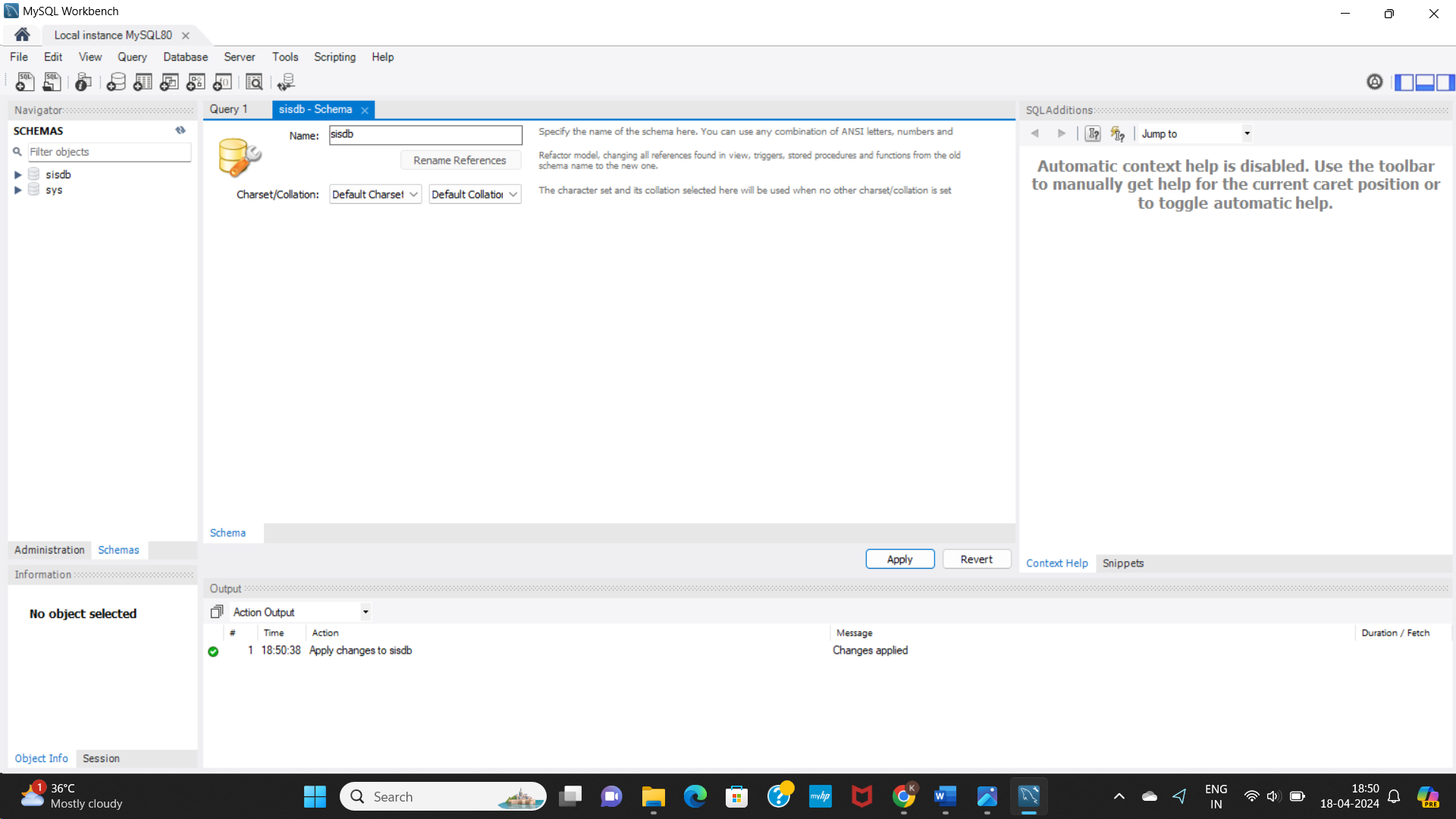
**TASK 1: DATABASE DESIGN**

1. Create the database named "SISDB"



2. Define the schema for the Students, Courses, Enrolments, 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. Enrolments

d. Teacher

e. Payments

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

CREATE TABLE Students (

­­­ student\_id int NOT NULL,

first\_name varchar (50),

last\_name varchar (50),

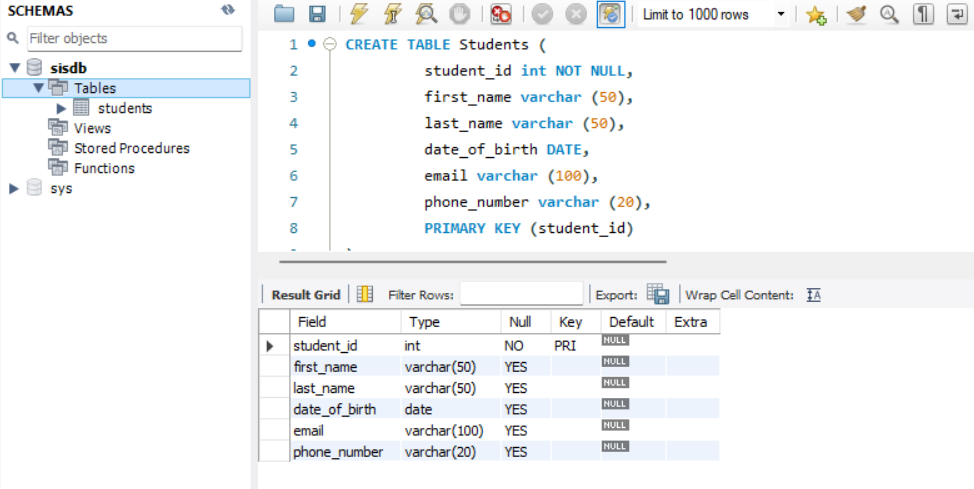
date\_of\_birth DATE,

email varchar (100),

phone\_number varchar (20),

PRIMARY KEY (student\_id)

);



