Assignment 2

Task 1. Database Design:

1. **Create the database named "SISDB"**

Soln. 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
   b. Courses
   c. Enrollments
   d. Teacher e. Payments

Soln. schema for the Students table

```
CREATE TABLE Students (
    student_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    date_of_birth DATE,
    email VARCHAR(50),
    phone_number VARCHAR(10)
);
```

schema for the Courses table

```
CREATE TABLE Courses (
    course_id INT PRIMARY KEY,
    course_name VARCHAR(100),
    credits INT,
    teacher_id INT,
    FOREIGN KEY (teacher_id) REFERENCES Teacher(teacher_id)
);
```

schema for the Enrollments table

```
CREATE TABLE Enrollments (
    enrollment_id INT PRIMARY KEY,
    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)
);
```

schema for the Teacher table

```
CREATE TABLE Teacher (
    teacher_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(50)
);
```

schema for the Payments table

```
CREATE TABLE Payments (
    payment_id INT,
    student_id INT,
    amount DECIMAL(10, 2),
    payment_date DATE
);
```

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

Soln. Primary Key constraint to Students table

```
ALTER TABLE Students

ADD CONSTRAINT PK_Students PRIMARY KEY (student_id);
```

Primary Key constraint to Courses table

```
ALTER TABLE Courses

ADD CONSTRAINT PK_Courses PRIMARY KEY (course_id);
```

Primary Key constraint to Enrollments table

```
ALTER TABLE Enrollments

ADD CONSTRAINT PK_Enrollments PRIMARY KEY (enrollment_id);
```

Foreign Key constraint to Enrollments table referencing Students table

```
ALTER TABLE Enrollments

ADD CONSTRAINT FK_Enrollments_Students FOREIGN KEY (student_id) REFERENCES
Students(student_id);
```

Foreign Key constraint to Enrollments table referencing Courses table

```
ALTER TABLE Enrollments

ADD CONSTRAINT FK_Enrollments_Courses FOREIGN KEY (course_id) REFERENCES
Courses(course_id);
```

Primary Key constraint to Teacher table

```
ALTER TABLE Teacher

ADD CONSTRAINT PK_Teacher PRIMARY KEY (teacher_id);
```

Foreign Key constraint to Courses table referencing Teacher table

```
ALTER TABLE Courses
```

ADD CONSTRAINT FK_Courses_Teacher FOREIGN KEY (teacher_id) REFERENCES Teacher(teacher_id);

Primary Key constraint to Payments table

ALTER TABLE Payments

ADD CONSTRAINT PK_Payments PRIMARY KEY (payment_id);

Foreign Key constraint to Payments table referencing Students table

ALTER TABLE Payments

ADD CONSTRAINT FK_Payments_Students FOREIGN KEY (student_id) REFERENCES Students(student_id);

5. Insert at least 10 sample records into each of the following tables. i. Students ii. Courses Enrollments iv. Teacher v. Payments

Soln. Insert into Students table

INSERT INTO Students VALUES

(1, 'Alice', 'Johnson', '1990-03-15', 'alice.johnson@email.com', '123-456-7890'),

(3, 'Charlie', 'Williams', '1992-05-10', 'charlie.williams@email.com', '987-654-3210'),

(4, 'David', 'Smith', '1988-11-18', 'david.smith@email.com', '555-123-4567'),

(5, 'Eva', 'Brown', '1995-07-03', 'eva.brown@email.com', '789-321-6540'),

(6, 'Frank', 'Miller', '1982-09-21', 'frank.miller@email.com', '333-444-5555'),

(7, 'Grace', 'Taylor', '1993-10-12', 'grace.taylor@email.com', '777-888-9999'),

(8, 'Henry', 'Anderson', '1986-06-27', 'henry.anderson@email.com', '111-222-3333'),

