Task 1. Database Design

1. Create the database named "SISDB"

   CREATE DATABASE SISDB;

2. Define the schema for the Students, Courses, Enrollments, Teacher, and Payments tables based on the provided schema. Write SQL scripts to create the mentioned tables with appropriate data types, constraints, and relationships.

   a. Students

      CREATE TABLE Students (

         student_id INT PRIMARY KEY,

         first_name VARCHAR(255),

         last_name VARCHAR(255),

         date_of_birth DATE,

         email VARCHAR(255),

         phone_number VARCHAR(15)

      );

   b. Courses

      CREATE TABLE Courses (

-> course_id INT PRIMARY KEY NOT NULL,

-> course_name LONGTEXT,

-> credits INT,

-> teacher_id INT,

-> FOREIGN KEY (teacher_id) REFERENCES Teacher(teacher_id)

->);

c. Enrollments

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

d. Teacher

```
CREATE TABLE Teacher (
->  teacher_id INT PRIMARY KEY NOT NULL,
->  first_name TEXT,
->  last_name TEXT,
->  email TEXT
-> );
```

e. Payments

```
CREATE TABLE Payments (
->  payment_id INT PRIMARY KEY NOT NULL,
->  student_id INT,
->  amount DECIMAL(10, 2),
->  payment_date DATE,
->  FOREIGN KEY (student_id) REFERENCES Students(student_id)
-> );
```

3. Create an ERD (Entity Relationship Diagram) for the database.

4. Create appropriate Primary Key and Foreign Key constraints for referential integrity.

5. Insert at least 10 sample records into each of the following tables.

i. Students

```
INSERT INTO Students (student_id, first_name, last_name, date_of_birth, email,
phone_number)
-> VALUES
-> (1, 'John', 'Doe', '1995-08-15', 'john.doe@example.com', '1234567890'),
-> (2, 'Jane', 'Smith', '1998-03-22', 'jane.smith@example.com', '9876543210'),
-> (3, 'Emily', 'Johnson', '1997-05-10', 'emily.j@example.com', '3456789012'),
-> (4, 'Alex', 'Smith', '1996-12-03', 'alex.smith@example.com', '5678901234'),
-> (5, 'Sophia', 'Williams', '1999-08-28', 'sophia.w@example.com', '7890123456'),
-> (6, 'Michael', 'Miller', '1994-04-17', 'michael.m@example.com', '8901234567'),
-> (7, 'Olivia', 'Brown', '1998-11-05', 'olivia.b@example.com', '9012345678'),
-> (8, 'Daniel', 'Jones', '1995-02-22', 'daniel.j@example.com', '1234509876'),
-> (9, 'Ava', 'Davis', '1996-09-14', 'ava.d@example.com', '2345678901'),
-> (10, 'Ethan', 'Garcia', '1997-06-30', 'ethan.g@example.com', '3456789012');
```

ii. Courses

```
INSERT INTO Courses (course_id, course_name, credits, teacher_id)
-> VALUES
-> (1, 'Introduction to Programming', 3, 1),
-> (2, 'Mathematics 101', 4, 2),
-> (3, 'Data Structures and Algorithms', 4, 1),
-> (4, 'Database Management Systems', 3, 2),
-> (5, 'Web Development Fundamentals', 3, 1),
-> (6, 'Linear Algebra', 4, 3),
```

-> (7, 'Software Engineering Principles', 3, 4),

-> (8, 'Computer Networks', 4, 2),

-> (9, 'Artificial Intelligence Basics', 3, 3),

-> (10, 'Operating Systems', 4, 4);

iii. Enrollments

INSERT INTO Enrollments (enrollment_id, student_id, course_id, enrollment_date)

-> VALUES

-> (1, 1, 1, '2023-01-10'),

-> (2, 2, 2, '2023-02-15'),

-> (3, 3, 3, '2023-02-25'),

-> (4, 4, 4, '2023-03-18'),

-> (5, 5, 5, '2023-04-20'),

-> (6, 6, 6, '2023-05-14'),

-> (7, 7, 7, '2023-06-10'),

-> (8, 8, 8, '2023-07-22'),

-> (9, 9, 9, '2023-08-28'),

-> (10, 10, 10, '2023-09-15');

iv. Teacher

INSERT INTO Teacher (teacher_id, first_name, last_name, email)