CREATE TABLE Teacher (

­­­ teacher\_id int NOT NULL,

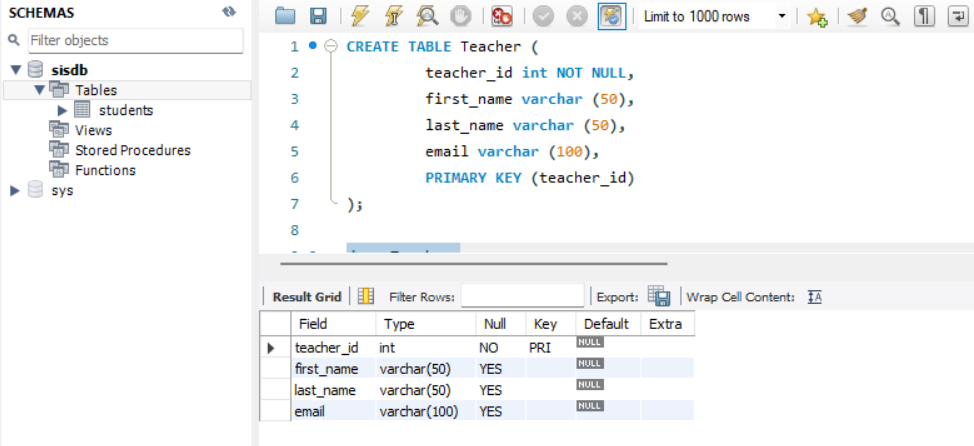
first\_name varchar (50),

last\_name varchar (50),

email varchar (100),

PRIMARY KEY (teacher\_id)

);



CREATE TABLE Courses (

­­­ course\_id int NOT NULL,

course\_name varchar (30),

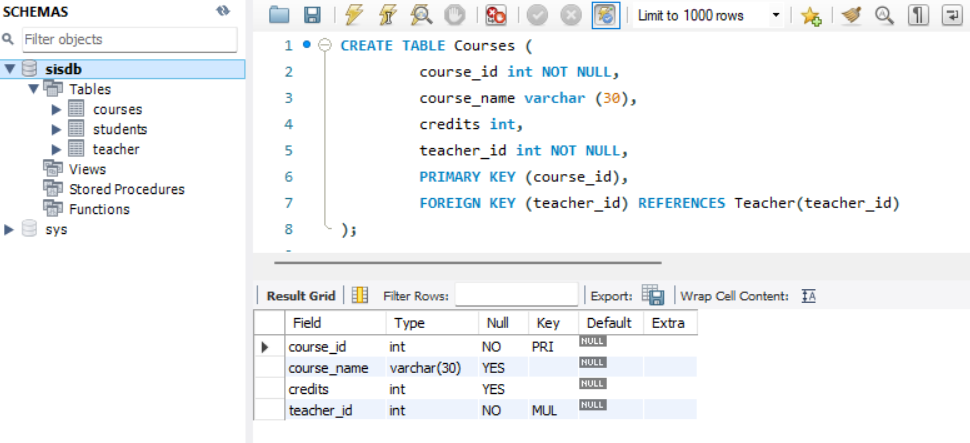
credits int,

teacher\_id int NOT NULL,

PRIMARY KEY (course\_id),

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

);



CREATE TABLE Enrolments (

­­­ enrolment\_id int NOT NULL,

student\_id int NOT NULL,

course\_id int NOT NULL,

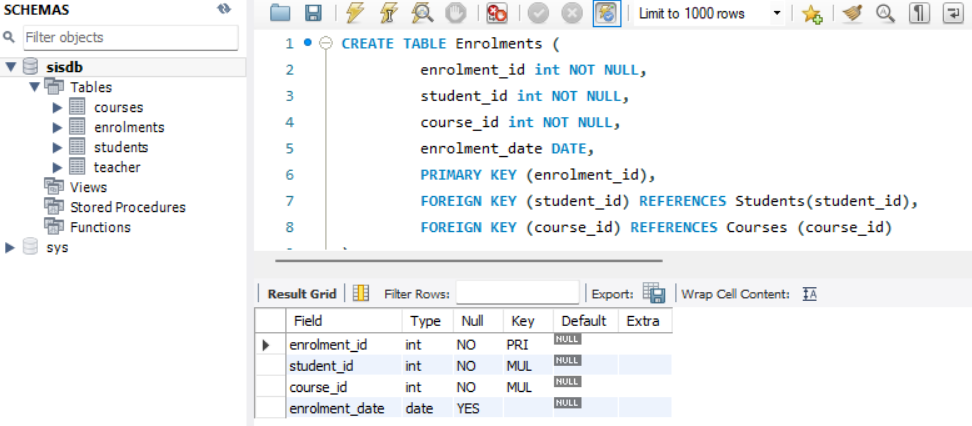
enrolment\_date DATE,

PRIMARY KEY (enrolment\_id),

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

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

);



