1. University Course Registration System

1. University Course Registration System

Entities and Attributes

Student

- **StudentID** (PK)
- Name
- Email
- Major

Course

- **CourseID** (PK)
- Title
- Credits
- **Department** (FK)

Instructor

- **InstructorID** (PK)
- Name
- **DeptID** (FK)

Enrollment *(Associative/Relationship Entity)*

- **StudentID** (PK, FK)
- **CourseID** (PK, FK)
- Semester
- Grade

Department

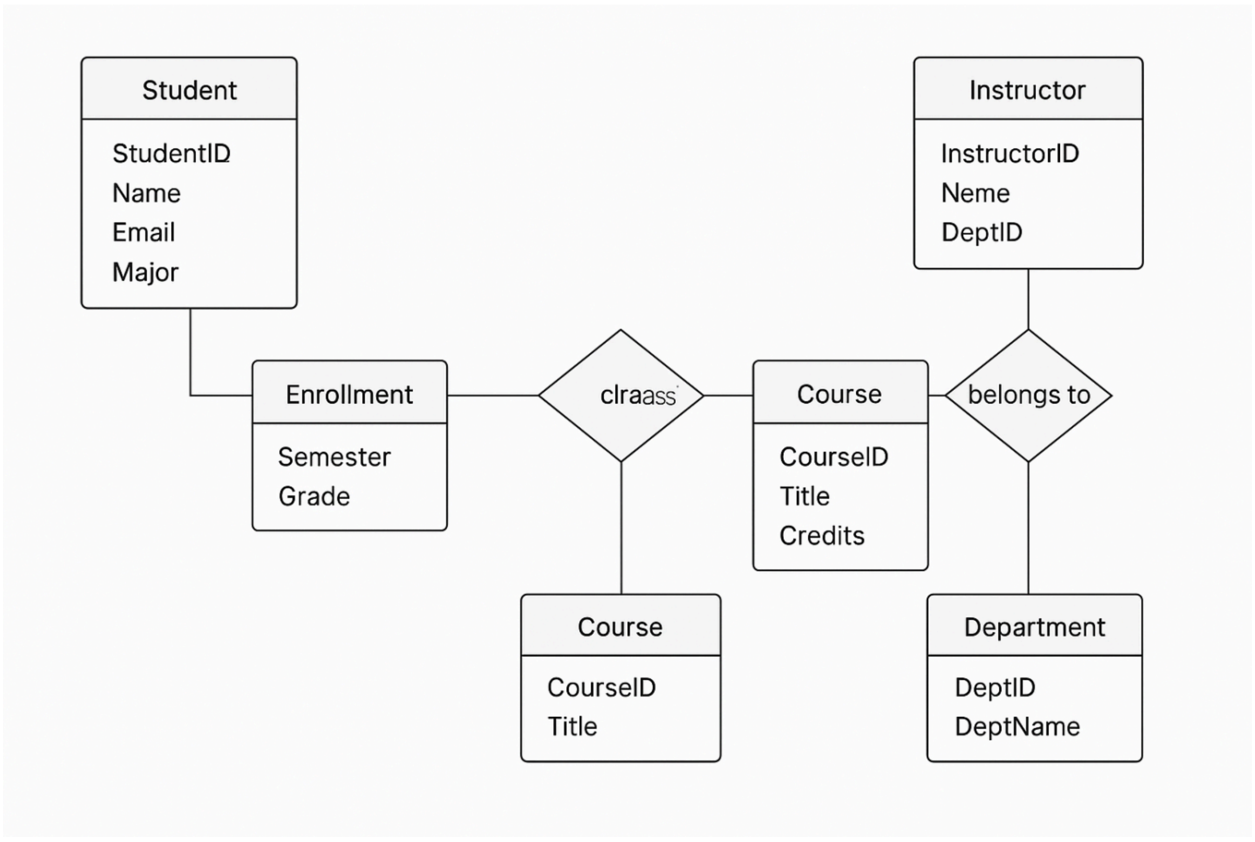
- **DeptID** (PK)
- DeptName

🔗 **Relationships**

1. **Student —< Enrollment >— Course**
   - *Many-to-Many*: A student can enroll in many courses, and each course can have many enrolled students.
   - Enrollment is a junction table with additional attributes like Semester and Grade.
2. **Instructor — teaches — Course**
   - *One-to-Many*: One instructor can teach many courses, but each course is taught by one instructor (unless you want to allow co-instructors, in which case it would be many-to-many with a junction table like CourseInstructor).
3. **Course — belongs to — Department**
   - *Many-to-One*: Many courses can belong to one department.
4. **Instructor — belongs to — Department**
   - *Many-to-One*: Each instructor is associated with one department.

