/*Question -- Building a database for an Education Institute. The system stores information about students, courses, enrollments,

instructors, and grades. You are required to create tables, insert data, and run queries ranging from basic

SELECTs to advanced aggregations and window functions.

Table Creation

Create the following tables with appropriate primary and foreign keys:

Students (student_id, name, email, dob, gender)

Instructors (instructor_id, name, email, department)

Courses (course_id, course_name, instructor_id, credits)

Enrollments (enrollment_id, student_id, course_id, enroll_date)

Grades (grade_id, enrollment_id, grade)

Add relevant constraints like NOT NULL, UNIQUE, and FOREIGN KEY.

Insert Sample Data

Insert at least 5 records per table to simulate real-world usage.

Basic SQL Queries

Show the names of all students.

List all students born after 2002.

Display all courses taught by instructors from the "Computer Science" department.

Show courses with more than 3 credits, sorted by credit descending.

Aggregations and GROUP BY

Find the total number of students enrolled in each course.

Find the average grade received per course.

Show the number of male and female students.

JOIN Queries

List students with their enrolled course names.

Show course names along with the instructor who teaches them.

Display student names with the grade they received in each course.

Subqueries

Find students who are enrolled in all courses taught by a specific instructor.

Find courses that have the highest number of students.

List students who scored above the average grade in any course.

Window Functions

Rank students in each course based on their grade (highest to lowest).

Calculate the cumulative number of enrollments over time.

Show the average grade of each student across courses, and compare it with the overall average grade using LAG() or AVG() OVER().

/* create an EducationInstitude database*/

CREATE DATABASE EducationInstitudeDB;

SHOW DATABASES;

USE EducationInstitudeDB;

```
# CREATE TABLE Students
CREATE TABLE Students (
    student_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    dob DATE NOT NULL,
    gender ENUM('Male', 'Female', 'Other') NOT NULL
);

CREATE TABLE Instructors (
    instructor_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    department VARCHAR(100) NOT NULL
);

CREATE TABLE Courses (
    course_id INT PRIMARY KEY AUTO_INCREMENT,
    course_name VARCHAR(100) NOT NULL,
    instructor_id INT,
    credits INT CHECK(credits > 0),
    FOREIGN KEY (instructor_id) REFERENCES Instructors(instructor_id)
);
```

```
CREATE TABLE Enrollments (
   enrollment_id INT PRIMARY KEY AUTO_INCREMENT,
   student_id INT,
   course_id INT,
   enroll_date DATE NOT NULL,
   FOREIGN KEY (student_id) REFERENCES Students(student_id) ,
   FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);

CREATE TABLE Grades (
   grade_id INT PRIMARY KEY AUTO_INCREMENT,
   enrollment_id INT,
   grade DECIMAL(4,2) CHECK(grade BETWEEN 0 AND 100),
   FOREIGN KEY (enrollment_id) REFERENCES Enrollments(enrollment_id)
);



# Insert Students
INSERT INTO Students (name, email, dob, gender) VALUES
('Dikshya', 'dikshya@gmail.com', '1995-03-15', 'Female'),
('Samir KC', 'samir@gmail.com', '1998-07-21', 'Male'),
('Sunil Khatri', 'sunil@gmail.com', '1993-05-10', 'Male'),
('Dina Shakya', 'dina@gmail.com', '2000-12-05', 'Female'),
('Roshan Adhakari', 'roshan@gmail.com', '2002-11-30', 'Male');

#  Insert Instructors
```

```
INSERT INTO Instructors (name, email, department) VALUES
('Dr. willams Smit', 'willamssmit@gmail.com', 'Mathematics'),
('Dr. Sunita Bista', 'sunita@gmail.com', 'Computer Science'),
('Dr. jon Davis', 'jondavis@gmail.com', 'Computer Science'),
('Dr. James Wilson', 'jamwilson@gmail.com', 'Computer Science');

# Insert Courses
INSERT INTO Courses (course_name, instructor_id, credits) VALUES
('Statistical Foundation', 1, 3),
('Machine Learning', 1, 4),
('Programming Foundation', 2, 3),
('Database Management', 3, 5),
('Data Structures', 4, 4);

# Insert Enrollments
INSERT INTO Enrollments (student_id, course_id, enroll_date) VALUES
(1, 1, '2024-01-10'),
(1, 2, '2024-01-11'),
(2, 1, '2024-01-12'),
(3, 3, '2024-01-13'),
(4, 4, '2024-01-14');

# Insert Grades
INSERT INTO Grades (enrollment_id, grade) VALUES
(1, 86.5),
(2, 92.0),
```

```
(3, 75.5),
(4, 90.0),
(5, 95.0);

# Show the names of all students.
# List all students born after 2002.
# Display all courses taught by instructors from the "Computer Science" department.
# Show courses with more than 3 credits, sorted by credit descending.

# Show names of all students
SELECT name FROM Students;

 # List all students born after 2002
SELECT name FROM Students WHERE dob > '2002-11-30';

 # Display all courses taught by instructors from "Computer Science" department
SELECT course_name FROM Courses
JOIN Instructors ON Courses.instructor_id = Instructors.instructor_id
WHERE department = 'Computer Science';

 # Show courses with more than 3 credits, sorted by credits descending
SELECT course_name, credits FROM Courses WHERE credits > 3 ORDER BY credits DESC;

# Aggregations and GROUP BY
# Find the total number of students enrolled in each course.
# Find the average grade received per course.
```