CREATE TABLE Payments (

­­­ payment\_id int NOT NULL,

student\_id int,

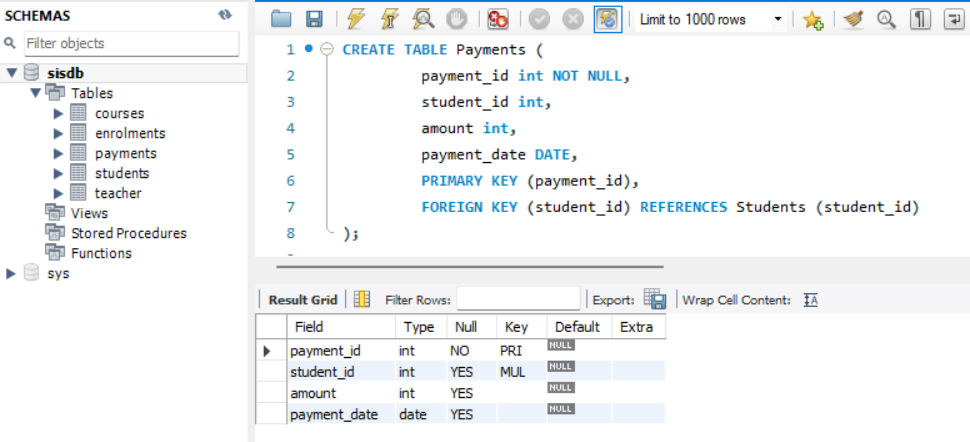
amount int,

payment\_date DATE,

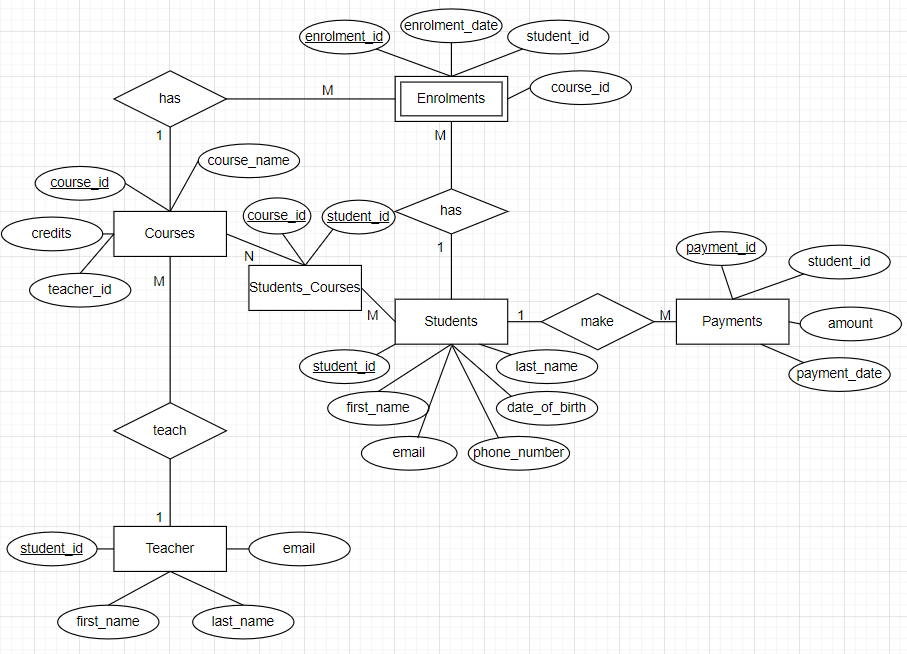
PRIMARY KEY (payment\_id),

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

);



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



Here, Enrolments is a weak entity which depends on Students and Courses.

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

i. Students

ii. Courses

iii. Enrolments

iv. Teacher

v. Payments

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

VALUES (1, 'Aditya', 'Roy', '2002-12-12', 'aditya.roy@example.com', 9856435255),

(2, 'Alice', 'Smith', '1999-11-05', 'alice.smith@example.com', 9761850325),

(3, 'Michael', 'Jones', '1999-09-25', 'michael.jones@example.com', 7645239015),

(4, 'Olivia', 'James', '2001-07-05', 'olivia.james@example.com', 8926451054),

(5, 'Emily', 'Brown', '2000-02-27', 'emily.brown@example.com', 7749803471),

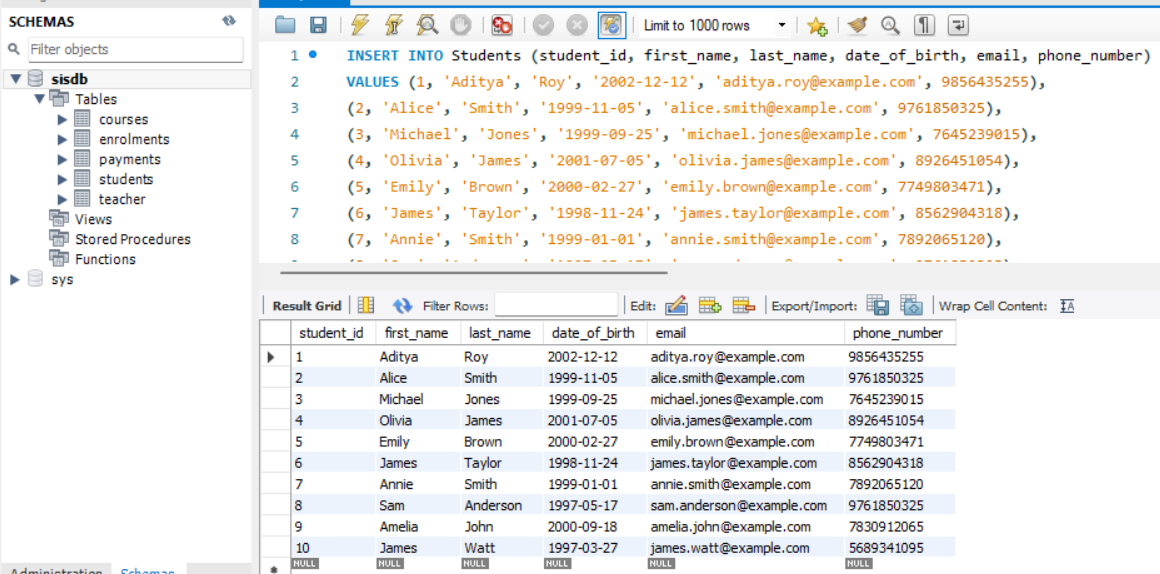
(6, 'James', 'Taylor', '1998-11-24', 'james.taylor@example.com', 8562904318),

(7, 'Annie', 'Smith', '1999-01-01', 'annie.smith@example.com', 7892065120),

(8, 'Sam', 'Anderson', '1997-05-17', 'sam.anderson@example.com', 9761850325),

(9, 'Amelia', 'John', '2000-09-18', 'amelia.john@example.com', 7830912065),

(10, 'James', 'Watt', '1997-03-27', 'james.watt@example.com', 5689341095);



