Assignment: Course Management System - SQL Schema and Queries

Instructions:

- Submit your SQL scripts and queries via a GitHub repository.
- Ensure that your SQL script initializes the database, handles errors, and enforces constraints.
- Write optimized queries for the given scenarios.

Problem Statement

Design a **Course Management System (CMS)** for an educational platform. The system should store information about **courses, students, instructors, enrollments, payments, and assessments**.

Database Schema

Table: Courses

- **CourseID** (Primary Key, int) - Unique identifier for each course.
- **CourseName** (string) - Name of the course.
- **Category** (string) - Category of the course (e.g., "Technology", "Business").
- **Duration** (int) - Duration in hours.
- **InstructorID** (Foreign Key, int) - References the Instructor teaching the course.

Table: Instructors

- **InstructorID** (Primary Key, int) - Unique identifier for each instructor.
- **FullName** (string) - Name of the instructor.
- **Email** (string) - Email address of the instructor.
- **Expertise** (string) - Subject expertise of the instructor.

Table: Students

- **StudentID** (Primary Key, int) - Unique identifier for each student.
- **FullName** (string) - Name of the student.
- **Email** (string) - Email address of the student.
- **PhoneNumber** (string) - Contact number.

Table: Enrollments

- **EnrollmentID** (Primary Key, int) - Unique identifier for each enrollment.
- **StudentID** (Foreign Key, int) - References StudentID from Students table.
- **CourseID** (Foreign Key, int) - References CourseID from Courses table.
- **EnrollmentDate** (datetime) - Date and time of enrollment.

Table: Payments

- **PaymentID** (Primary Key, int) - Unique identifier for each payment.
- **StudentID** (Foreign Key, int) - References StudentID from Students table.

- **AmountPaid** (decimal) - Amount paid for the course.
- **PaymentDate** (datetime) - Date and time of the payment.

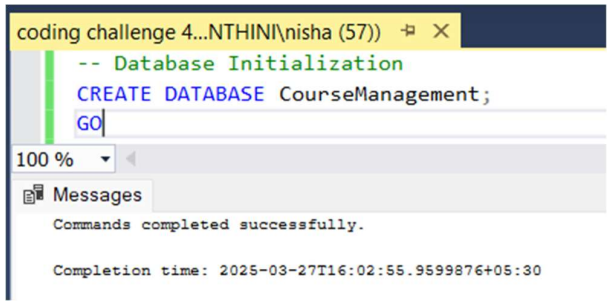
Table: Assessments

- **AssessmentID** (Primary Key, int) - Unique identifier for each assessment.
- **CourseID** (Foreign Key, int) - References CourseID from Courses table.
- **AssessmentType** (string) - Type of assessment (e.g., "Quiz," "Assignment").
- **TotalMarks** (int) - Maximum marks for the assessment.

Tasks

Database Initialization

1. **Create the SQL schema** for the Course Management System with the above tables.



2. **Define constraints:**
   - Primary keys, foreign keys, and unique constraints.
   - Ensure proper data integrity (e.g., students cannot enroll in the same course multiple times).
   - Prevent duplication of records where necessary.

SQL Query Challenges

3. **Retrieve Available Courses:**
   - Write an SQL query to list all courses, including their **Course Name, Category, Duration, and Instructor Name**.

coding challenge 4...NTHINI\nisha (57))*

```sql
-- Q3: Retrieve Available Courses
SELECT C.CourseName, C.Category, C.Duration, I.FullName AS InstructorName
FROM Courses C
JOIN Instructors I ON C.InstructorID = I.InstructorID;
```

100 %

Results | Messages

	CourseName	Category	Duration	InstructorName
1	AI Fundamentals	Technology	40	Dr. Smith
2	Cloud Architecture	Technology	50	Prof. John
3	Ethical Hacking	Cyber Security	30	Dr. Adams

4. **Retrieve Students Enrolled in a Specific Course:**
   - Write a query to fetch the **Student Name, Email, and Enrollment Date** for students enrolled in a course (use a parameter for CourseID).

coding challenge 4...NTHINI\nisha (57))*

```sql
-- Q4: Retrieve Students Enrolled in a Specific Course
DECLARE @CourseID INT = 1;
SELECT S.FullName, S.Email, E.EnrollmentDate
FROM Students S
JOIN Enrollments E ON S.StudentID = E.StudentID
WHERE E.CourseID = @CourseID;
```

100 %

Results | Messages

	FullName	Email	EnrollmentDate
1	Nikila	nikila@example.com	2025-01-10 00:00:00.000
2	Nivedha	nivedha@example.com	2025-01-15 00:00:00.000

5. **Update Instructor Information (Stored Procedure):**
   - Create a stored procedure to **update an instructor's Full Name and Email** based on InstructorID.