✅ Primary Keys (PK) and Foreign Keys (FK) Summary

Entity	Primary Key	Foreign Keys
Student	StudentID	—
Course	CourseID	Department (→ Department.DeptID)
Instructor	InstructorID	DeptID (→ Department.DeptID)
Enrollment	StudentID + CourseID	StudentID, CourseID (→ respective PKs)
Department	DeptID	—



```
-- Table: Department

CREATE TABLE Department (

    DeptID INT PRIMARY KEY,

    DeptName VARCHAR(100) NOT NULL

);


-- Table: Instructor

CREATE TABLE Instructor (

    InstructorID INT PRIMARY KEY,

    Name VARCHAR(100) NOT NULL,

    Email VARCHAR(100), -- Assuming email is needed

    DeptID INT,

    FOREIGN KEY (DeptID) REFERENCES Department(DeptID)

);


-- Table: Course

CREATE TABLE Course (

    CourseID INT PRIMARY KEY,

    Title VARCHAR(100) NOT NULL,

    Credits INT NOT NULL,

    Department INT,

    FOREIGN KEY (Department) REFERENCES Department(DeptID)

);


-- Table: Student

CREATE TABLE Student (

    StudentID INT PRIMARY KEY,

    Name VARCHAR(100) NOT NULL,

    Email VARCHAR(100) NOT NULL,
    Major VARCHAR(100) NOT NULL
);

-- Table: Enrollment (Associative/Junction Table)
CREATE TABLE Enrollment (
    StudentID INT,
    CourseID INT,
    Semester VARCHAR(50),
    Grade CHAR(2),
    PRIMARY KEY (StudentID, CourseID),
    FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Course(CourseID)
);

-- Insert into Department
INSERT INTO Department (DeptID, DeptName)
VALUES
(1, 'Computer Science'),
(2, 'Mathematics'),
(3, 'Physics'),
(4, 'Biology');

-- Insert into Instructor
INSERT INTO Instructor (InstructorID, Name, Email, DeptID)
VALUES
(101, 'Dr. Smith', 'smith@university.edu', 1),
(102, 'Dr. Johnson', 'johnson@university.edu', 2),
(103, 'Dr. Brown', 'brown@university.edu', 3);
```

```
-- Insert into Course

INSERT INTO Course (CourseID, Title, Credits, Department)

VALUES

(201, 'CSE101', 4, 1),

(202, 'MATH201', 3, 2),

(203, 'PHY301', 3, 3);


-- Insert into Student

INSERT INTO Student (StudentID, Name, Email, Major)

VALUES

('1001', 'John Doe', 'john.doe@university.edu', 'Computer Science'),

('1002', 'Jane Smith', 'jane.smith@university.edu', 'Mathematics'),

('1003', 'Bob Johnson', 'bob.johnson@university.edu', 'Physics'),

('1005', 'Alice Brown', 'alice@university.edu', 'Biology'); -- As requested


-- Insert into Enrollment

INSERT INTO Enrollment (StudentID, CourseID, Semester, Grade)

VALUES

('1001', 201, 'Fall2024', 'A'),

('1002', 202, 'Fall2024', NULL),

('1003', 203, 'Fall2024', 'B');
```

UPDATE Commands

```
-- Update the grade of student 'S1001' in course 'CSE101' to 'A'
UPDATE Enrollment
SET Grade = 'A'
WHERE StudentID = '1001' AND CourseID = 201;
```

