**Student Information System (SIS)**

-- Task 1. Database Design:

-- 1. Create the database named "SISDB"

CREATE DATABASE SISDB;

-- Use the database named SISDB

USE 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\*/

-- table creation

CREATE TABLE Teacher (

teacher\_id INT PRIMARY KEY AUTO\_INCREMENT,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

email VARCHAR(100) UNIQUE

);

-- desc Teacher;

CREATE TABLE Students (

student\_id INT PRIMARY KEY AUTO\_INCREMENT,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

date\_of\_birth DATE,

email VARCHAR(100) UNIQUE,

phone\_number VARCHAR(20)

);

-- desc students;

CREATE TABLE Courses (

course\_id INT PRIMARY KEY AUTO\_INCREMENT,

course\_name VARCHAR(100),

credits INT,

teacher\_id INT,

FOREIGN KEY (teacher\_id) REFERENCES Teacher(teacher\_id) ON DELETE CASCADE

);

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) ON DELETE CASCADE,

FOREIGN KEY (course\_id) REFERENCES Courses(course\_id)

);

CREATE TABLE Payments (

payment\_id INT PRIMARY KEY,

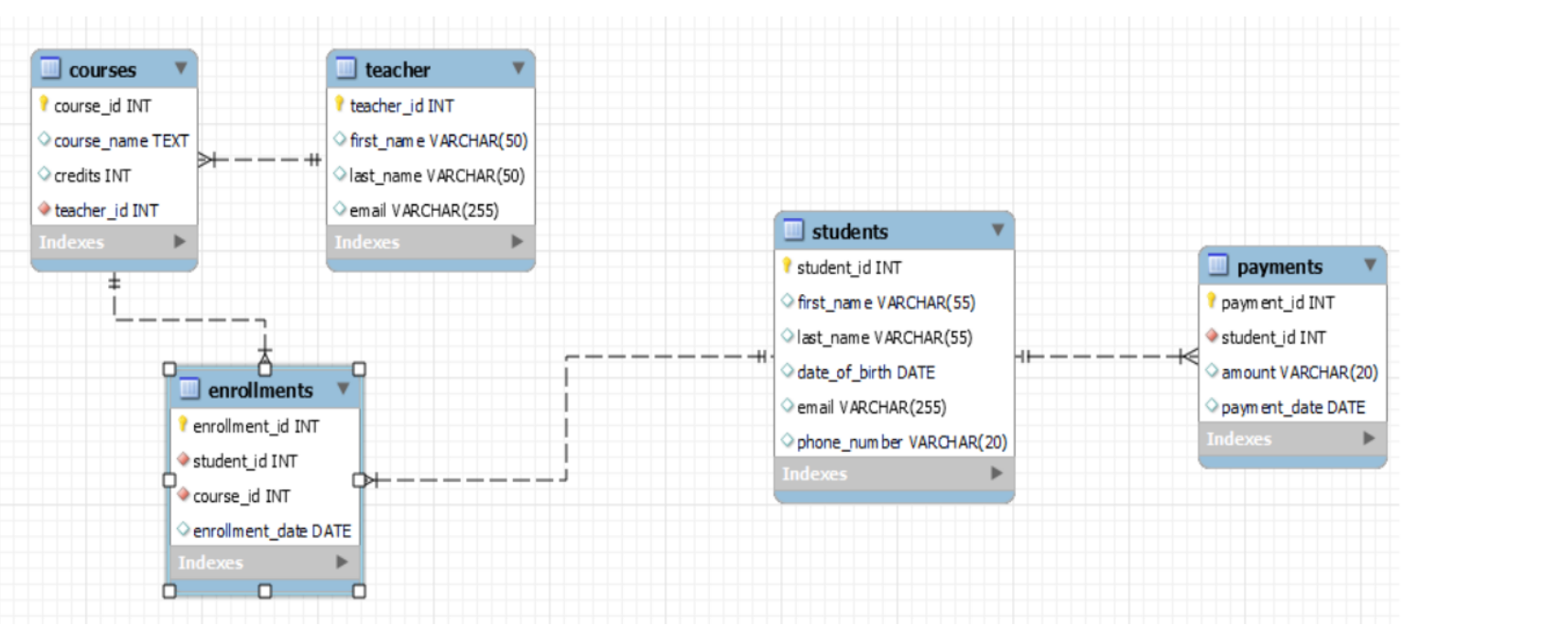
student\_id INT,

amount DECIMAL(10, 2),

payment\_date DATE,

FOREIGN KEY (student\_id) REFERENCES Students(student\_id) ON DELETE CASCADE

);