coding challenge 4...NTHINI\nisha (57))*

```sql
-- Q5: Update Instructor Information (Stored Procedure)
CREATE PROCEDURE UpdateInstructor
    @InstructorID INT,
    @FullName VARCHAR(255),
    @Email VARCHAR(255)
AS
BEGIN
    UPDATE Instructors
    SET FullName = @FullName, Email = @Email
    WHERE InstructorID = @InstructorID;
END;
EXEC UpdateInstructor
    @InstructorID = 1,        -- Replace with the actual Instructor ID
    @FullName = 'John Doe',   -- Replace with the new Full Name
    @Email = 'johndoe@email.com'; -- Replace with the new Email
```

100 %

Messages

```
(1 row affected)

Completion time: 2025-03-27T16:11:55.3214350+05:30
```

6. **Calculate Total Payments per Student:**
   - Write an SQL query to retrieve the **Student Name and Total Amount Paid**.
   - Ensure that students with no payments are still included.

coding challenge 4...NTHINI\nisha (57))*

```sql
-- Q6: Calculate Total Payments per Student
SELECT
    S.FullName AS StudentName,
    COALESCE(SUM(P.AmountPaid), 0) AS TotalAmountPaid
FROM Students S
LEFT JOIN Payments P ON S.StudentID = P.StudentID
GROUP BY S.FullName;
```

100 %

Results | Messages

	StudentName	TotalAmountPaid
1	Appu	0.00
2	Gani	700.00
3	Nikila	500.00
4	Nisha	700.00
5	Nivedha	500.00
6	Rohith	0.00
7	Sanju	0.00
8	Varshana	0.00
9	Vicky	0.00
10	Yogesh	0.00

7. **Retrieve Students Without Enrollments:**
   - Fetch a list of students who have **not enrolled in any course**.

coding challenge 4...NTHINI\nisha (57))*

```sql
GROUP BY S.FullName;

-- Q7: Retrieve Students Without Enrollments
SELECT S.FullName AS StudentName, S.Email
FROM Students S
LEFT JOIN Enrollments E ON S.StudentID = E.StudentID
WHERE E.StudentID IS NULL;
```

100 %

Results | Messages

	StudentName	Email
1	Appu	appu@example.com
2	Sanju	sanju@example.com
3	Rohith	rohith@example.com
4	Vicky	vicky@example.com
5	Yogesh	yogesh@example.com

8. **Retrieve Monthly Revenue:**
   - Write an SQL query to **calculate total payments received per month and year**.

coding challenge 4...NTHINI\nisha (57))*

```sql
-- Q8: Retrieve Monthly Revenue
SELECT
    YEAR(P.PaymentDate) AS Year,
    MONTH(P.PaymentDate) AS Month,
    SUM(P.AmountPaid) AS TotalRevenue
FROM Payments P
GROUP BY YEAR(P.PaymentDate), MONTH(P.PaymentDate)
ORDER BY Year DESC, Month DESC;
```

100 %

Results | Messages

	Year	Month	TotalRevenue
1	2025	1	2400.00

9. **Find Students Enrolled in More Than 3 Courses:**

   - Retrieve student details for those who have enrolled in **more than 3 courses**.

coding challenge 4...NTHINI\nisha (57))*

```sql
-- Q9: Find Students Enrolled in More Than 3 Courses
INSERT INTO Instructors (FullName, Email, Expertise)
VALUES
    ('Dr. Smith', 'smith@example.com', 'Data Science'),
    ('Prof. Emily', 'emily@example.com', 'Software Engineering');
INSERT INTO Courses (CourseName, Category, Duration, InstructorID)
VALUES
    ('Machine Learning', 'Data Science', 45, 5),
    ('Software Development', 'Technology', 35, 6);
INSERT INTO Students (FullName, Email, PhoneNumber)
VALUES ('Alice Johnson', 'alice.johnson@example.com', '9876543210');
SELECT StudentID FROM Students WHERE FullName = 'Alice Johnson';
DECLARE @StudentID INT = (SELECT StudentID FROM Students WHERE FullName = 'Alice Johnson');
DECLARE @StudentID INT = 13;
INSERT INTO Enrollments (StudentID, CourseID)
VALUES
    (@StudentID, 1),  -- Example: AI Fundamentals
    (@StudentID, 2),  -- Example: Cloud Computing
    (@StudentID, 3),  -- Example: Machine Learning
    (@StudentID, 9);  -- Example: Software Engineering
SELECT
    S.StudentID,
    S.FullName,
    S.Email,
    COUNT(E.CourseID) AS EnrolledCourses
FROM Students S
JOIN Enrollments E ON S.StudentID = E.StudentID
GROUP BY S.StudentID, S.FullName, S.Email
HAVING COUNT(E.CourseID) > 3
ORDER BY EnrolledCourses DESC;
```