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

VALUES (11, 'Sarah', 'John', 'sarah.john@example.com'),

(12, 'Peter', 'Paul', 'peter.paul@example.com'),

(13, 'David', 'Moore', 'david.moore@example.com'),

(14, 'Edward', 'Roy', 'edward.roy@example.com'),

(15, 'Mark', 'Davis', 'mark.davis@example.com'),

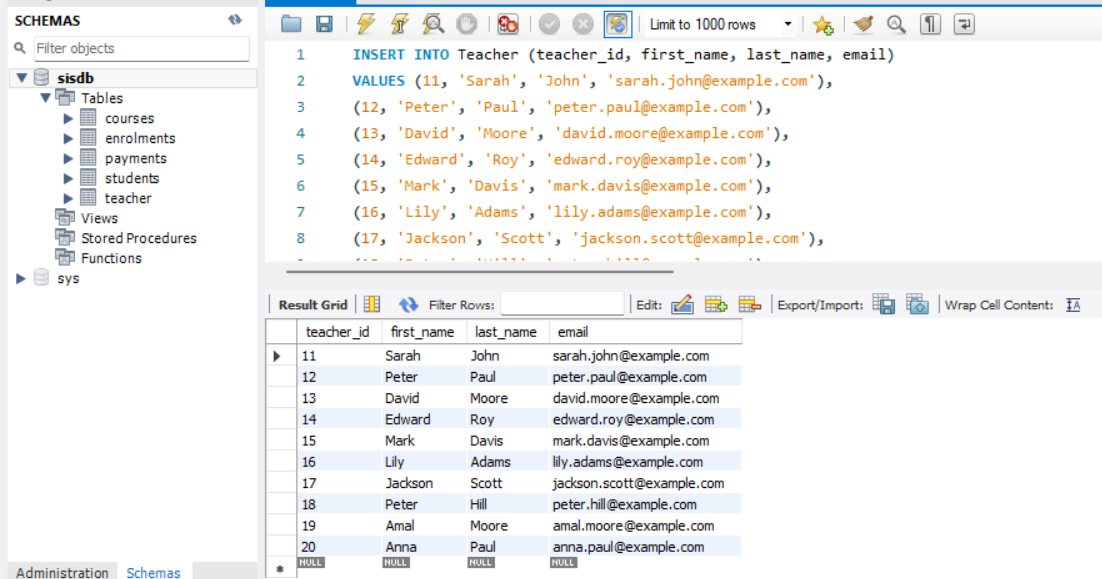
(16, 'Lily', 'Adams', 'lily.adams@example.com'),

(17, 'Jackson', 'Scott', 'jackson.scott@example.com'),

(18, 'Peter', 'Hill', 'peter.hill@example.com'),

(19, 'Amal', 'Moore', 'amal.moore@example.com'),

(20, 'Anna', 'Paul', 'anna.paul@example.com');



INSERT INTO Courses

VALUES (101, 'Physics', 4, 11),

(102, 'Economics', 3, 12),

(103, 'Psychology', 2, 13),

(104, 'Chemistry', 3, 14),

(105, 'Literature', 3, 15),

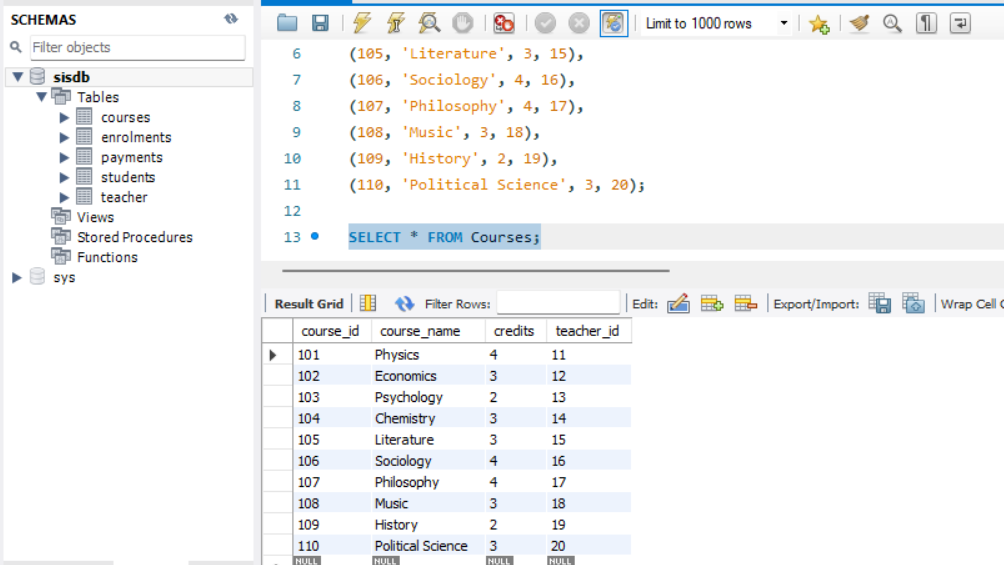
(106, 'Sociology', 4, 16),

(107, 'Philosophy', 4, 17),

(108, 'Music', 3, 18),

(109, 'History', 2, 19),

(110, 'Political Science', 3, 20);



INSERT INTO Enrolments

VALUES (1001, 1, 101, '2005-12-30'),

(1002, 2, 102, '2005-06-15'),

(1003, 3, 103, '2004-05-05'),

(1004, 4, 105, '2006-01-17'),

(1005, 5, 107, '2005-09-21'),

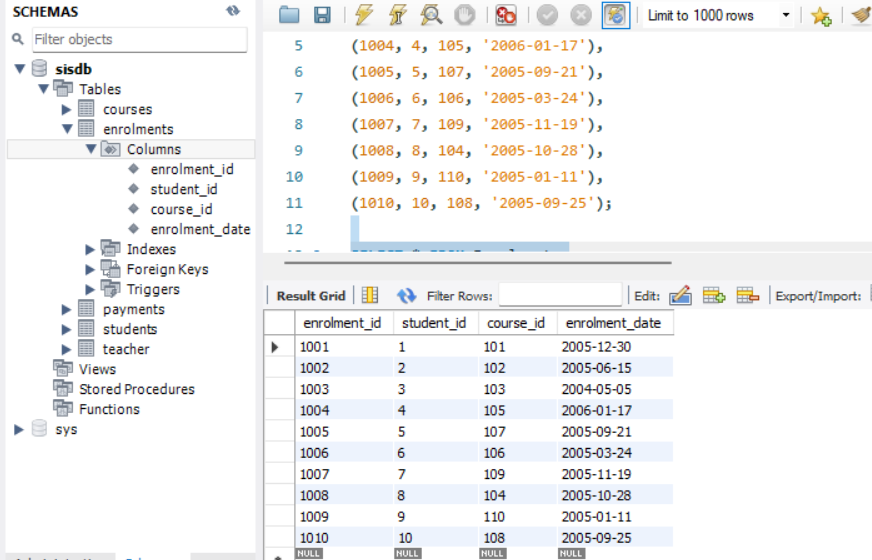
(1006, 6, 106, '2005-03-24'),

(1007, 7, 109, '2005-11-19'),

(1008, 8, 104, '2005-10-28'),

(1009, 9, 110, '2005-01-11'),

(1010, 10, 108, '2005-09-25');