-> VALUES

-> (1, 'Professor', 'Johnson', 'prof.johnson@example.com'),

-> (2, 'Dr.', 'Williams', 'dr.williams@example.com'),

-> (3, 'Dr.', 'Clark', 'dr.clark@example.com'),

-> (4, 'Professor', 'Taylor', 'prof.taylor@example.com'),

-> (5, 'Ms.', 'Anderson', 'ms.anderson@example.com'),

-> (6, 'Mr.', 'Martinez', 'mr.martinez@example.com'),

-> (7, 'Dr.', 'Moore', 'dr.moore@example.com'),

-> (8, 'Professor', 'Wright', 'prof.wright@example.com'),

-> (9, 'Ms.', 'Perez', 'ms.perez@example.com'),

-> (10, 'Mr.', 'Hill', 'mr.hill@example.com');

v. Payments

INSERT INTO Payments (payment_id, student_id, amount, payment_date)

-> VALUES

-> (1, 1, 500.00, '2023-01-15'),

-> (2, 2, 600.00, '2023-02-20'),

-> (3, 3, 750.00, '2023-03-12'),

-> (4, 4, 600.00, '2023-04-05'),

-> (5, 5, 450.00, '2023-05-18'),

-> (6, 6, 800.00, '2023-06-22'),

-> (7, 7, 550.00, '2023-07-14'),

-> (8, 8, 700.00, '2023-08-30'),

-> (9, 9, 900.00, '2023-09-08'),

-> (10, 10, 650.00, '2023-10-17');

Tasks 2: Select, Where, Between, AND, LIKE:

1. Write an SQL query to insert a new student into the "Students" table with the following details:

a. First Name: John

b. Last Name: Doe

c. Date of Birth: 1995-08-15

d. Email: john.doe@example.com

e. Phone Number: 1234567890

```
INSERT INTO Students (student_id, first_name, last_name, date_of_birth, email, phone_number)
-> VALUES (11,'John', 'Doe', '1995-08-15', 'john.doe@example.com', '1234567890');
```

2. Write an SQL query to enroll a student in a course. Choose an existing student and course and insert a record into the "Enrollments" table with the enrollment date.

```
INSERT INTO Enrollments (enrollment_id,student_id, course_id, enrollment_date)
-> VALUES (4,4,4 , '2023-01-20');
```

3. Update the email address of a specific teacher in the "Teacher" table. Choose any teacher and modify their email address.

```
UPDATE Teacher
    -> SET email = 'new_email@example.com'
    -> WHERE teacher_id = 1;
```

4. Write an SQL query to delete a specific enrollment record from the "Enrollments" table. Select an enrollment record based on the student and course.

```
DELETE FROM Enrollments
-> WHERE student_id = 1 AND course_id = 1;
```

5. Update the "Courses" table to assign a specific teacher to a course. Choose any course and teacher from the respective tables.

```
UPDATE Courses
-> SET teacher_id = 2
-> WHERE course_id = 2;
```

6. Delete a specific student from the "Students" table and remove all their enrollment records from the "Enrollments" table. Be sure to maintain referential integrity.

```
> DELETE FROM Students
-> WHERE student_id = 1;
```

7. Update the payment amount for a specific payment record in the "Payments" table. Choose any payment record and modify the payment amount.

```
UPDATE Payments
SET amount = 150.00 -- Adjust the new payment amount as needed
WHERE payment_id = 101;
```

Task 3. Aggregate functions, Having, Order By, GroupBy and Joins:

1. Write an SQL query to calculate the total payments made by a specific student. You will need to join the "Payments" table with the "Students" table based on the student's ID.

```
SELECT
    ->      s.student_id,
    ->      s.first_name,
    ->      s.last_name,
    ->      SUM(p.amount) AS total_payments
    -> FROM
    ->      Students s
    -> JOIN
    ->      Payments p ON s.student_id = p.student_id
    -> WHERE
    ->      s.student_id = 1
    -> GROUP BY
    ->      s.student_id, s.first_name, s.last_name;
```

2. Write an SQL query to retrieve a list of courses along with the count of students enrolled in each course. Use a JOIN operation between the "Courses" table and the "Enrollments" table.

```
SELECT
    ->      c.course_id,
    ->      c.course_name,
    ->      COUNT(e.student_id) AS total_students_enrolled
    -> FROM
    ->      Courses c
    -> JOIN
```

```
    ->      Enrollments e ON c.course_id = e.course_id
    -> GROUP BY
    ->      c.course_id, c.course_name;
```

3. Write an SQL query to find the names of students who have not enrolled in any course. Use a LEFT JOIN between the "Students" table and the "Enrollments" table to identify students without enrollments.

```
SELECT
    ->      s.student_id,
    ->      s.first_name,
    ->      s.last_name
    -> FROM
    ->      Students s
    -> LEFT JOIN
    ->      Enrollments e ON s.student_id = e.student_id
    -> WHERE
    ->      e.enrollment_id IS NULL;
```