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

A screenshot of a computer

Description automatically generated

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

i. Students

ii. Courses

iii. Enrollments

iv. Teacher

v. Payments \*/

-- values insertion

INSERT INTO Students (student\_id, first\_name, last\_name, date\_of\_birth, email, phone\_number)

VALUES

(301, 'Anurag', 'Kumar', '1995-03-20', 'anurag.kumar@example.com', '9876543210'),

(302, 'ankit', 'roy', '1998-07-12', 'ankit.roy@example.com', '8765432109'),

(303, 'Anil', 'Sharma', '1997-05-15', 'anil.sharma@example.com', '7654321098'),

(304, 'gopi', 'reddy', '1996-12-02', 'gopi.reddy@example.com', '6543210987'),

(305, 'Rohan', 'Verma', '1999-08-25', 'rohan.verma@example.com', '5432109876'),

(306, 'Neha', 'Gupta', '1994-10-08', 'neha.gupta@example.com', '4321098765'),

(307, 'Vivan', 'Rajput', '1993-06-30', 'vivan.rajput@example.com', '3210987654'),

(308, 'Avani', 'Reddy', '1992-04-12', 'avani.reddy@example.com', '2109876543'),

(309, 'Aryan', 'Malhotra', '1991-02-18', 'aryan.malhotra@example.com', '1098765432'),

(310, 'Sanya', 'Thakur', '1990-11-05', 'sanya.thakur@example.com', '9876543210'),

(311, 'Aisha', 'Malik', '1994-09-18', 'aisha.malik@example.com', '9876543210'),

(312, 'Rahul', 'Sharma', '1996-05-12', 'rahul.sharma@example.com', '8765432109'),

(313, 'Sara', 'Khan', '1998-03-25', 'sara.khan@example.com', '7654321098'),

(314, 'Aditya', 'Verma', '1997-11-02', 'aditya.verma@example.com', '6543210987'),

(315, 'Neha', 'Singh', '1995-08-25', 'neha.singh@example.com', '5432109876');

INSERT INTO Teacher (teacher\_id, first\_name, last\_name, email)

VALUES

(101, 'Professor', 'Brown', 'prof.brown@example.com'),

(102, 'Dr.', 'Taylor', 'dr.taylor@example.com'),

(103, 'Ms.', 'Clark', 'ms.clark@example.com'),

(104, 'Mr.', 'Lee', 'mr.lee@example.com'),

(105, 'Mrs.', 'Adams', 'mrs.adams@example.com'),

(106, 'Dr.', 'Garcia', 'dr.garcia@example.com'),

(107, 'Mrs.', 'Martin', 'mrs.martin@example.com'),

(108, 'Mr.', 'Roberts', 'mr.roberts@example.com'),

(109, 'Professor', 'Wang', 'prof.wang@example.com'),

(110, 'Dr.', 'Lopez', 'dr.lopez@example.com'),

(111, 'Prof.', 'Smith', 'prof.smith@example.com'),

(312, 'Dr.', 'Anderson', 'dr.anderson@example.com'),

(313, 'Ms.', 'Johnson', 'ms.johnson@example.com');

INSERT INTO Courses (course\_id, course\_name, credits, teacher\_id)

VALUES

(401, 'Math', 4, 101),

(402, 'Economics', 3, 102),

(403, 'Mathematics', 3, 103),

(404, 'philosphy', 4, 104),

(405, 'History', 3, 105),

(406, 'Computer Science', 4, 106),

(407, 'Biology', 4, 107),

(408, 'Chemistry', 3, 108),

(409, 'Geography', 3, 109),

(410, 'civics', 4, 110),

(411, 'Psychology', 3, 101),

(412, 'Sociology', 3, 102),

(413, 'Philosophy', 4, 103),

(414, 'Statistics', 3, 104),

(415, 'Political Science', 4, 105),

(421, 'Psychology', 3, 101),

(422, 'Sociology', 3, 102),

(423, 'Philosophy', 4, 103),

(424, 'Statistics', 3, 104),

(425, 'Political Science', 4, 105);

INSERT INTO Enrollments (enrollment\_id, student\_id, course\_id, enrollment\_date)

VALUES

(501, 301, 401, '2023-01-05'),

(502, 301, 402, '2023-02-10'),

(503, 302, 410, '2023-01-15'),

(504, 303, 403, '2023-03-01'),

(505, 304, 404, '2023-02-20'),

(506, 305, 405, '2023-01-10'),

(507, 306, 406, '2023-01-05'),

(508, 307, 407, '2023-02-15'),

(509, 308, 408, '2023-02-25'),

(510, 309, 409, '2023-03-05'),

(521, 311, 411, '2023-03-15'),

(522, 311, 412, '2023-03-20'),

(523, 312, 413, '2023-04-01'),

(524, 312, 414, '2023-04-10'),

(525, 313, 415, '2023-04-15'),

(526, 313, 401, '2023-05-01'),

(527, 314, 402, '2023-05-10'),

(528, 314, 403, '2023-05-20'),

(529, 315, 404, '2023-06-01'),

(530, 315, 405, '2023-06-10');