INSERT INTO Payments

VALUES (201, 1, 50000, '2006-05-01'),

(202, 5, 70000, '2005-11-01'),

(203, 6, 45000, '2005-08-24'),

(204, 2, 75000, '2005-08-15'),

(205, 8, 55000, '2005-11-01'),

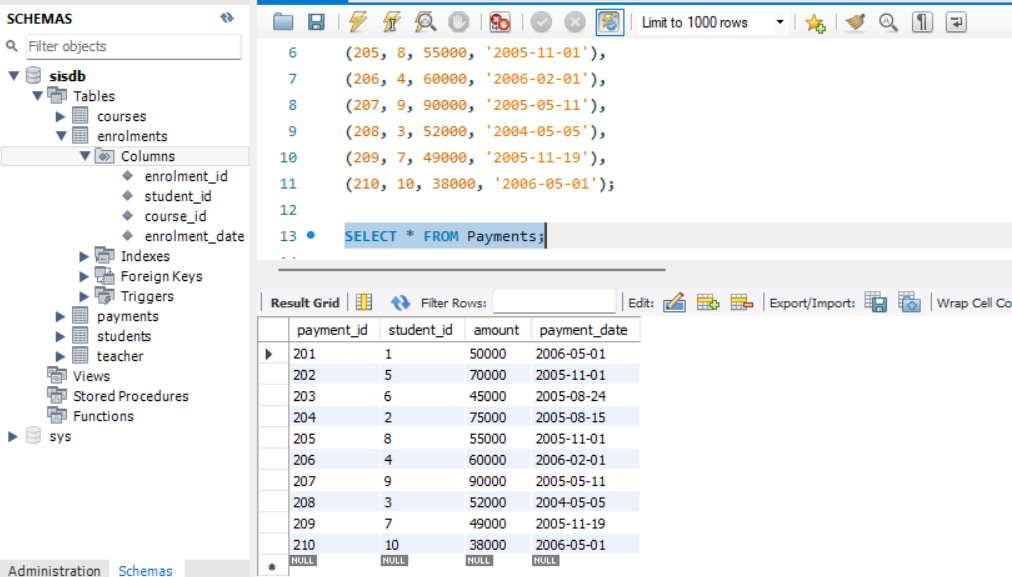
(206, 4, 60000, '2006-02-01'),

(207, 9, 90000, '2005-05-11'),

(208, 3, 52000, '2004-05-05'),

(209, 7, 49000, '2005-11-19'),

(210, 10, 38000, '2006-05-01');



**TASK 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

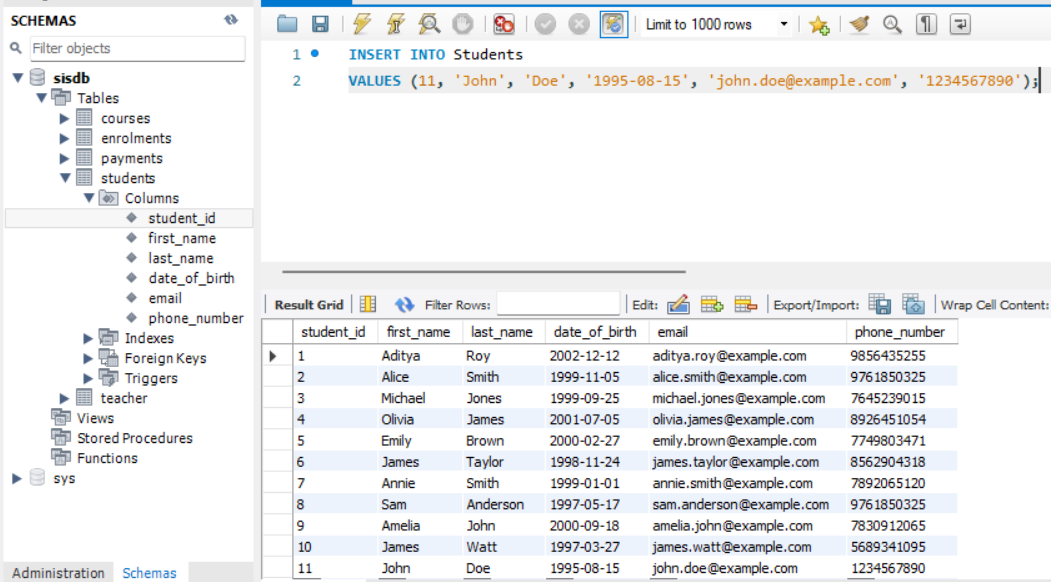
c. Date of Birth: 1995-08-15

d. Email: john.doe@example.com

e. Phone Number: 1234567890

INSERT INTO Students

VALUES (11, 'John', 'Doe', '1995-08-15', 'john.doe@example.com', '1234567890');