DELETE Commands

```
-- Delete all enrollments for student 'S1003' in the current semester (example
Fall2024)
DELETE FROM Enrollment
WHERE StudentID = '1003'
AND Semester = 'Fall2024';
```

SELECT Examples (Retrieval Commands)

```
-- Retrieve names and emails of all Computer Science students
SELECT Name, Email
FROM Student
WHERE Major = 'Computer Science';

-- Retrieve course titles and credits offered by Mathematics department
SELECT Title, Credits
FROM Course
WHERE Department = (SELECT DeptID FROM Department WHERE DeptName =
'Mathematics');
```

✅ **Summary:**

- INSERT adds initial data for Departments, Instructors, Courses, Students, and Enrollment.
- UPDATE modifies student grades.
- DELETE removes enrollments.
- SELECT retrieves necessary information.

AFTER DDL AND DML DCL COMMANDS WE CAN PERFORM THE BASIC OPERATIONS GIVEN BELOW:

Basic Retrieval Queries

1. Retrieve the names and email addresses of all students majoring in 'Computer Science'.

```
SELECT Name, Email

FROM Student

WHERE Major = 'Computer Science';
```

2. List the titles and credits of all courses offered by the 'Mathematics' department.

```
SELECT Title, Credits

FROM Course

WHERE Department = (

   SELECT DeptID

   FROM Department

   WHERE DeptName = 'Mathematics'

);
```

3. Show the names of all instructors who belong to the 'Physics' department.

```
SELECT Name

FROM Instructor

WHERE DeptID = (

   SELECT DeptID

   FROM Department

   WHERE DeptName = 'Physics'

);
```

🧾 Ambiguous Attributes & Aliasing

4. Retrieve a list of student names and the titles of the courses they are enrolled in, resolving any ambiguous column names using aliases.

   ```
   SELECT S.Name AS StudentName, C.Title AS CourseTitle

   FROM Student S

   JOIN Enrollment E ON S.StudentID = E.StudentID

   JOIN Course C ON E.CourseID = C.CourseID;
   ```

5. Display each course along with the name of the instructor teaching it, using table aliases to shorten query length.

   ```
   SELECT C.Title AS CourseTitle, I.Name AS InstructorName

   FROM Course C

   JOIN Instructor I ON C.Department = I.DeptID;
   ```

✳ Use of Asterisk & Set Operations

6. Select all attributes of students who have enrolled in any course.

   ```
   SELECT DISTINCT S.*

   FROM Student S

   JOIN Enrollment E ON S.StudentID = E.StudentID;
   ```

7. Retrieve a list of all unique course titles (no duplicates).

   ```
   SELECT DISTINCT Title

   FROM Course;
   ```

🔠 Pattern Matching

8. Find all students whose name starts with 'A'.

   ```
   SELECT *

   FROM Student

   WHERE Name LIKE 'A%';
   ```

9. List all instructors whose email ends with '@university.edu' (assuming email was stored for them).

   SELECT *

   FROM Instructor

   WHERE Email LIKE '%@university.edu';

📝 INSERT, DELETE, UPDATE

10. Insert a new student into the system with ID 'S1005', name 'Alice Brown', email 'alice@university.edu', and major 'Biology'.

    INSERT INTO Student (StudentID, Name, Email, Major)

    VALUES ('1006', 'Alice Brown', 'alice@university.edu', 'Biology');

11. Delete all enrollments for student 'S1003' in the current semester.

    DELETE FROM Enrollment

    WHERE StudentID = '1003'

    AND Semester = 'CurrentSemester'; -- Replace 'CurrentSemester' with the actual semester value

12. Update the grade of student 'S1001' in course 'CSE101' to 'A'.

    UPDATE Enrollment

    SET Grade = 'A'

    WHERE StudentID = '1001'

    AND CourseID = '201';

🧩 Complex Queries (Nested, Nulls, Co-Related)

13. Retrieve the names of students who are enrolled in a course taught by 'Dr. Smith'.

    SELECT DISTINCT S.Name

    FROM Student S

    JOIN Enrollment E ON S.StudentID = E.StudentID

    JOIN Course C ON E.CourseID = C.CourseID

```
JOIN Instructor I ON C.Department = I.DeptID
WHERE I.Name = 'Dr. Smith';
```

