**Assignment 2**

**In this assignment, you will work with a simplified Student Information System (SIS) database. The SIS database contains information about students, courses, and enrollments. Your task is to perform various SQL operations on this database to retrieve and manipulate data.**

**Database Tables: The SIS database consists of the following tables:**

**1. Students**

* **student\_id (Primary Key)**
* **first\_name**
* **last\_name**
* **date\_of\_birth**
* **email**
* **phone\_number**

**2. Courses**

* **course\_id (Primary Key)**
* **course\_name**
* **credits**
* **teacher\_id (Foreign Key)**

**3. Enrollments**

* **enrollment\_id (Primary Key)**
* **student\_id (Foreign Key)**
* **course\_id (Foreign Key)**
* **enrollment\_date**

**4. Teacher**

* **teacher\_id (Primary Key)**
* **first\_name**
* **last\_name**
* **email**

**5. Payments**

* **payment\_id (Primary Key)**
* **student\_id (Foreign Key)**
* **amount**
* **payment\_date**

**Task 1. Database Design:**

1. **Create the database named "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**
3. **Courses**
4. **Enrollments**
5. **Teacher**
6. **Payments**

CREATE DATABASE SISDB

CREATE TABLE Students( student\_id INT PRIMARY KEY, first\_name VARCHAR(30), last\_name VARCHAR(30), date\_of\_birth DATE, email VARCHAR(50), phone\_number VARCHAR(10)

);

CREATE TABLE Courses( course\_id INT PRIMARY KEY, course\_name VARCHAR(50), credits INT, teacher\_id INT,

FOREIGN KEY (teacher\_id) REFERENCES Teacher (teacher\_id)

);

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)

);

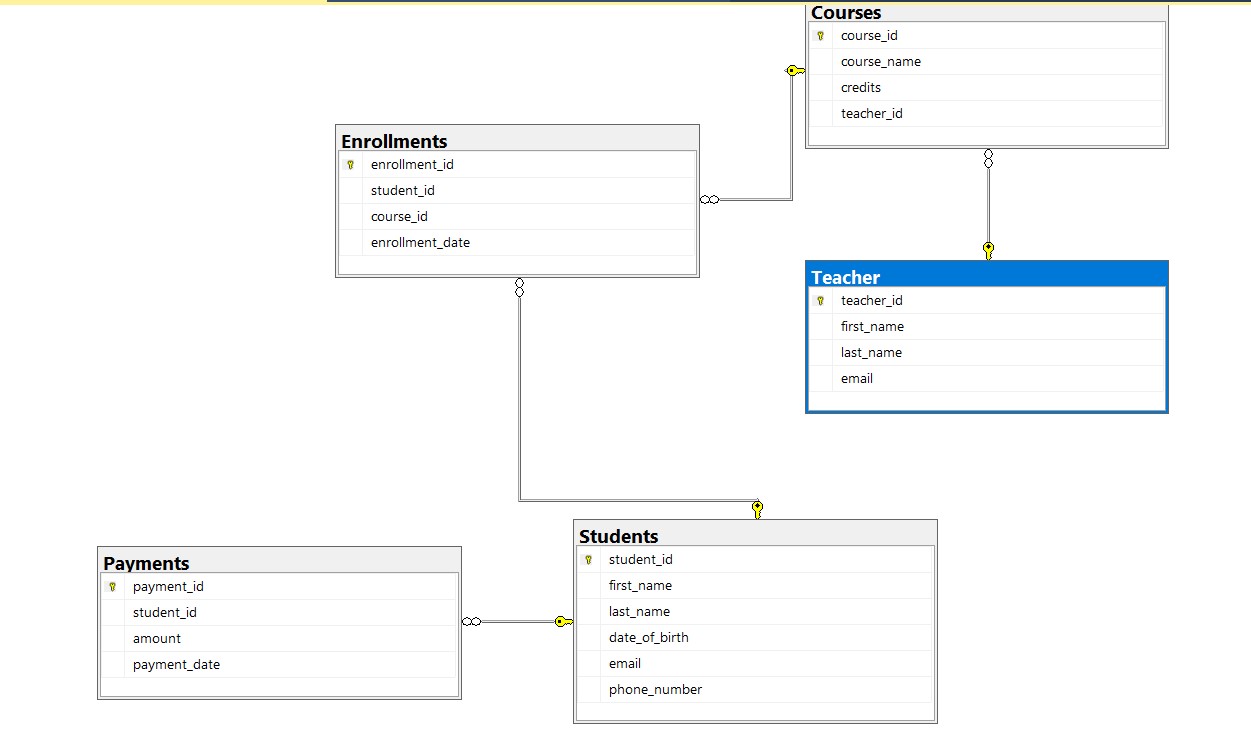
CREATE TABLE Teacher ( teacher\_id INT PRIMARY KEY, first\_name VARCHAR(50), last\_name VARCHAR(50), email VARCHAR(100)