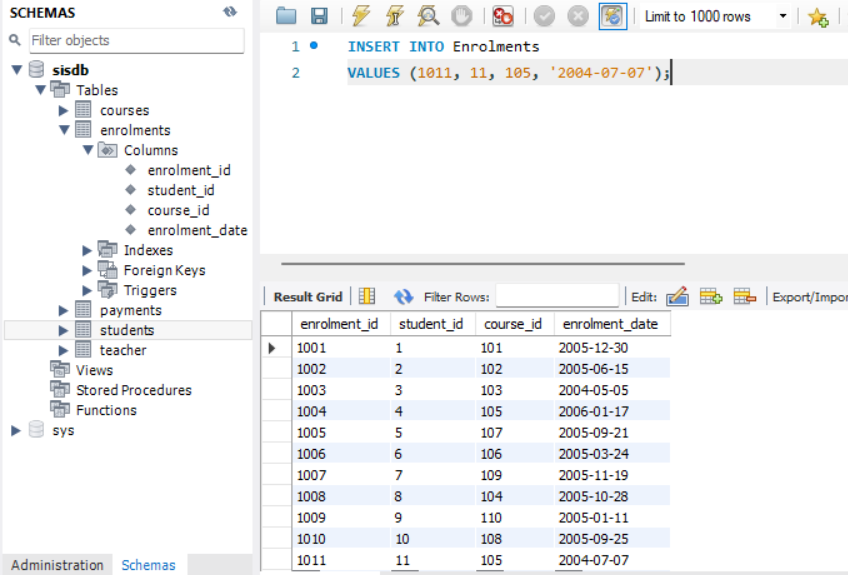


2. Write an SQL query to enrol a student in a course. Choose an existing student and course and

insert a record into the "Enrolments" table with the enrolment date.

INSERT INTO Enrolments

VALUES (1011, 11, 105, '2004-07-07');

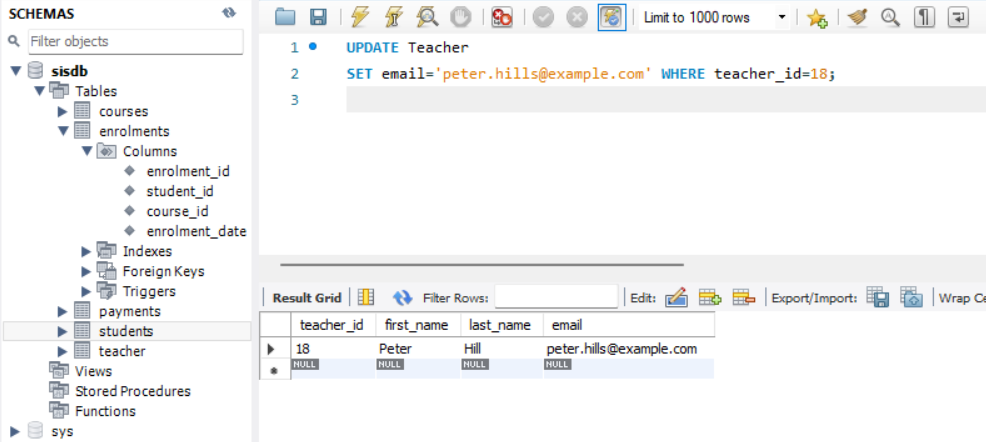


3. Update the email address of a specific teacher in the "Teacher" table. Choose any teacher and

modify their email address.

UPDATE Teacher

SET email='peter.hills@example.com' WHERE teacher\_id=18;

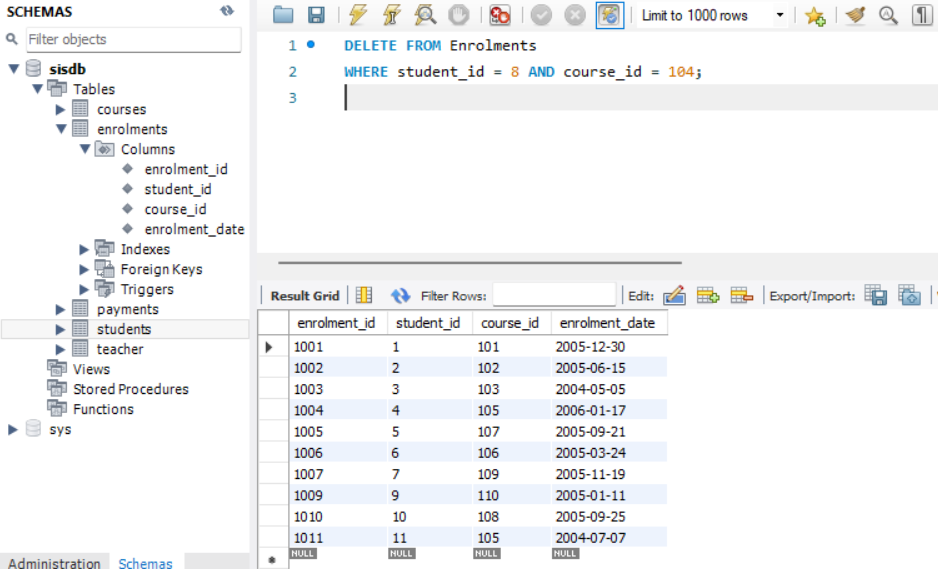


4. Write an SQL query to delete a specific enrolment record from the "Enrolments" table. Select

an enrolment record based on the student and course.

DELETE FROM Enrolments

WHERE student\_id = 8 AND course\_id = 104;

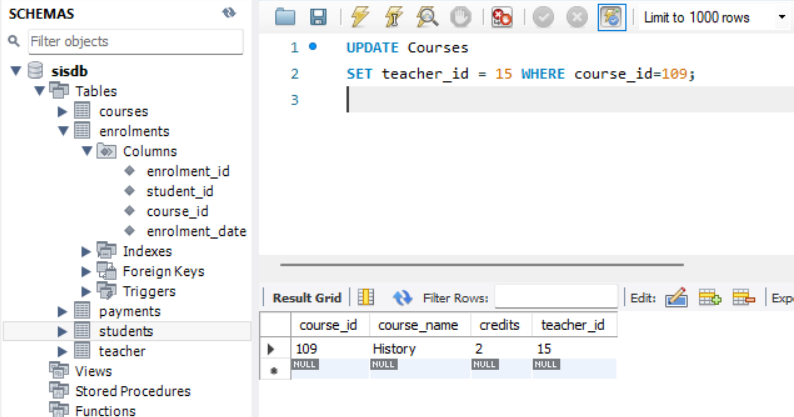


5. Update the "Courses" table to assign a specific teacher to a course. Choose any course and

teacher from the respective tables.

UPDATE Courses

SET teacher\_id = 15 WHERE course\_id=109;



6. Delete a specific student from the "Students" table and remove all their enrolment records

from the "Enrolments" table. Be sure to maintain referential integrity.