14. List all students who have never enrolled in any course.

```
SELECT S.Name
FROM Student S
LEFT JOIN Enrollment E ON S.StudentID = E.StudentID
WHERE E.StudentID IS NULL;
```

15. Find students who have received a NULL grade in any course.

```
SELECT DISTINCT S.Name
FROM Student S
JOIN Enrollment E ON S.StudentID = E.StudentID
WHERE E.Grade IS NULL;
```

16. Show the titles of courses that have never been enrolled in by any student.

```
SELECT Title
FROM Course
WHERE CourseID NOT IN (
   SELECT DISTINCT CourseID
   FROM Enrollment
);
```

17. Get the names of departments that do not have any courses associated with them.

```
SELECT DeptName
FROM Department
WHERE DeptID NOT IN (
    SELECT DISTINCT Department

    FROM Course

);
```

ALTER COMMANDS

Add a Phone column to Student

```
ALTER TABLE Student
ADD Phone VARCHAR(15);
```

Change the Credits column in Course to allow decimal values

```
ALTER TABLE Course
MODIFY Credits DECIMAL(4,2);
```

Add a UNIQUE constraint to Instructor Email

```
ALTER TABLE Instructor
ADD CONSTRAINT UC_InstructorEmail UNIQUE (Email);
```

Add a CHECK constraint to Grade in Enrollment (valid values: A, B, C, D, F, NULL)

```
ALTER TABLE Enrollment
ADD CONSTRAINT CHK_Grade
CHECK (Grade IN ('A', 'B', 'C', 'D', 'F'));
```

Rename column 'Department' in Course to 'DeptID' for consistency

```
ALTER TABLE Course
RENAME COLUMN Department TO DeptID;
```

Drop the Phone column if no longer needed

```
ALTER TABLE Student
DROP COLUMN Phone;
```

Add a default value to Semester in Enrollment

```
ALTER TABLE Enrollment
ALTER COLUMN Semester SET DEFAULT 'Fall2025';
```

1. Drop command

```
DROP TABLE Enrollment;
```

Drop a Column

```
ALTER TABLE Instructor
DROP COLUMN Email;
```

Drop a Foreign Key Constraint

```
ALTER TABLE Course
DROP FOREIGN KEY Course_ibfk_1;
```

Drop a Primary Key

```
ALTER TABLE Student
DROP PRIMARY KEY;
```

Drop a Unique Constraint

```
ALTER TABLE Instructor
DROP INDEX UC_InstructorEmail;
```

Drop a Check Constraint

```
ALTER TABLE Enrollment
DROP CONSTRAINT CHK_Grade;
```

Drop the Entire Database (Dangerous Operation!)

```
DROP DATABASE UniversityRegistration;
```

TRUNCATE

Truncate the Enrollment table

TRUNCATE TABLE Enrollment;

Truncate the Student table

Truncate the Instructor table

TRUNCATE TABLE Instructor;

Truncate the Course table

TRUNCATE TABLE Course;

Truncate the Department table

TRUNCATE TABLE Department;

RENAME

Rename a Table

```
ALTER TABLE Student RENAME TO Students;
RENAME TABLE Student TO Students;
```

Rename a Column

```
ALTER TABLE Department RENAME COLUMN DeptName TO DepartmentName;
ALTER TABLE Department CHANGE DeptName DepartmentName VARCHAR(100);
```

Rename Email to EmailAddress in Instructor Table

```
ALTER TABLE Instructor RENAME COLUMN Email TO EmailAddress;
ALTER TABLE Instructor CHANGE Email EmailAddress VARCHAR(100);
```

Rename Department column in Course to DeptID (for consistency)

```
ALTER TABLE Course RENAME COLUMN Department TO DeptID;
ALTER TABLE Course CHANGE Department DeptID INT;
```

COMMENT

Add a comment to the Student table

```
COMMENT ON TABLE Student IS 'Contains student personal and academic details';
```

Add a comment to the CourseID column in Course

```
COMMENT ON COLUMN Course.CourseID IS 'Primary key identifier for each course';
```

