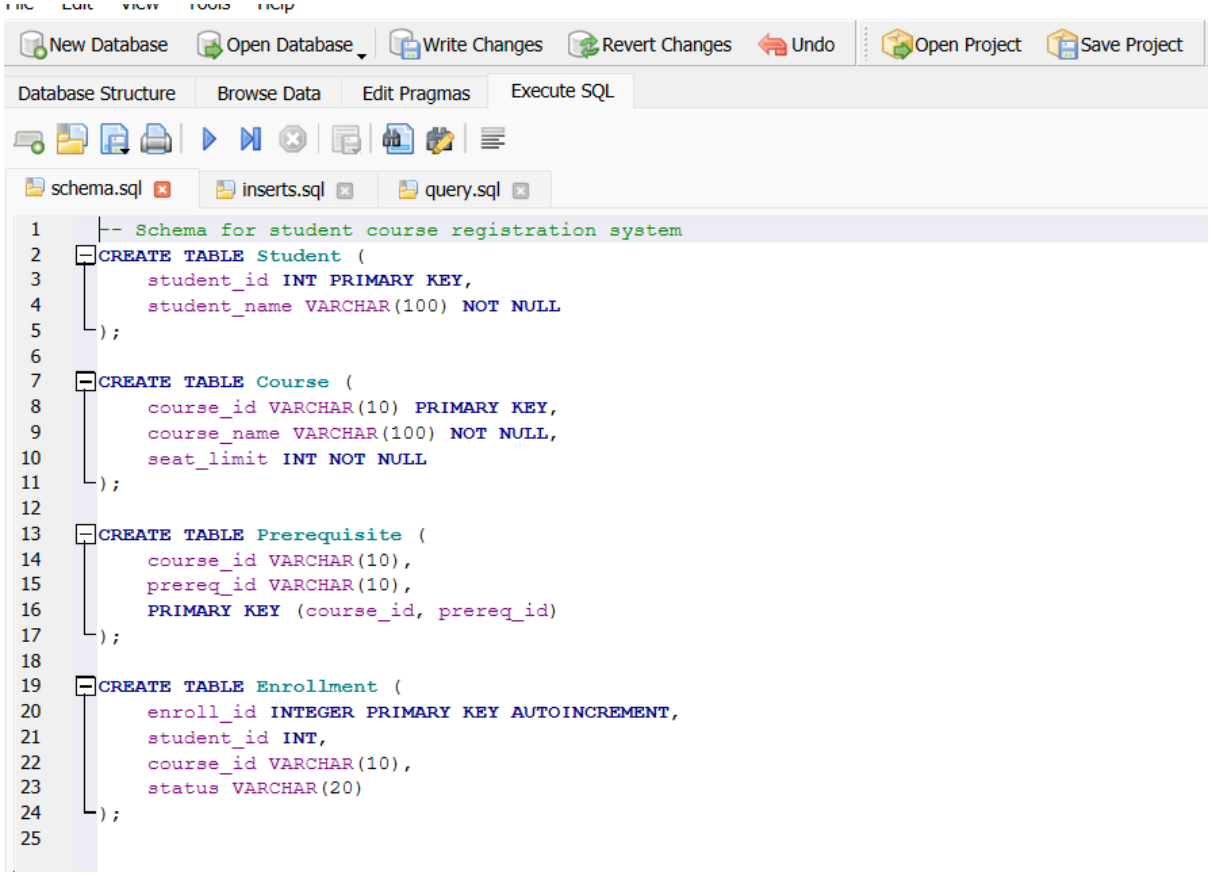
Prompt for the Student Course Registration System