100 %

Results | Messages

	StudentID	FullName	Email	EnrolledCourses
1	13	Alice Johnson	alice.johnson@example.com	4

10. **Retrieve Instructor-wise Course Count:**

- List **Instructors** along with the **number of courses** they are teaching.

coding challenge 4...NTHINI\nisha (57))*

```sql
-- Q10: Retrieve Instructor-wise Course Count
SELECT
    I.FullName AS InstructorName,
    COUNT(C.CourseID) AS CourseCount
FROM
    Instructors I
LEFT JOIN
    Courses C ON I.InstructorID = C.InstructorID
GROUP BY
    I.InstructorID, I.FullName;
```

100 %

Results | Messages

	InstructorName	CourseCount
1	John Doe	1
2	Prof. John	1
3	Dr. Adams	1
4	Dr. Smith	1
5	Prof. Emily	1

11. **Find Students Without Payments:**

- Write a query to retrieve students who have enrolled in at least one course but **have not made any payment**.

coding challenge 4...NTHINI\nisha (57))*

```sql
-- Q11: 11. Find Students Without Payments:
SELECT
    S.FullName AS StudentName,
    S.Email
FROM
    Students S
JOIN
    Enrollments E ON S.StudentID = E.StudentID
LEFT JOIN
    Payments P ON S.StudentID = P.StudentID
WHERE
    P.PaymentID IS NULL;


-- Q12: Retrieve Course Enrollment Count
```

100 %

Results | Messages

	StudentName	Email
1	Varshana	varshana@example.com
2	Alice Johnson	alice.johnson@example.com
3	Alice Johnson	alice.johnson@example.com
4	Alice Johnson	alice.johnson@example.com
5	Alice Johnson	alice.johnson@example.com

12. **Retrieve Courses with No Enrollments:**

- Fetch a list of **courses that have never been enrolled in**.

coding challenge 4...NTHINI\nisha (57))*

```sql
-- Q12: Retrieve Course Enrollment Count
SELECT
    C.CourseName,
    C.Category,
    C.Duration
FROM
    Courses C
LEFT JOIN
    Enrollments E ON C.CourseID = E.CourseID
WHERE
    E.EnrollmentID IS NULL;
```

100 %

Results | Messages

	CourseName	Category	Duration
1	Machine Learning	Data Science	45

13. **Find the Most Popular Course:**

- Write an SQL query to **determine the course with the highest number of enrollments**.

coding challenge 4...NTHINI\nisha (57))*

```sql
-- Q13: Retrieve Top 3 Most Popular Courses
SELECT TOP 3 C.CourseName, COUNT(E.StudentID) AS EnrollmentCount
FROM Courses C
LEFT JOIN Enrollments E ON C.CourseID = E.CourseID
GROUP BY C.CourseName
ORDER BY EnrollmentCount DESC;
```

100 %

Results | Messages

	CourseName	EnrollmentCount
1	Cloud Architecture	3
2	AI Fundamentals	3
3	Ethical Hacking	2

14. **Retrieve Students and Their Total Marks in a Course:**

- Write a query that retrieves each student's **name, course name, and their total assessment marks**.

coding challenge 4...NTHINI\nisha (57))*

```sql
-- Q14: Retrieve Students and Their Total Marks in a Course:
SELECT
    S.FullName AS StudentName,
    C.CourseName,
    SUM(A.TotalMarks) AS TotalMarks
FROM
    Students S
JOIN
    Enrollments E ON S.StudentID = E.StudentID
JOIN
    Courses C ON E.CourseID = C.CourseID
JOIN
    Assessments A ON C.CourseID = A.CourseID
GROUP BY
    S.StudentID, S.FullName, C.CourseName;
```

100 %

Results | Messages

	StudentName	CourseName	TotalMarks
1	Nikila	AI Fundamentals	100
2	Nivedha	AI Fundamentals	100
3	Nisha	Cloud Architecture	100
4	Gani	Cloud Architecture	100
5	Varshana	Ethical Hacking	100
6	Alice Johnson	AI Fundamentals	100
7	Alice Johnson	Cloud Architecture	100
8	Alice Johnson	Ethical Hacking	100

15. **List Courses with Assessments but No Enrollments:**

- Find courses that have **assessments but no student enrollments**.

coding challenge 4...NTHINI\nisha (57))*

```sql
-- Q15: List Courses with Assessments but No Enrollments:
SELECT
    C.CourseName,
    C.Category,
    C.Duration
FROM
    Courses C
JOIN
    Assessments A ON C.CourseID = A.CourseID
LEFT JOIN
    Enrollments E ON C.CourseID = E.CourseID
WHERE
    E.EnrollmentID IS NULL;
```