Add a comment to the Enrollment table

```
COMMENT ON TABLE Enrollment IS 'Junction table for student-course relationships including
semester and grade';
```

Add a comment to Grade column in Enrollment

```
COMMENT ON COLUMN Enrollment.Grade IS 'Letter grade received by the student (A-F)';
```

Add a comment to the Instructor table

```
COMMENT ON TABLE Instructor IS 'Stores instructor names, emails, and associated departments';
```

MySQL Alternative (Table/Column Comment Syntax):

```
CREATE TABLE Student (
   StudentID INT PRIMARY KEY COMMENT 'Unique ID for each student',
   Name VARCHAR(100) NOT NULL COMMENT 'Full name of the student',
   Email VARCHAR(100) NOT NULL COMMENT 'Student email address',
   Major VARCHAR(100) NOT NULL COMMENT 'Declared major'
) COMMENT = 'Contains student personal and academic details';
```

SQL INSERT Statement: Syntax and Example

1. Basic Syntax

```
INSERT INTO table_name (column1, column2, ..., columnN)
VALUES (value1, value2, ..., valueN);
```

Example 1: Insert a New Student

```
INSERT INTO Student (StudentID, Name, Email, Major)
VALUES ('1006', 'Alice Brown', 'alice@university.edu', 'Biology');
```

2. Basic Syntax 2:

```
INSERT INTO table_name VALUES (value1,value2, …..,valueN);
```

Example 1: Insert a New Student

```
INSERT INTO Student VALUES ('1006', 'Alice Brown', 'alice@university.edu', 'Biology');
```

SQL DELETE Statement: Syntax and Examples

Basic Syntax

DELETE FROM table_name WHERE condition;

Example 1: Delete a Specific Student

DELETE FROM Student
WHERE StudentID = '1001';

QUESTIONARIES:

1. Delete all students majoring in 'Biology'.
2. Delete all instructors not from the Computer Science department.
3. Delete courses with more than 3 credits.
4. Delete courses with fewer than 4 credits.
5. Delete courses with credits greater than or equal to 4.
6. Delete courses with credits less than or equal to 3.
7. Delete students with IDs between 1002 and 1003.
8. Delete enrollment records for CourseID in (201, 203).
9. Delete enrollment records where CourseID is not in (201, 202).
10. Delete enrollments where Grade is NULL.
11. Delete enrollments where Grade is NOT NULL.
12. Delete students whose names start with 'J'.
13. Delete instructors whose email does not end with '@university.edu'.
14. Delete students who are enrolled in at least one course.
15. Delete students who are not enrolled in any courses.

1. = (Equal)

DELETE FROM Student
WHERE Major = 'Biology';

2. != or <> (Not Equal)

DELETE FROM Instructor
WHERE DeptID != 1;

3. > (Greater Than)

DELETE FROM Course
WHERE Credits > 3;

4. < (Less Than)

DELETE FROM Course
WHERE Credits < 4;

5. >= (Greater Than or Equal To)

DELETE FROM Course
WHERE Credits >= 4;

6. <= (Less Than or Equal To)

```
DELETE FROM Course
WHERE Credits <= 3;
```

7. BETWEEN

```
DELETE FROM Student
WHERE StudentID BETWEEN 1002 AND 1003;
```

8. IN

```
DELETE FROM Enrollment
WHERE CourseID IN (201, 203);
```

9. NOT IN

```
DELETE FROM Enrollment
WHERE CourseID NOT IN (201, 202);
```

10. IS NULL

```
DELETE FROM Enrollment
WHERE Grade IS NULL;
```

11. IS NOT NULL

```
DELETE FROM Enrollment
WHERE Grade IS NOT NULL;
```

12. LIKE

```
DELETE FROM Student
WHERE Name LIKE 'J%';
```

13. NOT LIKE

```
DELETE FROM Instructor
WHERE Email NOT LIKE '%@university.edu';
```

14. EXISTS

```
DELETE FROM Student
WHERE EXISTS (
   SELECT 1 FROM Enrollment E WHERE E.StudentID = Student.StudentID
);
```