);

CREATE TABLE Payments ( payment\_id INT PRIMARY KEY, student\_id INT, amount DECIMAL(10, 2), payment\_date DATE,

FOREIGN KEY (student\_id) REFERENCES Students(student\_id) );

1. **Create an ERD (Entity Relationship Diagram) for the database.**
2. **Create appropriate Primary Key and Foreign Key constraints for referential integrity.**



1. **Insert at least 10 sample records into each of the following tables. i. Students ii. Courses iii. Enrollments iv. Teacher**

**v. Payments**

INSERT INTO Students VALUES

(1, 'John', 'Doe', '1995-08-15', 'john.doe@example.com', '1234567890'),

(2, 'Jane', 'Smith', '1998-05-22', 'jane.smith@example.com',

'9876543210'),

(3, 'Alice', 'Johnson', '1997-11-30', 'alice.johnson@example.com', '5555555555'),

(4, 'Bob', 'Williams', '1996-04-10', 'bob.williams@example.com',

'7777777777'),

(5, 'Eva', 'Brown', '1999-09-18', 'eva.brown@example.com', '8888888888'),

(6, 'Charlie', 'Taylor', '1994-03-05', 'charlie.taylor@example.com', '6666666666'),

(7, 'Sophia', 'Martin', '1993-01-25', 'sophia.martin@example.com', '9999999999'),

(8, 'Daniel', 'Clark', '1992-07-12', 'daniel.clark@example.com',

'1111111111'),

(9, 'Olivia', 'Anderson', '1997-12-08', 'olivia.anderson@example.com', '2222222222'),

(10, 'Michael', 'Davis', '1996-06-20', 'michael.davis@example.com',

'3333333333');

INSERT INTO Teacher VALUES

(1, 'Professor1', 'Johnson', 'professor1.johnson@example.com'),

(2, 'Professor2', 'Smith', 'professor2.smith@example.com'),

(3, 'Professor3', 'Brown', 'professor3.brown@example.com'),

(4, 'Professor4', 'Clark', 'professor4.clark@example.com'),

(5, 'Professor5', 'Taylor', 'professor5.taylor@example.com'),

(6, 'Professor6', 'Anderson', 'professor6.anderson@example.com'),

(7, 'Professor7', 'Martin', 'professor7.martin@example.com'),

(8, 'Professor8', 'Williams', 'professor8.williams@example.com'),

(9, 'Professor9', 'Davis', 'professor9.davis@example.com'),

(10, 'Professor10', 'White', 'professor10.white@example.com');

INSERT INTO Courses VALUES

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

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

(3, 'Chemistry', 3, 2),

(4, 'History', 3, 3),

(5, 'Computer Science', 4, 4),

(6, 'English Literature', 3, 4),

(7, 'Biology', 4, 5),

(8, 'Art', 2, 6),

(9, 'Economics', 3, 7),

(10, 'Psychology', 3, 8),

(11, 'Algebra', 9, 9),

(12, 'Geography', 2, 10);