4. Write an SQL query to retrieve the first name, last name of students, and the names of the courses they are enrolled in. Use JOIN operations between the "Students" table and the "Enrollments" and "Courses" tables.

```
SELECT
    ->      s.first_name,
    ->      s.last_name,
    ->      c.course_name
    -> FROM
```

```
    ->      Students s
    -> JOIN
    ->      Enrollments e ON s.student_id = e.student_id
    -> JOIN
    ->      Courses c ON e.course_id = c.course_id;
```

5. Create a query to list the names of teachers and the courses they are assigned to. Join the "Teacher" table with the "Courses" table.

```
SELECT
    ->      t.first_name AS teacher_first_name,
    ->      t.last_name AS teacher_last_name,
    ->      c.course_name
    -> FROM
    ->      Teacher t
    -> JOIN
    ->      Courses c ON t.teacher_id = c.teacher_id;
```

6. Retrieve a list of students and their enrollment dates for a specific course. You'll need to join the "Students" table with the "Enrollments" and "Courses" tables.

```
SELECT
    ->      s.first_name,
    ->      s.last_name,
    ->      e.enrollment_date
    -> FROM
    ->      Students s
```

```
-> JOIN
->      Enrollments e ON s.student_id = e.student_id
-> JOIN
->      Courses c ON e.course_id = c.course_id
-> WHERE
->      c.course_name = ' Database Management Systems';
```

7. Find the names of students who have not made any payments. Use a LEFT JOIN between the "Students" table and the "Payments" table and filter for students with NULL payment records.

```
SELECT
->      s.first_name,
->      s.last_name
-> FROM
->      Students s
-> LEFT JOIN
->      Payments p ON s.student_id = p.student_id
-> WHERE
->      p.payment_id IS NULL;
```

8. Write a query to identify courses that have no enrollments. You'll need to use a LEFT JOIN between the "Courses" table and the "Enrollments" table and filter for courses with NULL enrollment records.

```
SELECT
->      c.course_name
-> FROM
```

```
    ->      Courses c
    -> LEFT JOIN
    ->      Enrollments e ON c.course_id = e.course_id
    -> WHERE
    ->      e.enrollment_id IS NULL;
```

9. Identify students who are enrolled in more than one course. Use a self-join on the "Enrollments" table to find students with multiple enrollment records.

```
SELECT
    ->      s.student_id,
    ->      s.first_name,
    ->      s.last_name
    -> FROM
    ->      Students s
    -> JOIN
    ->      Enrollments e1 ON s.student_id = e1.student_id
    -> JOIN
    ->      Enrollments e2 ON s.student_id = e2.student_id
    -> WHERE
    ->      e1.enrollment_id <> e2.enrollment_id;
```

10. Find teachers who are not assigned to any courses. Use a LEFT JOIN between the "Teacher" table and the "Courses" table and filter for teachers with NULL course assignments.

```
mysql> SELECT
```

```
->      t.teacher_id,
->      t.first_name,
->      t.last_name
-> FROM
->      Teacher t
-> LEFT JOIN
->      Courses c ON t.teacher_id = c.teacher_id
-> WHERE
->      c.course_id IS NULL;
```

Task 4. Subquery and its type:

1. Write an SQL query to calculate the average number of students enrolled in each course. Use aggregate functions and subqueries to achieve this.

```
SELECT
    ->      c.course_id,
    ->      c.course_name,
    ->      AVG(enrollment_count) AS average_students
    -> FROM
    ->      Courses c
    -> LEFT JOIN
    ->      (
    ->          SELECT
    ->              course_id,
    ->              COUNT(DISTINCT student_id) AS enrollment_count
    ->          FROM
    ->              Enrollments
    ->          GROUP BY
    ->              course_id
    ->      ) e ON c.course_id = e.course_id
    -> GROUP BY
    ->      c.course_id, c.course_name;
```

2. Identify the student(s) who made the highest payment. Use a subquery to find the maximum payment amount and then retrieve the student(s) associated with that amount.

```
SELECT
```

```
->      p.student_id,
->      s.first_name,
->      s.last_name,
->      MAX(p.amount) AS highest_payment
-> FROM
->      Payments p
-> JOIN
->      Students s ON p.student_id = s.student_id
-> WHERE
->      p.amount = (SELECT MAX(amount) FROM Payments)
-> GROUP BY
->      p.student_id, s.first_name, s.last_name;
```

3. Retrieve a list of courses with the highest number of enrollments. Use subqueries to find the course(s) with the maximum enrollment count.

```
SELECT
->      c.course_id,
->      c.course_name,
->      COUNT(e.enrollment_id) AS enrollment_count
-> FROM
->      Courses c
-> LEFT JOIN
->      Enrollments e ON c.course_id = e.course_id
-> GROUP BY
->      c.course_id, c.course_name
-> HAVING
```

-> COUNT(e.enrollment_id) = (SELECT MAX(enrollment_count) FROM (SELECT course_id, COUNT(enrollment_id) AS enrollment_count FROM Enrollments GROUP BY course_id) AS max_enrollments);