15. NOT EXISTS

```
DELETE FROM Student
WHERE NOT EXISTS (
   SELECT 1 FROM Enrollment E WHERE E.StudentID = Student.StudentID
);
```

SQL UPDATE Statement: Syntax and Example

Basic Syntax

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Example 1: Update a Student's Email

```
UPDATE Student
SET Email = 'john.doe@newdomain.edu'
WHERE StudentID = '1001';
```

QUESTIONARIES:

1. Update the email of student '1001' to 'updated.email@university.edu'.
2. Change the major to 'Undeclared' for all students not in 'Computer Science'.
3. Increase credits by 1 for courses with more than 3 credits.
4. Add ' - Basic' to course titles with less than 4 credits.
5. Set course credits to 4 where credits are 4 or more.
6. Set course credits to 3 where credits are 3 or fewer.
7. Change major to 'Interdisciplinary' for students with IDs between 1002 and 1003.
8. Set DeptID to 4 for instructors with IDs 101 and 103.
9. Assign DeptID = 1 to all instructors except 101 and 102.
10. Set grade to 'F' where it's currently NULL.
11. Set all non-null grades to 'B'.
12. Update email to 'default@university.edu' for students whose names start with 'A'.
13. Change major to 'General Studies' for students whose names don't contain 'Smith'.
14. Update major to 'Engineering' for students enrolled in course 201.
15. Set major to 'Prospective' for students not enrolled in any course.
16. Update grades A → A+, B → B+, others remain unchanged.

UPDATE Queries with All Possible Operators & Keywords

1. = (Equal)

```
UPDATE Student
SET Email = 'updated.email@university.edu'
WHERE StudentID = '1001';
```

2. != or <> (Not Equal)

```
UPDATE Student
SET Major = 'Undeclared'
WHERE Major <> 'Computer Science';
```

3. > (Greater Than)

```
UPDATE Course
SET Credits = Credits + 1
WHERE Credits > 3;
```

4. < (Less Than)

```
UPDATE Course
```

```
SET Title = CONCAT(Title, ' - Basic')
WHERE Credits < 4;
```

5. >= (Greater Than or Equal To)

```
UPDATE Course
SET Credits = 4
WHERE Credits >= 4;
```

6. <= (Less Than or Equal To)

```
UPDATE Course
SET Credits = 3
WHERE Credits <= 3;
```

7. BETWEEN

```
UPDATE Student
SET Major = 'Interdisciplinary'
WHERE StudentID BETWEEN '1002' AND '1003';
```

8. IN

```
UPDATE Instructor
SET DeptID = 4
WHERE InstructorID IN (101, 103);
```

9. NOT IN

```
UPDATE Instructor
SET DeptID = 1
WHERE InstructorID NOT IN (101, 102);
```

10. IS NULL

```
UPDATE Enrollment
SET Grade = 'F'
WHERE Grade IS NULL;
```

11. IS NOT NULL

```
UPDATE Enrollment
SET Grade = 'B'
WHERE Grade IS NOT NULL;
```

12. LIKE

```
UPDATE Student
SET Email = 'default@university.edu'
WHERE Name LIKE 'A%';
```

13. NOT LIKE

```
UPDATE Student
SET Major = 'General Studies'
WHERE Name NOT LIKE '%Smith%';
```

14. EXISTS

```
UPDATE Student
SET Major = 'Engineering'
WHERE EXISTS (
  SELECT 1 FROM Enrollment E
```

```
   WHERE E.StudentID = Student.StudentID AND E.CourseID = 201
);
```

15. NOT EXISTS

```
UPDATE Student
SET Major = 'Prospective'
WHERE NOT EXISTS (
   SELECT 1 FROM Enrollment E
   WHERE E.StudentID = Student.StudentID
);
```

16. CASE Statement (Conditional Update)

```
UPDATE Enrollment
SET Grade = CASE
   WHEN Grade = 'A' THEN 'A+'
   WHEN Grade = 'B' THEN 'B+'
   ELSE Grade
END;
```

SQL SELECT Statement: Syntax and Example

Basic Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

1. Display all student records.
2. List the names and emails of all students.
3. Find all students majoring in 'Computer Science'.
4. Show unique majors in the Student table.
5. List names and grades from the Enrollment table in descending grade order.
6. Find instructors whose email ends with '@university.edu'.
7. Count the number of students in each major.
8. Find the names of instructors from the Physics department.
9. List the names of students enrolled in any course.
10. List the students who have never enrolled in a course.
11. Show the course titles that no student has enrolled in.
12. List departments with no associated courses.
13. Show students with NULL grades.
14. List student names and course titles using joins.
15. Display each course title with the name of the instructor teaching it.