SET foreign\_key\_checks = 0;

DELETE FROM Enrolments

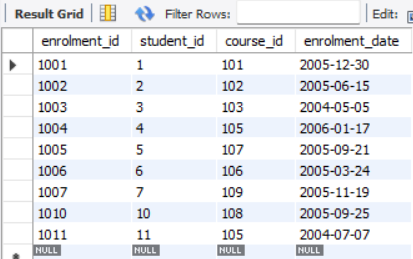
WHERE student\_id =9;

DELETE FROM Students

WHERE student\_id=9;

SET foreign\_key\_checks = 1;

Enrolments Table:



Students Table:



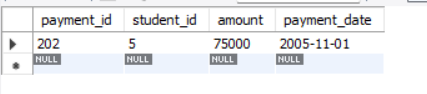
7. Update the payment amount for a specific payment record in the "Payments" table. Choose any

payment record and modify the payment amount.

UPDATE Payments

SET amount=75000

WHERE payment\_id=202;



**TASK 3: AGGREGATE FUNCTIONS, HAVING, ORDER BY, GROUP BY AND JOINS**

1. Write an SQL query to calculate the total payments made by a specific student. You will need to

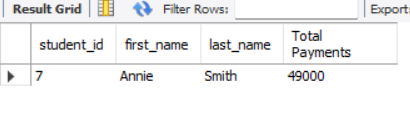
join the "Payments" table with the "Students" table based on the student's ID.

SELECT Students.student\_id, Students.first\_name, Students.last\_name, SUM(Payments.amount) AS

'Total Payments' FROM Students

INNER JOIN Payments ON Students.student\_id = Payments.student\_id

WHERE Students.student\_id = 7;



2. Write an SQL query to retrieve a list of courses along with the count of students enrolled in each

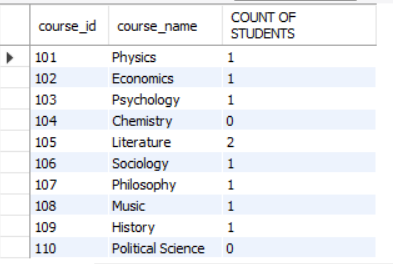
course. Use a JOIN operation between the "Courses" table and the "Enrolments" table.

SELECT Courses.course\_id, Courses.course\_name, COUNT(Enrolments.student\_id) AS 'COUNT OF STUDENTS'

FROM Courses

LEFT JOIN Enrolments ON Courses.course\_id = Enrolments.course\_id

GROUP BY Courses.course\_id, Courses.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 "Enrolments" table to identify students

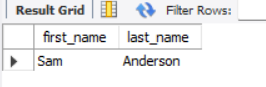
without enrolments.

SELECT Students.first\_name, Students.last\_name

FROM Students

LEFT JOIN Enrolments ON Student.student\_id=Enrolments.student\_id

WHERE Enrolments.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

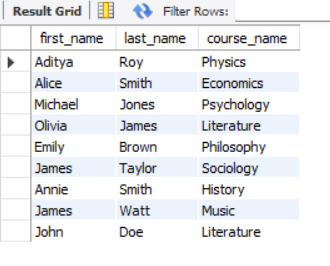
"Enrolments" and "Courses" tables.

SELECT Students.first\_name, Students.last\_name, Courses.course\_name

FROM Students

JOIN Enrolments ON Students.student\_id=Enrolments.student\_id

JOIN Courses ON Enrolments.course\_id=Courses.course\_id;



5. Create a query to list the names of teachers and the courses they are assigned to. Join the

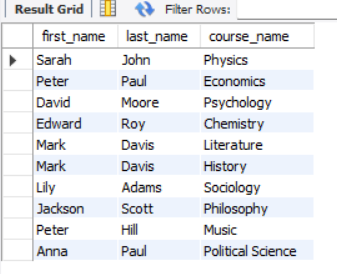
"Teacher" table with the "Courses" table.

SELECT Teacher.first\_name, Teacher.last\_name, Courses.course\_name

FROM Teacher

JOIN Courses

ON Teacher.teacher\_id = Courses.teacher\_id;



6. Retrieve a list of students and their enrolment dates for a specific course. You'll need to join the

"Students" table with the "Enrolments" and "Courses" tables.

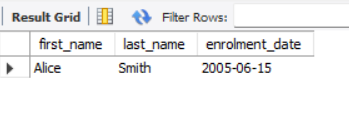
SELECT Students.first\_name,Students.last\_name, Enrolments.enrolment\_date

FROM Students

JOIN Enrolments ON Students.student\_id = Enrolments.student\_id

JOIN Courses ON Enrolments.course\_id = Courses.course\_id

WHERE Courses.course\_name ='Economics';



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.

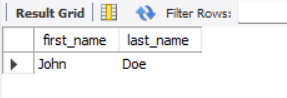
SELECT Students.first\_name, Students.last\_name

FROM Students

LEFT JOIN Payments

ON Students.student\_id=Payments.student\_id

WHERE Payments.payment\_id IS NULL;



8. Write a query to identify courses that have no enrolments. You'll need to use a LEFT JOIN

between the "Courses" table and the "Enrolments" table and filter for courses with NULL

enrolment records.

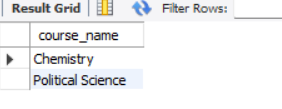
SELECT Courses.course\_name

FROM Courses

LEFT JOIN Enrolments

ON Courses.course\_id=Enrolments.course\_id

WHERE Enrolments.enrolment\_id IS NULL;



9. Identify students who are enrolled in more than one course. Use a self-join on the "Enrolments"

table to find students with multiple enrolment records.

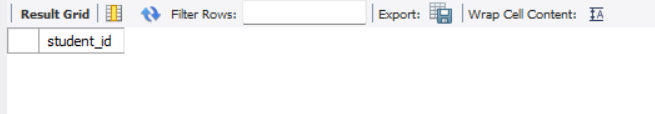
SELECT DISTINCT e1.student\_id

FROM Enrolments e1

JOIN Enrolments e2

ON e1.student\_id=e2.student\_id

WHERE e1.course\_id <> e2.course\_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.