Show the number of male and female students.

Find the total number of students enrolled in each course

```
SELECT course_name, COUNT(enrollment_id) AS total_students
FROM Enrollments
JOIN Courses ON Enrollments.course_id = Courses.course_id
GROUP BY course_name;
```

Find the average grade received per course

```
SELECT course_name, AVG(grade) AS avg_grade
FROM Grades
JOIN Enrollments ON Grades.enrollment_id = Enrollments.enrollment_id
JOIN Courses ON Enrollments.course_id = Courses.course_id
GROUP BY course_name;
```

Show the number of male and female students

```
SELECT gender, COUNT(*) AS count FROM Students GROUP BY gender;
```

JOIN Queries

List students with their enrolled course names.

Show course names along with the instructor who teaches them.

Display student names with the grade they received in each course.

List students with their enrolled course names

```
SELECT Students.name AS Student, Courses.course_name AS Course
FROM Students
```

```
JOIN Enrollments ON Students.student_id = Enrollments.student_id
JOIN Courses ON Enrollments.course_id = Courses.course_id;

# Show course names along with the instructor who teaches them
SELECT Courses.course_name, Instructors.name AS Instructor
FROM Courses
JOIN Instructors ON Courses.instructor_id = Instructors.instructor_id;

# Display student names with the grade they received in each course
SELECT Students.name, Courses.course_name, Grades.grade
FROM Students
JOIN Enrollments ON Students.student_id = Enrollments.student_id
JOIN Grades ON Enrollments.enrollment_id = Grades.enrollment_id
JOIN Courses ON Enrollments.course_id = Courses.course_id;

# Subqueries
# Find students who are enrolled in all courses taught by a specific instructor.
# Find courses that have the highest number of students.
# List students who scored above the average grade in any course.

# Find students who are enrolled in all courses taught by a specific instructor
SELECT name FROM Students WHERE student_id IN (
   SELECT student_id FROM Enrollments WHERE course_id IN (
      SELECT course_id FROM Courses WHERE instructor_id = 1
   )
);
```

```
# Find courses that have the highest number of students
SELECT course_name FROM Courses WHERE course_id = (
    SELECT course_id FROM Enrollments
    GROUP BY course_id
    ORDER BY COUNT(student_id) DESC
    LIMIT 1
);

# List students who scored above the average grade in any course
SELECT DISTINCT Students.name
FROM Students
JOIN Enrollments ON Students.student_id = Enrollments.student_id
JOIN Grades ON Enrollments.enrollment_id = Grades.enrollment_id
WHERE grade > (SELECT AVG(grade) FROM Grades);

# Window Functions
# Rank students in each course based on their grade (highest to lowest).
# Calculate the cumulative number of enrollments over time.
# Show the average grade of each student across courses, and compare it with the overall average grade using LAG() or AVG() OVER().

# Rank students in each course based on their grade (highest to lowest)
SELECT Students.name, Courses.course_name, Grades.grade,
      RANK() OVER(PARTITION BY Courses.course_name ORDER BY Grades.grade DESC)
FROM Students
```

```
JOIN Enrollments ON Students.student_id = Enrollments.student_id

JOIN Grades ON Enrollments.enrollment_id = Grades.enrollment_id

JOIN Courses ON Enrollments.course_id = Courses.course_id;


# Calculate the cumulative number of enrollments over time

SELECT enroll_date, COUNT(enrollment_id) OVER (ORDER BY enroll_date) AS
cumulative_enrollments

FROM Enrollments;


# Show the average grade of each student across courses and compare it with the overall
average grade

SELECT Students.name, AVG(Grades.grade) OVER(PARTITION BY Students.student_id) AS
student_avg_grade,

    AVG(Grades.grade) OVER() AS overall_avg_grade

FROM Students

JOIN Enrollments ON Students.student_id = Enrollments.student_id

JOIN Grades ON Enrollments.enrollment_id = Grades.enrollment_id;
```