1. Select All records in the table

```
SELECT * FROM Student;
```

2. List the Names and Emails of all students

```
SELECT Name, Email FROM Student;
```

3. WHERE (Find all students majoring in ‘Computer Science’)

```
SELECT * FROM Student
WHERE Major = 'Computer Science';
```

4. Show unique majors in the student table.

```
SELECT DISTINCT Major FROM Student;
```

5. Find the names of instructors from the physics department

```
SELECT Name
FROM Instructor
WHERE DeptID = (
   SELECT DeptID
   FROM Department
   WHERE DeptName = 'Physics'
);
```

6. Retrieve the names of instructors in the department that offers the course titled 'MCA101'

```
SELECT Name
FROM Instructor
WHERE DeptID = (
   SELECT Department
   FROM Course
   WHERE Title = 'CSE101'
);
```

7. List the names of students who have never enrolled in any course

```
SELECT Name
FROM Student
WHERE StudentID NOT IN (
   SELECT StudentID
   FROM Enrollment
);
```

8. Show the titles of courses that no student has enrolled in.

```
SELECT Title
FROM Course
WHERE CourseID NOT IN (
   SELECT CourseID
   FROM Enrollment
);
```

9. Get the names of departments that do not offer any courses.

```
SELECT DeptName
FROM Department
WHERE DeptID NOT IN (
   SELECT Department
   FROM Course
);
```

10. Find all students whose grade is NULL in any course.

```
SELECT Name
FROM Student
WHERE StudentID IN (
   SELECT StudentID
   FROM Enrollment
   WHERE Grade IS NULL
);
```

11. Retrieve the email addresses of students enrolled in '2025' but with no assigned grade yet.

```
SELECT Email
FROM Student
WHERE StudentID IN (
   SELECT StudentID
   FROM Enrollment
   WHERE Semester = 'Fall2024' AND Grade IS NULL
);
```

12. Show students who have enrolled in at least one course.

```
SELECT Name
FROM Student S
WHERE EXISTS (
   SELECT 1
   FROM Enrollment E
   WHERE E.StudentID = S.StudentID
);
```

13. Show students who have never enrolled in a course.

```
SELECT Name
FROM Student S
WHERE NOT EXISTS (
   SELECT 1
   FROM Enrollment E
   WHERE E.StudentID = S.StudentID
);
```

14. BETWEEN

Q: List names of students with IDs **between 1002 and 1003** and are enrolled in some course.

```
SELECT Name
FROM Student
WHERE StudentID BETWEEN 1002 AND 1003
AND StudentID IN (
   SELECT StudentID
   FROM Enrollment
);
```

15. LIKE

Q: Find all courses enrolled by students whose name starts with 'J'

```
SELECT DISTINCT C.Title
FROM Course C
JOIN Enrollment E ON C.CourseID = E.CourseID
JOIN Student S ON E.StudentID = S.StudentID
WHERE S.Name LIKE 'J%';
```

Ambiguity of attribute names in SQL typically arises when multiple tables in a query have **columns with the same name**, such as ID, Name, or DeptID. SQL needs clarification on which table’s column you're referencing—this is where **table aliases and qualified column names** come in.

Here’s a concise explanation followed by examples from your **University Course Registration System**.

Ambiguity occurs when a column name appears in multiple tables involved in a query, and the SQL engine doesn't know which one you mean.

Solution: Use Table Aliases or Fully Qualify Column Names
Example Scenario:
You join the Student, Enrollment, and Course tables. All three tables may have columns like StudentID, CourseID, or Name.