SELECT Teacher.first\_name, Teacher.last\_name

FROM Teacher

LEFT JOIN Courses

ON Teacher.teacher\_id=Courses.teacher\_id

WHERE Courses.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\_id, AVG(student\_count) AS 'Average no of students'

FROM (

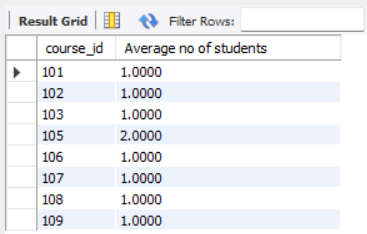
SELECT course\_id, COUNT(student\_id) AS student\_count

FROM Enrolments

GROUP BY course\_id

) AS count

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.

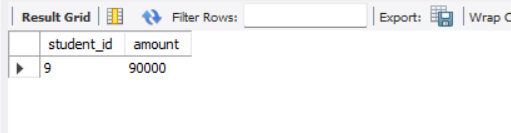
SELECT student\_id, amount

FROM Payments

WHERE amount = (

SELECT MAX(amount)

FROM Payments);



3. Retrieve a list of courses with the highest number of enrolments. Use subqueries to find the

course(s) with the maximum enrolment count.

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

FROM Courses c

JOIN Enrolments e

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

GROUP BY c.course\_id

HAVING enrolment\_count = (

SELECT MAX(enrolment\_count)

FROM (

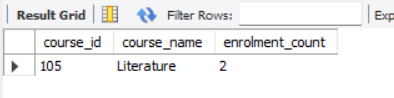
SELECT COUNT(enrolment\_id) AS enrolment\_count

FROM Enrolments

GROUP BY course\_id

) AS max\_enrolment

);



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\_payment

FROM Teacher t

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

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

LEFT JOIN Payments p ON e.student\_id = p.student\_id

GROUP BY t.teacher\_id

HAVING t.teacher\_id IN (

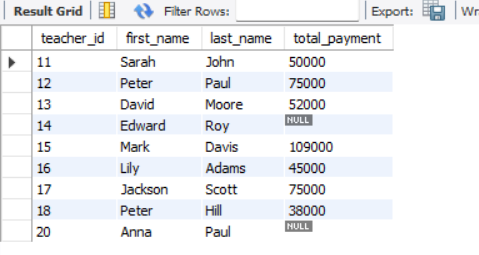
SELECT c.teacher\_id

FROM Courses c

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

GROUP BY c.teacher\_id

);



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

student's enrolments with the total number of courses.

SELECT student\_id, COUNT(course\_id) AS enroled\_courses

FROM Enrolments

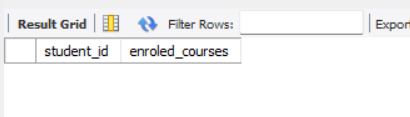
GROUP BY student\_id

HAVING enroled\_courses = (

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.

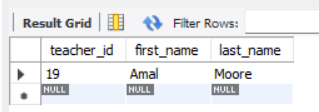
SELECT teacher\_id, first\_name, last\_name

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.

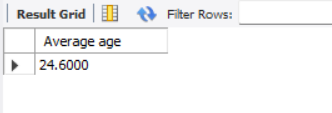
SELECT AVG(age) AS 'Average age'

FROM (

SELECT TIMESTAMPDIFF(YEAR, date\_of\_birth, CURDATE()) AS age

FROM Students

) AS student\_age;



8. Identify courses with no enrolments. Use subqueries to find courses without enrolment

records.

SELECT course\_id, course\_name

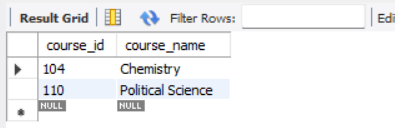
FROM Courses

WHERE course\_id NOT IN(

SELECT DISTINCT course\_id

FROM Enrolments

);



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 student\_id,course\_id, SUM(amount) AS 'Total Payment'

FROM(

SELECT s.student\_id, e.course\_id, p.amount

FROM Students s

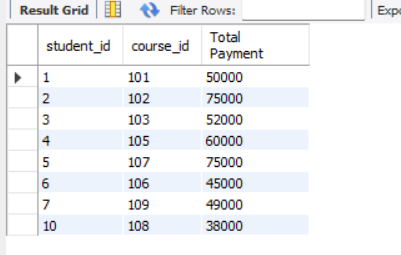
JOIN Enrolments e

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

JOIN Payments p

ON e.student\_id = p.student\_id) AS paymentmade

GROUP BY student\_id, 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.

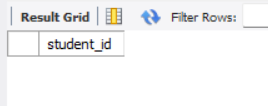
SELECT student\_id

FROM (SELECT student\_id, COUNT(payment\_id) AS payment\_count

FROM Payments

GROUP BY student\_id) AS payment\_counts

WHERE 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.

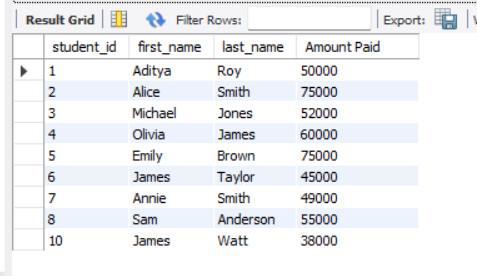
SELECT s.student\_id, s.first\_name, s.last\_name, SUM(p.amount) AS 'Amount Paid'

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 "Enrolments" table and GROUP BY to

count enrolments.

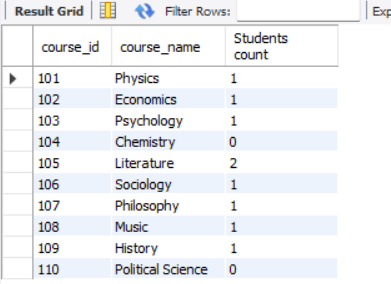
SELECT c.course\_id, c.course\_name, COUNT(e.student\_id) AS 'Students count'

FROM Courses c

LEFT JOIN Enrolments 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, 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;

