ER DIAGRAM

Phone\_number

makes

has

enroll

pays

enrollment

teacher

courses

payment

student

create database sisdb;

use sisdb;

create table students (

student\_id int primary key auto\_increment,

first\_name varchar(50) not null,

last\_name varchar(50) not null,

date\_of\_birth date not null,

email varchar(100) unique not null,

phone\_number varchar(15)

);

create table teacher (

teacher\_id int primary key auto\_increment ,

first\_name varchar(50) not null,

last\_name varchar(50) not null,

email varchar(100)unique not null

);

create table courses (

course\_id int primary key auto\_increment,

course\_name varchar(100)not null,

course\_code varchar(10)unique not null,

credits int not null,

teacher\_id int,

foreign key (teacher\_id) references teacher(teacher\_id)

);

create table enrollments (

enrollment\_id int primary key auto\_increment,

student\_id int,

course\_id int,

enrollment\_date date not null,

foreign key (student\_id) references students(student\_id),

foreign key (course\_id) references courses(course\_id)

);

create table payments (

payment\_id int primary key auto\_increment ,

student\_id int,

amount decimal(10,2) not null,

payment\_date date not null,

foreign key (student\_id) references students(student\_id)

);

**------------------creation of tables over------------------------**

insert into students (student\_id, first\_name, last\_name, date\_of\_birth, email, phone\_number) values

(1, 'John', 'Doe', '2001-05-15', 'john.doe@example.com', '1234567890'),

(2, 'Jane', 'Smith', '2002-08-22', 'jane.smith@example.com', '0987654321'),

(3, 'Alice', 'Brown', '2000-11-10', 'alice.brown@example.com', '1122334455'),

(4, 'Bob', 'Johnson', '1999-03-30', 'bob.johnson@example.com', '2233445566'),

(5, 'Charlie', 'White', '2001-07-25', 'charlie.white@example.com', '3344556677'),

(6, 'David', 'Lee', '2003-09-14', 'david.lee@example.com', '4455667788'),

(7, 'Emma', 'Harris', '2002-02-17', 'emma.harris@example.com', '5566778899'),

(8, 'Frank', 'Clark', '2000-06-20', 'frank.clark@example.com', '6677889900'),

(9, 'Grace', 'Walker', '2001-12-05', 'grace.walker@example.com', '7788990011'),

(10, 'Hank', 'Adams', '2002-04-28', 'hank.adams@example.com', '8899001122');

insert into teacher (teacher\_id, first\_name, last\_name, email) values

(1, 'Michael', 'Anderson', 'michael.anderson@example.com'),

(2, 'Sarah', 'Taylor', 'sarah.taylor@example.com'),

(3, 'Robert', 'Brown', 'robert.brown@example.com'),

(4, 'Laura', 'Wilson', 'laura.wilson@example.com'),

(5, 'James', 'Moore', 'james.moore@example.com'),

(6, 'Emily', 'Thomas', 'emily.thomas@example.com'),

(7, 'William', 'Garcia', 'william.garcia@example.com'),

(8, 'Olivia', 'Martinez', 'olivia.martinez@example.com'),

(9, 'David', 'Rodriguez', 'david.rodriguez@example.com'),

(10, 'Sophia', 'Hernandez', 'sophia.hernandez@example.com');

insert into courses (course\_id, course\_name, course\_code, credits, teacher\_id) values

(1, 'Mathematics', 'MTH101', 3, 1),

(2, 'Physics', 'PHY102', 4, 2),

(3, 'Chemistry', 'CHM103', 3, 3),

(4, 'Biology', 'BIO104', 3, 4),

(5, 'English Literature', 'ENG105', 2, 5),

(6, 'Computer Science', 'CSC106', 4, 6),

(7, 'History', 'HIS107', 3, 7),

(8, 'Economics', 'ECO108', 3, 8),

(9, 'Psychology', 'PSY109', 2, 9),

(10, 'Sociology', 'SOC110', 3, 10);

insert into enrollments (enrollment\_id, student\_id, course\_id, enrollment\_date) values