(9, 'Ivy', 'Clark', '1998-03-07', 'ivy.clark@email.com', '666-777-8888'),

(10, 'Jack', 'Davis', '1980-05-05', 'jack.davis@email.com', '444-555-6666'),

(11, 'Liam', 'Garcia', '1994-08-12', 'liam.garcia@email.com', '222-333-4444');

Insert 10 sample records into the "Courses" table

```
INSERT INTO Courses (course_id, course_name, credits, teacher_id)
VALUES
(11, 'Geography', 3, 110),
(12, 'Economics', 4, 111),
(13, 'Psychology', 3, 112),
(14, 'Engineering', 4, 113),
(15, 'Political Science', 3, 114),
(16, 'Environmental Science', 4, 115),
(17, 'Sociology', 3, 116),
(18, 'Drama', 2, 117),
(19, 'Business Management', 4, 118),
(20, 'Language Arts', 3, 119);
```

Insert at least 10 sample records into the " Enrollments" table

```
INSERT INTO Enrollments VALUES
(2, 1, 1, '2023-01-20'),
(2, 3, 3, '2023-02-25'),
(3, 1, 2, '2023-03-15'),
(1, 2, 1, '2023-04-20'),
(1, 5, 4, '2023-05-27'),
(5, 6, 3, '2023-06-28'),
(9, 4, 4, '2023-07-30'),
(3, 6, 1, '2023-08-25'),
(8, 10, 9, '2023-09-15'),
(2, 10, 1, '2023-10-20');
```

Insert 10 sample records into the "Teacher" table

```
INSERT INTO Teacher (teacher_id, first_name, last_name, email)
VALUES
(110, 'Alice', 'Johnson', 'alice.johnson@email.com'),
```

(111, 'Charlie', 'Smith', 'charlie.smith@email.com'),

(112, 'David', 'Brown', 'david.brown@email.com'),

(113, 'Eva', 'Martinez', 'eva.martinez@email.com'),

(114, 'Frank', 'Williams', 'frank.williams@email.com'),

(115, 'Grace', 'Taylor', 'grace.taylor@email.com'),

(116, 'Henry', 'Jones', 'henry.jones@email.com'),

(117, 'Ivy', 'Anderson', 'ivy.anderson@email.com'),

(118, 'Jack', 'Davis', 'jack.davis@email.com'),

(119, 'Liam', 'Miller', 'liam.miller@email.com');

Insert at least 10 sample records into the "Payments" table

INSERT INTO Payments VALUES

(1, 1, 120.50, '2023-03-05'),

(2, 2, 180.25, '2023-03-10'),

(3, 3, 220.75, '2023-03-15'),

(4, 4, 135.80, '2023-03-20'),

(5, 5, 95.50, '2023-03-25'),

(6, 6, 145.20, '2023-03-30'),

(7, 7, 105.30, '2023-04-05'),

(8, 8, 195.60, '2023-04-10'),

(9, 9, 125.40, '2023-04-15'),

(10, 10, 160.75, '2023-04-20');

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:

- First Name: John
- Last Name: Doe
- Date of Birth: 1995-08-15
- Email: john.doe@example.com
- Phone Number: 1234567890

Ans. Insert a new student into the "Students" table

```
INSERT INTO Students (first_name, last_name, date_of_birth, email, phone_number)
VALUES ('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.

Ans. Enroll a student in a course

```
INSERT INTO Enrollments (student_id, course_id, enrollment_date)
VALUES (existing_student_id, existing_course_id, '2023-01-20');
```

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

Ans. Update the email address of a specific teacher

```
UPDATE Teacher
SET email = 'newemail@example.com'
WHERE teacher_id = specific_teacher_id;
```

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

Ans. Delete a specific enrollment record

```
DELETE FROM Enrollments
WHERE student_id = specific_student_id
  AND course_id = specific_course_id;
```

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

Ans. Delete a specific student and their enrollments (maintaining referential integrity)

```
DELETE FROM Enrollments
WHERE student_id = specific_student_id;