1. Ambiguous Query (Incorrect / Risky):

```
SELECT Name, Title
FROM Student, Enrollment, Course
WHERE Student.StudentID = Enrollment.StudentID
AND Enrollment.CourseID = Course.CourseID;
```

2. Corrected with Table Aliases (Clear & Safe)

```
SELECT S.Name AS StudentName, C.Title AS CourseTitle
FROM Student S
JOIN Enrollment E ON S.StudentID = E.StudentID
JOIN Course C ON E.CourseID = C.CourseID;
```

CORELATED NESTED QUERIES

1. **Q:** List students who have received a grade higher than the **average grade (non-null)** in their enrolled course.

```
SELECT S.Name
```

```
FROM Student S
JOIN Enrollment E1 ON S.StudentID = E1.StudentID
WHERE E1.Grade IS NOT NULL AND E1.Grade > (
   SELECT AVG(
      CASE E2.Grade
         WHEN 'A' THEN 4
         WHEN 'B' THEN 3
         WHEN 'C' THEN 2
         WHEN 'D' THEN 1
         WHEN 'F' THEN 0
      END
   )
   FROM Enrollment E2
   WHERE E2.CourseID = E1.CourseID
   AND E2.Grade IS NOT NULL
);
```

2. **Q:** List departments where **at least one course** has **fewer than 3 credits**.

```
SELECT DISTINCT D.DeptName
FROM Department D
WHERE EXISTS (
   SELECT 1
   FROM Course C
   WHERE C.Department = D.DeptID AND C.Credits < 3
);
```

3. **Q:** Show students who are **enrolled in more than one course**.

```
SELECT S.Name
FROM Student S
WHERE (
   SELECT COUNT(*)
   FROM Enrollment E
   WHERE E.StudentID = S.StudentID
) > 1;
```

4. **Q:** Find students who are enrolled in **every course offered by their major's department**.

```
SELECT S.Name
FROM Student S
WHERE NOT EXISTS (
   SELECT C.CourseID
   FROM Course C
   WHERE C.Department = (SELECT DeptID FROM Department WHERE DeptName = S.Major)
   EXCEPT
   SELECT E.CourseID
   FROM Enrollment E
   WHERE E.StudentID = S.StudentID
);
```

AGGREGATE FUNCTIONS:

These are great for analyzing grouped data using functions like

- `COUNT()`
- `AVG()`

- `SUM()`
- `MIN()`
- `MAX()`

1. **Q:** Count the number of students in each major.

```
SELECT Major, COUNT(*) AS StudentCount
FROM Student
GROUP BY Major;
```

2. **Q:** What is the average number of credits for all courses?

```
SELECT AVG(Credits) AS AverageCredits
FROM Course;
```

3. Q: What is the total number of credits for all courses?

```
SELECT SUM(Credits) AS TotalCredits
FROM Course;
```

4. Q: Show the maximum and minimum credits assigned to any course.

```
SELECT MAX(Credits) AS MaxCredits, MIN(Credits) AS MinCredits
FROM Course;
```

GROUPING WITH SELECT

For analyzing and summarizing data

1. **Q: Count the number of students in each major.**
```
SELECT Major, COUNT(*) AS StudentCount
FROM Student
GROUP BY Major;
```

2. Q: Count the number of enrollments per semester.
```
SELECT Semester, COUNT(*) AS EnrollmentCount
FROM Enrollment
GROUP BY Semester;
```

3. Q: List departments that offer **more than 1 course**.
```
SELECT Department, COUNT(*) AS CourseCount
FROM Course
GROUP BY Department
HAVING COUNT(*) > 1;
```

4. Q: Find courses with **more than 1 enrolled student**
```
SELECT CourseID, COUNT(StudentID) AS EnrollmentCount
FROM Enrollment
GROUP BY CourseID
HAVING COUNT(StudentID) > 1;
```

5. Q: Show semesters with **at least 2 enrollments**.
```
SELECT Semester, COUNT(*) AS Enrollments
FROM Enrollment
GROUP BY Semester
HAVING COUNT(*) >= 2;
```