4. Calculate the total payments made to courses taught by each teacher. Use subqueries to sum payments for each teacher's courses.

```
SELECT
    -> t.teacher_id,
    -> t.first_name,
    -> t.last_name,
    -> SUM(p.amount) AS total_payments
    -> FROM
    -> Teacher t
    -> JOIN
    -> Courses c ON t.teacher_id = c.teacher_id
    -> LEFT JOIN
    -> Enrollments e ON c.course_id = e.course_id
    -> LEFT JOIN
    -> Payments p ON e.enrollment_id = p.enrollment_id
    -> GROUP BY
    -> t.teacher_id, t.first_name, t.last_name;
```

5. Identify students who are enrolled in all available courses. Use subqueries to compare a student's enrollments with the total number of courses.

```
SELECT
```

```
    ->      s.student_id,
    ->      s.first_name,
    ->      s.last_name
    -> FROM
    ->      Students s
    -> WHERE
    ->      (SELECT COUNT(DISTINCT e.course_id) FROM Enrollments e WHERE e.student_id =
s.student_id) =
    ->      (SELECT COUNT(DISTINCT course_id) FROM Courses);
```

6. Retrieve the names of teachers who have not been assigned to any courses. Use subqueries to find teachers with no course assignments.

```
SELECT
    ->      teacher_id,
    ->      first_name,
    ->      last_name
    -> FROM
    ->      Teacher
    -> WHERE
    ->      teacher_id NOT IN (SELECT DISTINCT teacher_id FROM Courses);
```

7. Calculate the average age of all students. Use subqueries to calculate the age of each student based on their date of birth.

```
SELECT
    ->      AVG(TIMESTAMPDIFF(YEAR, date_of_birth, CURDATE())) AS average_age
    -> FROM
    ->      Students;
```

8. Identify courses with no enrollments. Use subqueries to find courses without enrollment records.

```
mysql> SELECT
    ->      course_id,
    ->      course_name
    -> FROM
    ->      Courses
    -> WHERE
    ->      course_id NOT IN (SELECT DISTINCT course_id FROM Enrollments);
```

9. Calculate the total payments made by each student for each course they are enrolled in. Use subqueries and aggregate functions to sum payments.

```
SELECT
    ->      s.student_id,
    ->      s.first_name,
    ->      s.last_name,
    ->      c.course_id,
    ->      c.course_name,
    ->      SUM(p.amount) AS total_payments
    -> FROM
```

```
    ->      Students s
    -> JOIN
    ->      Enrollments e ON s.student_id = e.student_id
    -> JOIN
    ->      Courses c ON e.course_id = c.course_id
    -> LEFT JOIN
    ->      Payments p ON s.student_id = p.student_id AND e.course_id = p.course_id
    -> GROUP BY
    ->      s.student_id, c.course_id;
```

10. Identify students who have made more than one payment. Use subqueries and aggregate functions to count payments per student and filter for those with counts greater than one.

```
SELECT
    ->      s.student_id,
    ->      s.first_name,
    ->      s.last_name
    -> FROM
    ->      Students s
    -> JOIN
    ->      Payments p ON s.student_id = p.student_id
    -> GROUP BY
    ->      s.student_id
    -> HAVING
    ->      COUNT(p.payment_id) > 1;
```

11. Write an SQL query to calculate the total payments made by each student. Join the "Students" table with the "Payments" table and use GROUP BY to calculate the sum of payments for each student.

```
SELECT
    ->      s.student_id,
    ->      s.first_name,
    ->      s.last_name,
    ->      SUM(p.amount) AS total_payments
    -> FROM
    ->      Students s
    -> LEFT JOIN
    ->      Payments p ON s.student_id = p.student_id
    -> GROUP BY
    ->      s.student_id, s.first_name, s.last_name;
```

12. Retrieve a list of course names along with the count of students enrolled in each course. Use JOIN operations between the "Courses" table and the "Enrollments" table and GROUP BY to count enrollments.

```
SELECT
    ->      c.course_name,
    ->      COUNT(e.student_id) AS students_enrolled
    -> FROM
    ->      Courses c
    -> LEFT JOIN
    ->      Enrollments e ON c.course_id = e.course_id
    -> GROUP BY
    ->      c.course_name;
```

13. Calculate the average payment amount made by students. Use JOIN operations between the "Students" table and the "Payments" table and GROUP BY to calculate the average.

```
SELECT
    -> s.student_id,
    -> s.first_name,
    -> s.last_name,
    -> AVG(p.amount) AS average_payment
    -> FROM
    -> Students s
    -> JOIN
    -> Payments p ON s.student_id = p.student_id
    -> GROUP BY
    -> s.student_id, s.first_name, s.last_name;
```