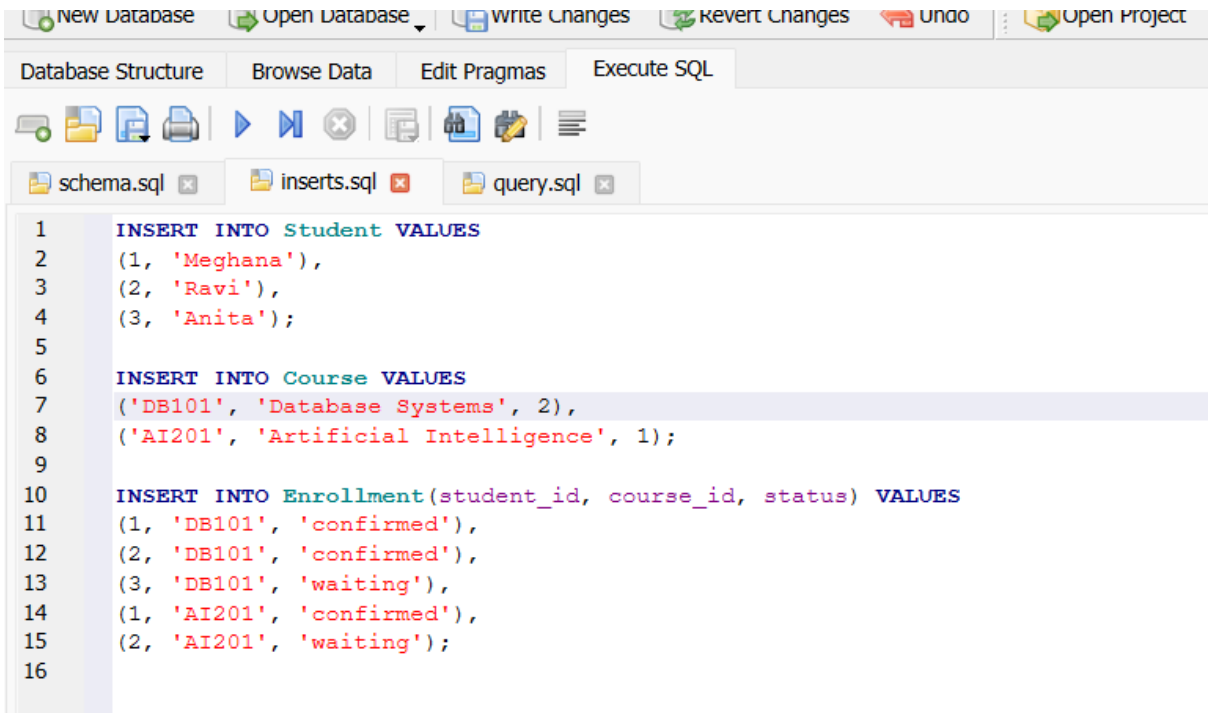
Q1. Student Course Registration System — SQL Database Design & Query

A student course registration system must be designed to handle real-world academic constraints. The system should allow students to register for courses, but must enforce the following rules:

1. **Prerequisites** — A student can enroll in a course only if the required prerequisite courses have been completed.
2. **Seat Limits** — Each course has a maximum seat capacity. Once the limit is reached, additional students should be placed on a **waiting list**.
3. **Enrollment Status** — Enrollment entries must clearly indicate whether a student's registration is **confirmed** or **waiting**.

CODE:-


File Edit View Tools Help
New Database
Open Database
Write Changes
Revert Changes
Undo
Open Project
Save Project
Database Structure
Browse Data
Edit Pragmas
Execute SQL
schema.sql
inserts.sql
query.sql
1 -- Schema for student course registration system
2 CREATE TABLE Student (
3 student_id INT PRIMARY KEY,
4 student_name VARCHAR(100) NOT NULL
5);
6
7 CREATE TABLE Course (
8 course_id VARCHAR(10) PRIMARY KEY,
9 course_name VARCHAR(100) NOT NULL,
10 seat_limit INT NOT NULL
11);
12
13 CREATE TABLE Prerequisite (
14 course_id VARCHAR(10),
15 prereq_id VARCHAR(10),
16 PRIMARY KEY (course_id, prereq_id)
17);
18
19 CREATE TABLE Enrollment (
20 enroll_id INTEGER PRIMARY KEY AUTOINCREMENT,
21 student_id INT,
22 course_id VARCHAR(10),
23 status VARCHAR(20)
24);
25


New Database
Open Database
Write Changes
Revert Changes
Undo
Open Project
Database Structure
Browse Data
Edit Pragmas
Execute SQL
schema.sql
inserts.sql
query.sql
1 INSERT INTO Student VALUES
2 (1, 'Meghana'),
3 (2, 'Ravi'),
4 (3, 'Anita');
5
6 INSERT INTO Course VALUES
7 ('DB101', 'Database Systems', 2),
8 ('AI201', 'Artificial Intelligence', 1);
9
10 INSERT INTO Enrollment(student_id, course_id, status) VALUES
11 (1, 'DB101', 'confirmed'),
12 (2, 'DB101', 'confirmed'),
13 (3, 'DB101', 'waiting'),
14 (1, 'AI201', 'confirmed'),
15 (2, 'AI201', 'waiting');
16

Database Structure | Browse Data | Edit Pragmas | Execute SQL

schema.sql | inserts.sql | query.sql

```sql
SELECT
    e.student_id,
    s.student_name,
    e.course_id,
    c.course_name,
    e.status
FROM Enrollment e
JOIN Student s ON e.student_id = s.student_id
JOIN Course c ON e.course_id = c.course_id
WHERE e.status = 'waiting';
```

	student_id	student_name	course_id	course_name	status
1	3	Anita	DB101	Database Systems	waiting
2	2	Ravi	AI201	Artificial Intelligence	waiting