INSERT INTO Enrollments VALUES

(101, 1, 1, '2023-01-15'),

(102, 2, 2, '2023-01-16'),

(103, 3, 2, '2023-01-17'),

(104, 4, 2, '2023-01-18'),

(105, 5, 3, '2023-01-19'),

(106, 6, 3, '2023-01-20'),

(107, 7, 4, '2023-01-21'),

(108, 8, 4, '2023-01-22'),

(109, 9, 5, '2023-01-23'),

(110, 10, 5, '2023-01-24');

INSERT INTO Payments VALUES

(1, 1, 100.00, '2023-02-01'),

(2, 2, 120.00, '2023-02-02'), (3, 3, 90.00, '2023-02-03'),

(4, 4, 110.00, '2023-02-04'),

(5, 5, 80.00, '2023-02-05'),

(6, 6, 130.00, '2023-02-06'),

(7, 7, 95.00, '2023-02-07'),

(8, 8, 105.00, '2023-02-08'),

(9, 9, 75.00, '2023-02-09'),

(10, 10, 85.00, '2023-02-10');

**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**

1. **Last Name: Doe**
2. **Date of Birth: 1995-08-15**
3. **Email: john.doe@example.com**
4. **Phone Number: 1234567890**

INSERT INTO Students VALUES (11, 'John', 'Doe', '1995-08-15',

'john.doe@example.com', '1234567890');

1. **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.**

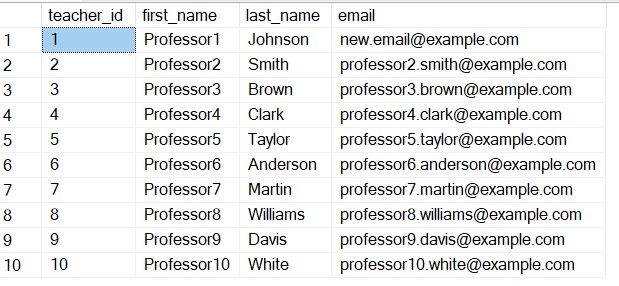
INSERT INTO Enrollments VALUES (1, 1, 2, '2023-02-01');

1. **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'

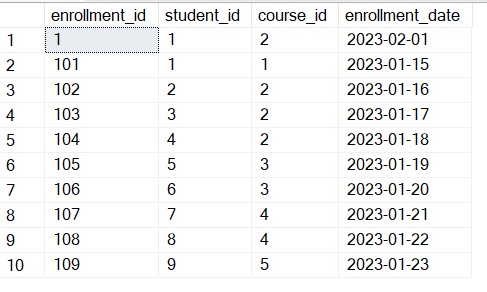
WHERE teacher\_id = 1;



1. **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

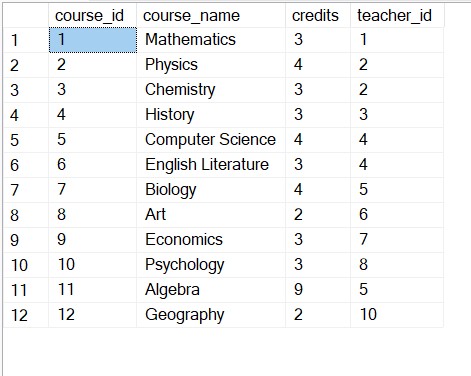
WHERE enrollment\_id=110 AND course\_id=5;



1. **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 = 5 WHERE course\_id = 11;



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

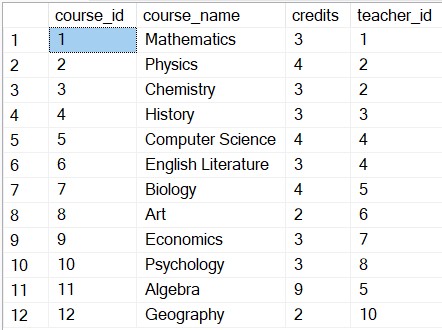
DELETE FROM Enrollments WHERE student\_id = 1;