INSERT INTO Enrollments (enrollment\_id, student\_id, course\_id, enrollment\_date)

VALUES

(536, 303, 401, '2023-06-20'),

(537, 303, 402, '2023-07-05'),

(538, 303, 403, '2023-07-15'),

(539, 303, 404, '2023-08-01'),

(540, 303, 405, '2023-08-10'),

(541, 303, 406, '2023-08-20'),

(542, 303, 407, '2023-09-05'),

(543, 303, 408, '2023-09-15'),

(544, 303, 409, '2023-10-01'),

(545, 303, 410, '2023-10-10'),

(546, 303, 411, '2023-10-20'),

(547, 303, 412, '2023-11-05'),

(548, 303, 413, '2023-11-10'),

(549, 303, 414, '2023-11-20'),

(550, 303, 415, '2023-12-05'),

(551, 303, 421, '2023-12-10'),

(552, 303, 422, '2023-12-20'),

(553, 303, 423, '2024-01-05'),

(554, 303, 424, '2024-01-10'),

(555, 303, 425, '2024-01-20'),

(556,303,440,'2024-01-10');

INSERT INTO Payments (payment\_id, student\_id, amount, payment\_date)

VALUES

(601, 301, 180.00, '2023-01-10'),

(602, 302, 160.50, '2023-02-15'),

(603, 303, 190.00, '2023-03-01'),

(604, 304, 175.00, '2023-02-20'),

(605, 305, 100.00, '2023-01-15'),

(606, 306, 120.00, '2023-03-05'),

(607, 307, 95.50, '2023-02-10'),

(608, 308, 85.00, '2023-01-20'),

(609, 309, 110.00, '2023-02-25'),

(610, 310, 70.00, '2023-01-05');

-- Cheching All Tables

-- select \* from courses;

-- select \* from enrollments;

-- select \* from payments;

-- select \* from students;

-- select \* from teacher;

**-- 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 ('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 (512, 301, 401, '2023-12-01');

INSERT INTO Enrollments (enrollment\_id, student\_id, course\_id, enrollment\_date)

VALUES (531, 301, 401, '2023-12-06');

INSERT INTO Enrollments (enrollment\_id, student\_id, course\_id, enrollment\_date)

VALUES (517, 301, 401, '2023-12-08');

INSERT INTO Enrollments (enrollment\_id, student\_id, course\_id, enrollment\_date)

VALUES (518, 306, 401, '2023-12-05');

-- FOR CHECKING INSERTED VALUE USE

-- select \* from enrollments;

/\* 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 = 'mew@gmailcom' WHERE teacher\_id = 107;

-- FOR CHECKING UPDATED VALUE USE

-- select \* from teacher;

/\* 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 = 301 AND course\_id = 401;

/\* 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 = 101

WHERE course\_id = 402;

-- 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 = 302;

-- FOR CHECKING UPDATED VALUE USE

-- select \* from Students;

-- 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 = 120.00

WHERE payment\_id = 602;

-- FOR CHECKING UPDATED VALUE USE

-- select \* from payments;

**-- 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, CONCAT(s.first\_name,' ',s.last\_name) AS FULL\_NAME, SUM(p.amount) AS total\_payments

FROM Students s

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

WHERE s.student\_id = 305

GROUP BY s.student\_id;

/\* 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 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.\*/

SELECT CONCAT(s.first\_name,' ', s.last\_name) AS Full\_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 CONCAT(t.first\_name,' ', t.last\_name) AS NAME\_OF\_TEACHER, 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 CONCAT(s.first\_name,' ',s.last\_name) FULL\_NAME , e.enrollment\_date,e.course\_id

FROM Students s

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

WHERE e.course\_id = 403;

/\* .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 CONCAT(s.first\_name,' ', s.last\_name) AS FULL\_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\_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.\*/

SELECT e1.student\_id, CONCAT(s.first\_name,' ',s.last\_name) AS FULL\_NAME, COUNT(DISTINCT e1.course\_id) AS courses\_enrolled

FROM Enrollments e1

JOIN Enrollments e2 ON e1.student\_id = e2.student\_id AND e1.enrollment\_id <> e2.enrollment\_id

JOIN Students s on s.student\_id=e1.student\_id

GROUP BY e1.student\_id

HAVING COUNT(DISTINCT e1.course\_id) > 1;