(1, 1, 1, '2024-01-15'),

(2, 2, 2, '2024-01-16'),

(3, 3, 3, '2024-01-17'),

(4, 4, 4, '2024-01-18'),

(5, 5, 5, '2024-01-19'),

(6, 6, 6, '2024-01-20'),

(7, 7, 7, '2024-01-21'),

(8, 8, 8, '2024-01-22'),

(9, 9, 9, '2024-01-23'),

(10, 10, 10, '2024-01-24');

insert into payments (payment\_id, student\_id, amount, payment\_date) values

(1, 1, 500.00, '2024-02-01'),

(2, 2, 600.50, '2024-02-02'),

(3, 3, 700.75, '2024-02-03'),

(4, 4, 800.25, '2024-02-04'),

(5, 5, 900.00, '2024-02-05'),

(6, 6, 1000.99, '2024-02-06'),

(7, 7, 1100.50, '2024-02-07'),

(8, 8, 1200.75, '2024-02-08'),

(9, 9, 1300.30, '2024-02-09'),

(10, 10, 1400.45, '2024-02-10');

**------------------insertion of values over---------------------**

**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 (first\_name, last\_name, date\_of\_birth, email, phone\_number)

values (1,'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 (101, 5, 3, '2025-03-18');

**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 = 'teacher@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 = 6 and course\_id = 3;

**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 = 3;

**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 enrollments

where student\_id = 5;

delete from students

where student\_id = 5;

delete from enrollments

where student\_id = 5;

**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 = 500.00 where payment\_id = 3;

**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 = 6 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 student\_count 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.first\_name, s.last\_name from students s left join enrollments e on s.student\_id = e.student\_id where e.student\_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 AS 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 AS Student\_First\_Name, s.last\_name AS Student\_Last\_Name, e.enrollment\_date AS 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 = 'Mathematics';

**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 AS Student\_First\_Name, s.last\_name AS Student\_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 AS Course\_Name, c.course\_code AS Course\_Code 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, COUNT(e.course\_id) AS Course\_Count 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.course\_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.**

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 AVG(student\_count) AS Average\_Enrollment\_Per\_Course FROM (SELECT course\_id, COUNT(student\_id) AS student\_count FROM enrollments GROUP BY course\_id) AS CourseEnrollmentCounts;

**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 s.student\_id, s.first\_name, s.last\_name, p.amount AS Highest\_Payment FROM students s JOIN payments p ON s.student\_id = p.student\_id

WHERE p.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.**

SELECT c.course\_id, c.course\_name, COUNT(e.student\_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.student\_id) = ( SELECT MAX(enrollment\_count)

FROM ( SELECT course\_id, COUNT(student\_id) AS enrollment\_count

FROM enrollments GROUP BY course\_id ) AS CourseEnrollments);

**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,

COALESCE(SUM(p.amount), 0) AS total\_payments FROM teacher t

LEFT 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.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.**

SELECT s.student\_id, s.first\_name, s.last\_name

FROM students s

WHERE ( SELECT COUNT(e.course\_id)

FROM enrollments e WHERE e.student\_id = s.student\_id) = ( SELECT COUNT(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 first\_name, last\_name

FROM teacher

WHERE teacher\_id NOT IN (

SELECT DISTINCT teacher\_id FROM courses WHERE teacher\_id IS NOT NULL

);

**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(student\_age) AS average\_age

FROM ( SELECT FLOOR(DATEDIFF(CURDATE(), date\_of\_birth) / 365) AS student\_ageFROM students)

AS age\_subquery;

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

SELECT 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\_name,

(SELECT SUM(p.amount) FROM payments p

WHERE p.student\_id = s.student\_id) AS total\_payment FROM students s

JOIN enrollments e ON s.student\_id = e.student\_id

JOIN courses c ON e.course\_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 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.**

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**.

SELECT c.course\_id, c.course\_name, COALESCE(COUNT(e.student\_id), 0) AS student\_count

FROM courses c

LEFT JOIN enrollments e ON c.course\_id = e.course\_id

GROUP BY c.course\_id, c.course\_name

ORDER BY student\_count DESC;

**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,

CONCAT(s.first\_name, ' ', s.last\_name) AS student\_name,

COALESCE(AVG(p.amount), 0) AS average\_payment

FROM students s

LEFT JOIN payments p ON s.student\_id = p.student\_id

GROUP BY s.student\_id, student\_name

ORDER BY average\_payment DESC;