DELETE FROM Students
WHERE student_id = specific_student_id;
```

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.

Ans. Delete enrollments of a specific student from the "Enrollments" table

```
DELETE FROM Enrollments
WHERE student_id = specific_student_id;
```

Delete the specific student from the "Students" table

```
DELETE FROM Students
WHERE student_id = specific_student_id;
```

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

Ans. Update the payment amount for a specific payment record

```
UPDATE Payments
SET amount = new_amount
WHERE payment_id = specific_payment_id;
```

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.

Ans. Calculate total payments made by a specific student

```
SELECT 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.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.

Ans. Retrieve a list of courses with the count of students enrolled in each course

```
SELECT c.course_id, c.course_name, COUNT(e.student_id) AS enrolled_students
FROM Courses c
LEFT 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.

Ans. Find the names of students who have not enrolled in any course

```
SELECT 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.

Ans. Retrieve the first name, last name of students, and the names of the courses they are enrolled in

```
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.

Ans. List names of teachers and the courses they are assigned to

```
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.

Ans. Retrieve a list of students and their enrollment dates for a specific course

```
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_id = specific_course_id;
```

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.

Ans. Find names of students who have not made any payments

```
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.

Ans. Identify courses that have no enrollments

```
SELECT c.course_id, 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.

Ans. Identify students who are enrolled in more than one course

```
SELECT s.student_id, s.first_name, s.last_name, COUNT(e.enrollment_id) AS num_enrollments
FROM Students s
JOIN Enrollments e ON s.student_id = e.student_id
GROUP BY s.student_id, s.first_name, s.last_name
HAVING COUNT(e.enrollment_id) > 1;
```

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

Ans. Find teachers who are not assigned to any courses

```
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.

Ans.

```
SELECT course_id, AVG(num_students) AS average_students
FROM (
    SELECT course_id, COUNT(DISTINCT student_id) AS num_students
    FROM Enrollments
    GROUP BY course_id
) AS CourseEnrollments
GROUP BY course_id;
```

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.

Ans.

```
SELECT student_id, amount, payment_date
FROM Payments
WHERE amount = (SELECT MAX(amount) FROM Payments);
```

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

Ans.

```
SELECT c.course_id, c.course_name, COUNT(e.enrollment_id) AS enrollment_count
FROM Courses c
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 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.

Ans.

```
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
JOIN Enrollments e ON c.course_id = e.course_id
JOIN Payments p ON e.student_id = p.student_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.

Ans.

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

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

Ans.

```
SELECT teacher_id, first_name, last_name
FROM Teacher t
WHERE NOT EXISTS (
   SELECT 1
   FROM Courses c
   WHERE t.teacher_id = c.teacher_id);
```

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

Ans.

```
SELECT AVG(student_age) AS average_age
FROM (
```

```
    SELECT TIMESTAMPDIFF(YEAR, date_of_birth, CURDATE()) AS student_age
    FROM Students
) AS student_ages;
```

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

Ans.
```
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.

Ans.
```
SELECT
    s.student_id,
    s.first_name,
    s.last_name,
    c.course_id,
    c.course_name,
    COALESCE(SUM(p.amount), 0) 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
GROUP BY
    s.student_id,
    s.first_name,
    s.last_name,
    c.course_id,
    c.course_name;
```

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.

Ans.
```
SELECT
    student_id,
```

```
    first_name,
    last_name
FROM Students
WHERE student_id IN (
    SELECT student_id
    FROM Payments
    GROUP BY student_id
    HAVING COUNT(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.

Ans.

```
SELECT
    s.student_id,
    s.first_name,
    s.last_name,
    COALESCE(SUM(p.amount), 0) 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

Ans.

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

Ans.

```
SELECT
```

```
    s.student_id,
    s.first_name,
    s.last_name,
    COALESCE(AVG(p.amount), 0) AS average_payment_amount
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;
```