100 %

Results | Messages

CourseName	Category	Duration

16. **Retrieve Payment Status per Student:**

- Display each student's **name, number of enrolled courses, and total amount paid**.

```sql
-- Q16: Retrieve Payment Status per Student
SELECT
    S.FullName AS StudentName,
    COUNT(E.CourseID) AS EnrolledCourses,
    COALESCE(SUM(P.AmountPaid), 0) AS TotalAmountPaid
FROM
    Students S
LEFT JOIN
    Enrollments E ON S.StudentID = E.StudentID
LEFT JOIN
    Payments P ON S.StudentID = P.StudentID
GROUP BY
    S.StudentID, S.FullName;
```

100 %

Results | Messages

	StudentName	EnrolledCourses	TotalAmountPaid
1	Nikila	1	500.00
2	Nivedha	1	500.00
3	Nisha	1	700.00
4	Gani	1	700.00
5	Varshana	1	0.00
6	Appu	0	0.00
7	Sanju	0	0.00
8	Rohith	0	0.00
9	Vicky	0	0.00
10	Yogesh	0	0.00
11	Alice Johnson	4	0.00

17. **Find Course Pairs with the Same Instructor:**

- List pairs of courses that are **taught by the same instructor**.

```sql
-- Q17: Find Course Pairs with the Same Instructor
SELECT
    C1.CourseName AS Course1,
    C2.CourseName AS Course2,
    I.FullName AS InstructorName
FROM
    Courses C1
JOIN
    Courses C2 ON C1.InstructorID = C2.InstructorID
JOIN
    Instructors I ON C1.InstructorID = I.InstructorID
WHERE
    C1.CourseID < C2.CourseID;  -- To avoid duplicate pairs like (Course1, Course2) and (Course2, Course1)
```

100 %

Results | Messages

Course1	Course2	InstructorName

18. **List All Possible Student-Course Combinations:**

- Retrieve a **Cartesian product** of all students and courses (potential enrollments).

coding challenge 4...NTHINI\nisha (57))*

```sql
-- Q18: List All Possible Student-Course Combinations
SELECT
    S.FullName AS StudentName,
    C.CourseName AS CourseName
FROM
    Students S
CROSS JOIN
    Courses C;
```

100 %

Results | Messages

	StudentName	CourseName
1	Nikila	AI Fundamentals
2	Nivedha	AI Fundamentals
3	Nisha	AI Fundamentals
4	Gani	AI Fundamentals
5	Varshana	AI Fundamentals
6	Appu	AI Fundamentals
7	Sanju	AI Fundamentals
8	Rohith	AI Fundamentals
9	Vicky	AI Fundamentals
10	Yogesh	AI Fundamentals
11	Alice Johnson	AI Fundamentals
12	Nikila	Cloud Architecture
13	Nivedha	Cloud Architecture
14	Nisha	Cloud Architecture
15	Gani	Cloud Architecture
16	Varshana	Cloud Architecture
17	Appu	Cloud Architecture
18	Sanju	Cloud Architecture
19	Rohith	Cloud Architecture
20	Vicky	Cloud Architecture
21	Yogesh	Cloud Architecture
22	Alice Johnson	Cloud Architecture
23	Nikila	Ethical Hacking
24	Nivedha	Ethical Hacking
25	Nisha	Ethical Hacking

Query executed successfully.

19. **Determine the Instructor with the Highest Number of Students:**

- Find the **Instructor Name and the number of students enrolled in their courses**.

coding challenge 4...NTHINI\nisha (57))*

```sql
-- Q19: Determine the Instructor with the Highest Number of Students
SELECT TOP 1
    I.FullName AS InstructorName,
    COUNT(DISTINCT E.StudentID) AS NumberOfStudents
FROM
    Instructors I
JOIN
    Courses C ON I.InstructorID = C.InstructorID
JOIN
    Enrollments E ON C.CourseID = E.CourseID
GROUP BY
    I.InstructorID, I.FullName
ORDER BY
    NumberOfStudents DESC;
```

100 %

Results | Messages

	InstructorName	NumberOfStudents
1	Prof. John	3

20. **Trigger to Prevent Double Enrollment:**

- Create a **trigger** to prevent a student from enrolling in the same course more than once.

```sql
-- Trigger: Prevent Double Enrollment (Q20)
CREATE TRIGGER PreventDuplicateEnrollment
ON Enrollments
AFTER INSERT
AS
BEGIN
    ROLLBACK;
    PRINT 'Error: Student is already enrolled in this course';
END;
```

100 %

Messages

```
Commands completed successfully.

Completion time: 2025-03-27T16:52:11.1276582+05:30
```