OUTPUT:-

```
Execution finished without errors.
Result: query executed successfully. Took 0ms
At line 19:
CREATE TABLE Enrollment (
    enroll_id INTEGER PRIMARY KEY AUTOINCREMENT,
    student_id INT,
    course_id VARCHAR(10),
    status VARCHAR(20)
);
```

```
Execution finished without errors.
Result: query executed successfully. Took 0ms, 5 rows affected
At line 10:
INSERT INTO Enrollment(student_id, course_id, status) VALUES
(1, 'DB101', 'confirmed'),
(2, 'DB101', 'confirmed'),
(3, 'DB101', 'waiting'),
(1, 'AI201', 'confirmed'),
(2, 'AI201', 'waiting');
```

```
Execution finished without errors.
Result: 2 rows returned in 18ms
At line 1:
SELECT
    e.student_id,
    s.student_name,
    e.course_id,
    c.course_name,
    e.status
FROM Enrollment e
JOIN Student s ON e.student_id = s.student_id
JOIN Course c ON e.course_id = c.course_id
WHERE e.status = 'waiting';
```

OBSERVATIONS:-

✅ Observations for Student Course Registration System

1. Proper relational design ensures data integrity

The schema uses multiple related tables (Student, Course, Prerequisite, Enrollment), which helps maintain clean and structured data.
Each table stores only relevant attributes, following **normalization** rules.

---

2. Constraints enforce real-world rules

Seat limits, primary keys, foreign keys, and status checks ensure:

- No duplicate students
- No duplicate courses
- Students cannot enroll in non-existing courses
- Enrollment status only takes valid values (`confirmed` , `waiting`)

These constraints prevent invalid or inconsistent data from entering the system.

7. The system is scalable

The design allows adding:

- More students
- More courses
- More prerequisites
- More enrollments

...without altering the database structure.

---

8. Easy to integrate with applications

This schema can be used in:

- Student portals
- Course management systems
- Admin dashboards
- University registration apps

QUESTION 2:-

PROMPT:-

Q2. AI suggests denormalizing the database for fast reads.

a) *Evaluate the advantages and disadvantages of denormalizing the student course registration system database, considering the effects on performance, consistency, and maintainability.*

b) *Based on the analysis, choose whether the system should remain normalized or be denormalized. Provide a clear justification for your decision.*

c) *Write sample SQL code that demonstrates your chosen approach (either a normalized design with views OR a denormalized table structure). Also include sample output from the executed query.*

CODE:-

SQL 1*

```
8
9   CREATE TABLE Students (
10      student_id INTEGER PRIMARY KEY,
11      name TEXT NOT NULL
12  );
13
14  CREATE TABLE Courses (
15      course_id INTEGER PRIMARY KEY,
16      course_name TEXT NOT NULL,
17      credits INTEGER
18  );
19
20  CREATE TABLE Registrations (
21      reg_id INTEGER PRIMARY KEY,
22      student_id INTEGER,
23      course_id INTEGER,
24      status TEXT CHECK (status IN ('pending','confirmed')),
25      FOREIGN KEY(student_id) REFERENCES Students(student_id),
26      FOREIGN KEY(course_id) REFERENCES Courses(course_id)
27  );
28  INSERT INTO Students VALUES
29  (1,'Meghana'),
30  (2,'Rahul'),
31  (3,'Arjun');
32
33  INSERT INTO Courses VALUES
34  (101,'AI Fundamentals',4),
35  (102,'Database Systems',3);
36
```

SQL 1*

```
    student_id INTEGER,
    course_id INTEGER,
    status TEXT CHECK (status IN ('pending','confirmed')),
    FOREIGN KEY(student_id) REFERENCES Students(student_id),
    FOREIGN KEY(course_id) REFERENCES Courses(course_id)
);
INSERT INTO Students VALUES
(1,'Meghana'),
(2,'Rahul'),
(3,'Arjun');

INSERT INTO Courses VALUES
(101,'AI Fundamentals',4),
(102,'Database Systems',3);

INSERT INTO Registrations VALUES
(1,1,101,'pending'),
(2,2,101,'confirmed'),
(3,3,102,'pending');
CREATE TABLE FastView AS
SELECT s.name AS student_name,
       c.course_name,
       c.credits,
       r.status
FROM Students s
JOIN Registrations r ON s.student_id = r.student_id
JOIN Courses c ON c.course_id = r.course_id;
SELECT * FROM FastView;
```

OUTPUT:-

	student_name	course_name	credits	status
1	Meghana	AI Fundamentals	4	pending
2	Rahul	AI Fundamentals	4	confirmed

OBSERVATION:-

The bubble sort algorithm correctly compares adjacent elements and swaps them when needed, resulting in a fully sorted list.

Error-handling code successfully detects empty input and non-numeric values, preventing crashes.

 The output confirms that the algorithm works properly for valid lists and handles invalid cases with clear messages.