/\* 10. 0. 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 CONCAT(t.first\_name,' ', t.last\_name) AS NAME\_OF\_TEACHER

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\_enrolled

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 CONCAT(s.first\_name,s.last\_name) AS Full\_Name, a.amt FROM students s JOIN (

SELECT student\_id,sum(amount) amt FROM Payments

GROUP BY student\_id

ORDER BY amt DESC LIMIT 1) a ON s.student\_id=a.student\_id ;

-- CHECKING THE ENTRY

-- select \* from payments;

-- select \* from courses;

/\* .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 en\_count

FROM Courses c LEFT JOIN Enrollments e ON c.course\_id = e.course\_id GROUP BY c.course\_id, c.course\_name

HAVING en\_count = (SELECT max(en\_count) FROM (SELECT count(enrollment\_id) AS en\_count FROM Enrollments

GROUP BY course\_id) AS maximum\_enrolls);

/\* 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,c.course\_id,c.course\_name,CONCAT(t.first\_name ,' ',t.last\_name )AS NAME\_OF\_TEACHER ,SUM(p.amount) AS total\_payments\_IN$

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 c.course\_id,t.teacher\_id;

/\* .5. Identify students who are enrolled in all available courses. Use subqueries to compare a

student's enrollments with the total number of courses\*/

-- adding data so that 303 is enrolled in all the courses

INSERT INTO Enrollments (enrollment\_id, student\_id, course\_id, enrollment\_date)

VALUES

(536, 303, 401, '2023-06-20'),

(537, 303, 402, '2023-07-05'),

(538, 303, 403, '2023-07-15'),

(539, 303, 404, '2023-08-01'),

(540, 303, 405, '2023-08-10'),

(541, 303, 406, '2023-08-20'),

(542, 303, 407, '2023-09-05'),

(543, 303, 408, '2023-09-15'),

(544, 303, 409, '2023-10-01'),

(545, 303, 410, '2023-10-10'),

(546, 303, 411, '2023-10-20'),

(547, 303, 412, '2023-11-05'),

(548, 303, 413, '2023-11-10'),

(549, 303, 414, '2023-11-20'),

(550, 303, 415, '2023-12-05'),

(551, 303, 421, '2023-12-10'),

(552, 303, 422, '2023-12-20'),

(553, 303, 423, '2024-01-05'),

(554, 303, 424, '2024-01-10'),

(555, 303, 425, '2024-01-20'),

(556,303,440,'2024-01-10'),

(557,303,449,'2024-01-05');

SELECT e.student\_id ,CONCAT(s.first\_name,' ',s.last\_name) AS Full\_Name FROM Enrollments e

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

GROUP BY e.student\_id

HAVING count( DISTINCT course\_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, CONCAT(first\_name,' ' ,last\_name) AS NAME\_OF\_TEACHER 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(student\_age) AS average\_age

FROM (SELECT DATEDIFF(CURDATE(), date\_of\_birth) / 365 AS student\_age FROM Students) AS avg\_ages;

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

records\*/

-- adding a course which has no enrollments

INSERT INTO Courses (course\_id, course\_name, credits, teacher\_id)

VALUES

(459, 'ML', 9, 102);

-- (450, 'sql',30,102);

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 e.student\_id,concat(s.first\_name,' ',s.last\_name) AS FULL\_NAME ,e.course\_id,(SELECT sum(amount) FROM payments p

WHERE p.student\_id = e.student\_id) AS total\_payments

FROM enrollments e

JOIN Students s on s.student\_id=e.student\_id

GROUP BY e.student\_id,e.course\_id;

-- select \* from payments

/\* 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, CONCAT(s.first\_name, '',s.last\_name) AS Full\_Name FROM students s

JOIN ( SELECT student\_id, COUNT(payment\_id) AS payment\_count FROM payments GROUP BY student\_id)

AS payment\_counts ON s.student\_id = payment\_counts.student\_id

WHERE payment\_counts.payment\_count > 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, CONCAT(s.first\_name,' ', s.last\_name) AS FULL\_NAME, SUM(p.amount) AS total\_payments FROM Students s

JOIN Payments p ON s.student\_id = p.student\_id GROUP BY s.student\_id;

/\* 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\*/

-- IN QUESTION IT IS MENTIONED TO USE JOIN

SELECT c.course\_name, count(e.enrollment\_id) AS students\_enrolled FROM Courses c

LEFT JOIN Enrollments e ON c.course\_id = e.course\_id GROUP BY c.course\_id;

/\* 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 FULL\_NAME, AVG(p.amount) AS average\_payment\_amount from Students s

JOIN Payments p ON s.student\_id = p.student\_id GROUP BY s.student\_id;