-- Step 1: Delete related records in the Payments table

DELETE FROM Payments WHERE student\_id = 1;

-- Step 2: Delete the student from the Students table

DELETE FROM Students WHERE student\_id = 1;

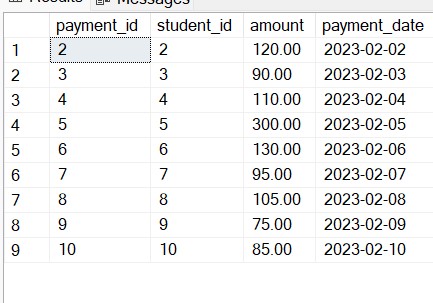


1. **Update the payment amount for a specific payment record in the "Payments" table. Choose any payment record and modify the payment amount.**

UPDATE Payments

SET amount = 300.00

WHERE payment\_id=5;



**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.**

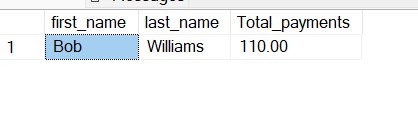
SELECT s.first\_name, s.last\_name, SUM(p.amount) AS Total\_payments FROM

Students s

JOIN Payments p ON p.student\_id = s.student\_id

WHERE p.student\_id=4

GROUP BY s.first\_name,s.last\_name;



1. **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,

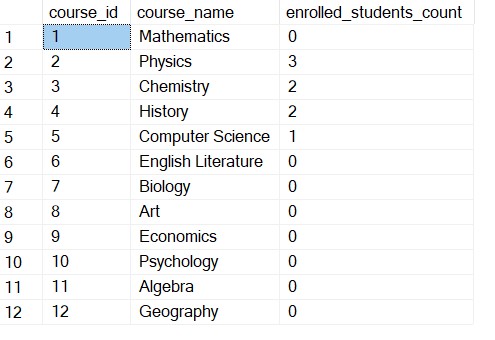
COUNT(E.student\_id) AS enrolled\_students\_count

FROM Courses C

LEFT JOIN

Enrollments E ON C.course\_id = E.course\_id

GROUP BY C.course\_id, C.course\_name;



1. **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.**

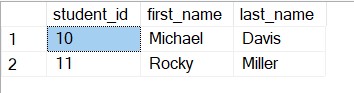
SELECT s.student\_id,s.first\_name, s.last\_name

FROM Students s

LEFT JOIN Enrollments e

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

WHERE e.student\_id IS NULL;



1. **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.**

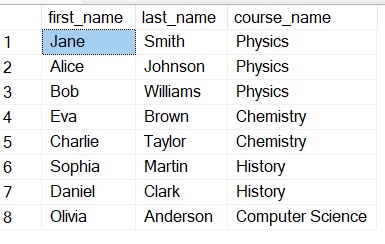
SELECT s.first\_name,s.last\_name, c.course\_name

FROM Students s

JOIN Enrollments e

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

JOIN Courses c ON c.course\_id = e.course\_id;

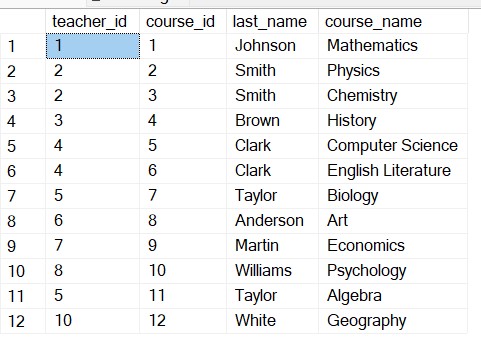


1. **Create a query to list the names of teachers and the courses they are assigned to. Join the "Teacher" table with the "Courses" table.**

SELECT t.teacher\_id,c.course\_id,t.last\_name,c.course\_name

FROM Teacher t

JOIN Courses c ON t.teacher\_id = c.teacher\_id;



1. **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.student\_id,s.first\_name,s.last\_name,e.enrollment\_date,c.course\_name

FROM Students s

JOIN

Enrollments e ON s.student\_id = e.student\_id

JOIN

Courses c ON c.course\_id = e.course\_id;



1. **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.**

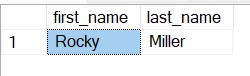
SELECT s.first\_name, s.last\_name

FROM Students s

LEFT JOIN

Payments p ON s.student\_id = p.student\_id

WHERE p.amount 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.**

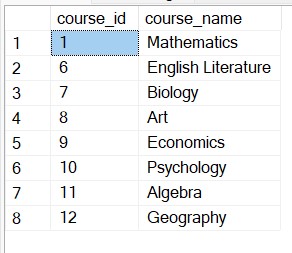
SELECT c.course\_id,c.course\_name

FROM Courses c

LEFT JOIN

Enrollments e ON c.course\_id = e.course\_id

WHERE e.course\_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

E.student\_id,

S.first\_name,

S.last\_name

FROM

Enrollments E

JOIN

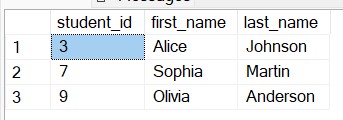
Students S ON E.student\_id = S.student\_id

GROUP BY

E.student\_id, S.first\_name, S.last\_name

HAVING

COUNT(DISTINCT 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.**

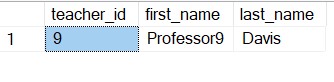
SELECT t.teacher\_id, t.first\_name,t.last\_name

FROM Teacher t

LEFT JOIN

Courses c ON t.teacher\_id = c.teacher\_id

WHERE c.teacher\_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 course\_name, AVG(student\_count) AS avg\_students\_enrolled FROM (

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

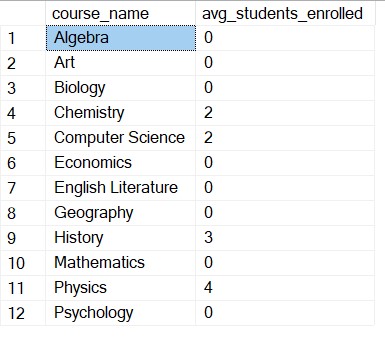
FROM Courses c

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

GROUP BY c.course\_name

) AS CourseStudentCount

GROUP BY course\_name;



1. **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

FROM Students s

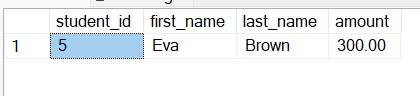
JOIN

Payments p ON s.student\_id = p.student\_id

WHERE p.amount = (

SELECT MAX(p.amount) FROM Payments p

)



1. **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,

C.credits,

C.teacher\_id,

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, C.credits, C.teacher\_id

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 CourseEnrollmentCounts

);



**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

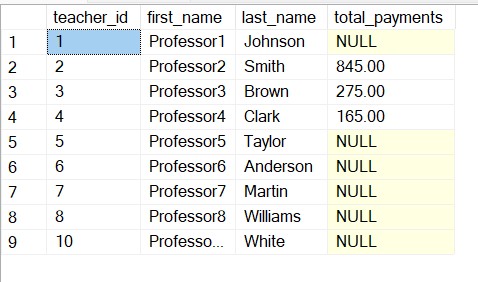
LEFT JOIN

Enrollments E ON C.course\_id = E.course\_id

LEFT JOIN

Payments P ON E.student\_id = P.student\_id

GROUP BY T.teacher\_id, T.first\_name, T.last\_name;



1. **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.first\_name, s.last\_name

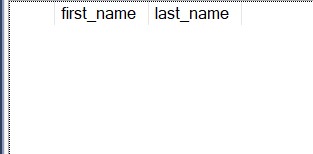
FROM Students s

WHERE (SELECT COUNT(DISTINCT e.course\_id)

FROM Enrollments e

WHERE e.student\_id = s.student\_id) = (

SELECT COUNT(DISTINCT course\_id) FROM Courses);



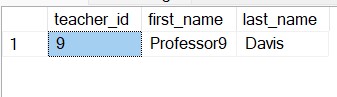
1. **Retrieve the names of teachers who have not been assigned to any courses. Use subqueries to find teachers with no course assignments.**

SELECT t.teacher\_id,t.first\_name,t.last\_name

FROM Teacher t

WHERE t.teacher\_id NOT IN

(SELECT DISTINCT c.teacher\_id FROM Courses c)



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

SELECT

AVG(age) AS average\_age

FROM (

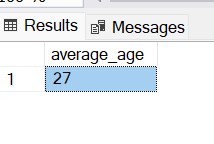
SELECT

DATEDIFF(YEAR, date\_of\_birth, GETDATE()) AS age

FROM

Students

) AS StudentAges;



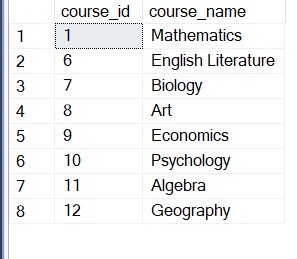
1. **Identify courses with no enrollments. Use subqueries to find courses without enrollment records.**

SELECT c.course\_id, c.course\_name

FROM Courses c

WHERE c.course\_id NOT IN (

SELECT DISTINCT e.course\_id FROM Enrollments e)



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

SELECT

E.student\_id,

C.course\_id,

SUM(P.amount) AS total\_payments

FROM

Enrollments E

JOIN

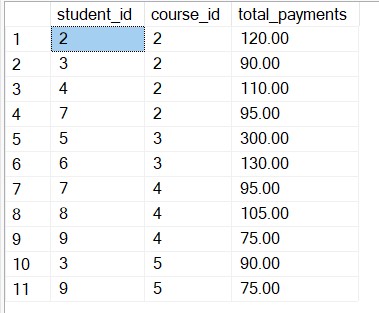
Courses C ON E.course\_id = C.course\_id

LEFT JOIN

Payments P ON E.student\_id = P.student\_id

GROUP BY

E.student\_id, C.course\_id;



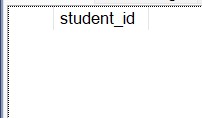
1. **Identify students who have made more than one payment. Use subqueries and aggregate functions to count payments per student and filter for those with counts greater than one.**

SELECT student\_id FROM

Payments GROUP BY student\_id

HAVING

COUNT(payment\_id) > 1;



1. **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

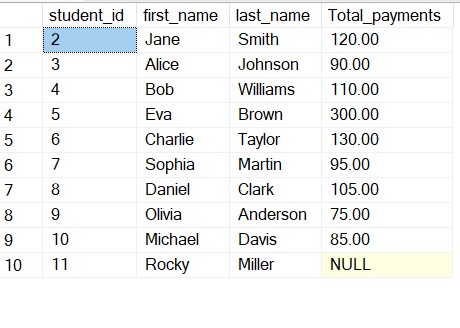
Total\_payments

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

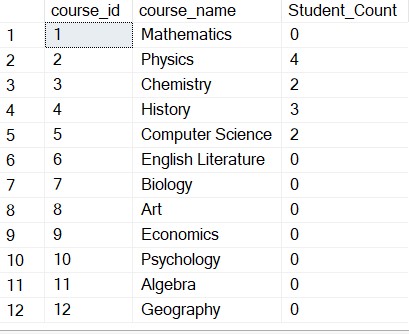


1. **Retrieve a list of course names along with the count of students enrolled in each course. Use JOIN operations between the "Courses" table and the "Enrollments" table and GROUP BY to count enrollments.**

SELECT c.course\_id,c.course\_name, COUNT(e.student\_id) 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;



1. **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

Averga\_payment

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

