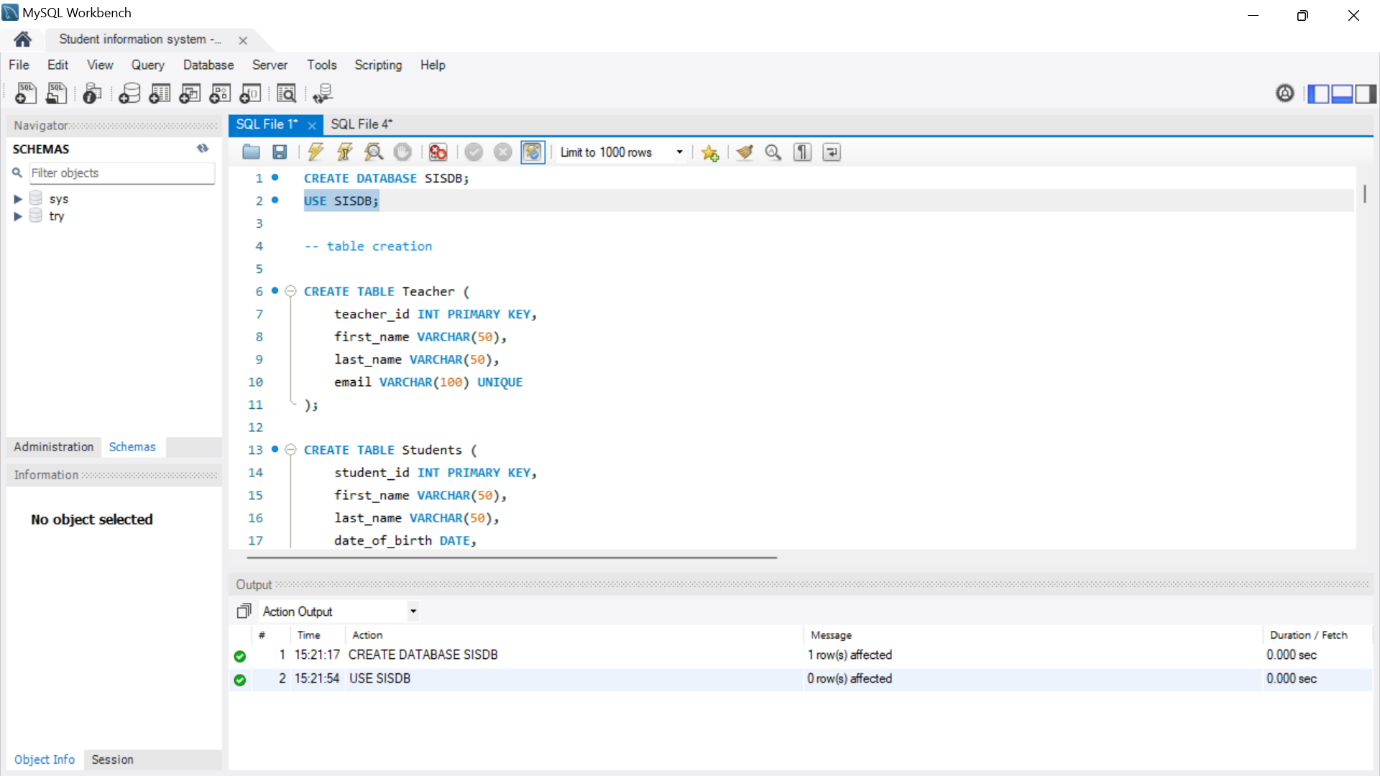
**TASK1)**

-- **database creation**

CREATE DATABASE SISDB;

USE SISDB;



**-- table creation**

CREATE TABLE Teacher (

teacher\_id INT PRIMARY KEY,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

email VARCHAR(100) UNIQUE

);

CREATE TABLE Students (

student\_id INT PRIMARY KEY,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

date\_of\_birth DATE,

email VARCHAR(100) UNIQUE,

phone\_number VARCHAR(20)

);

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)

);

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 Payments (

payment\_id INT PRIMARY KEY,

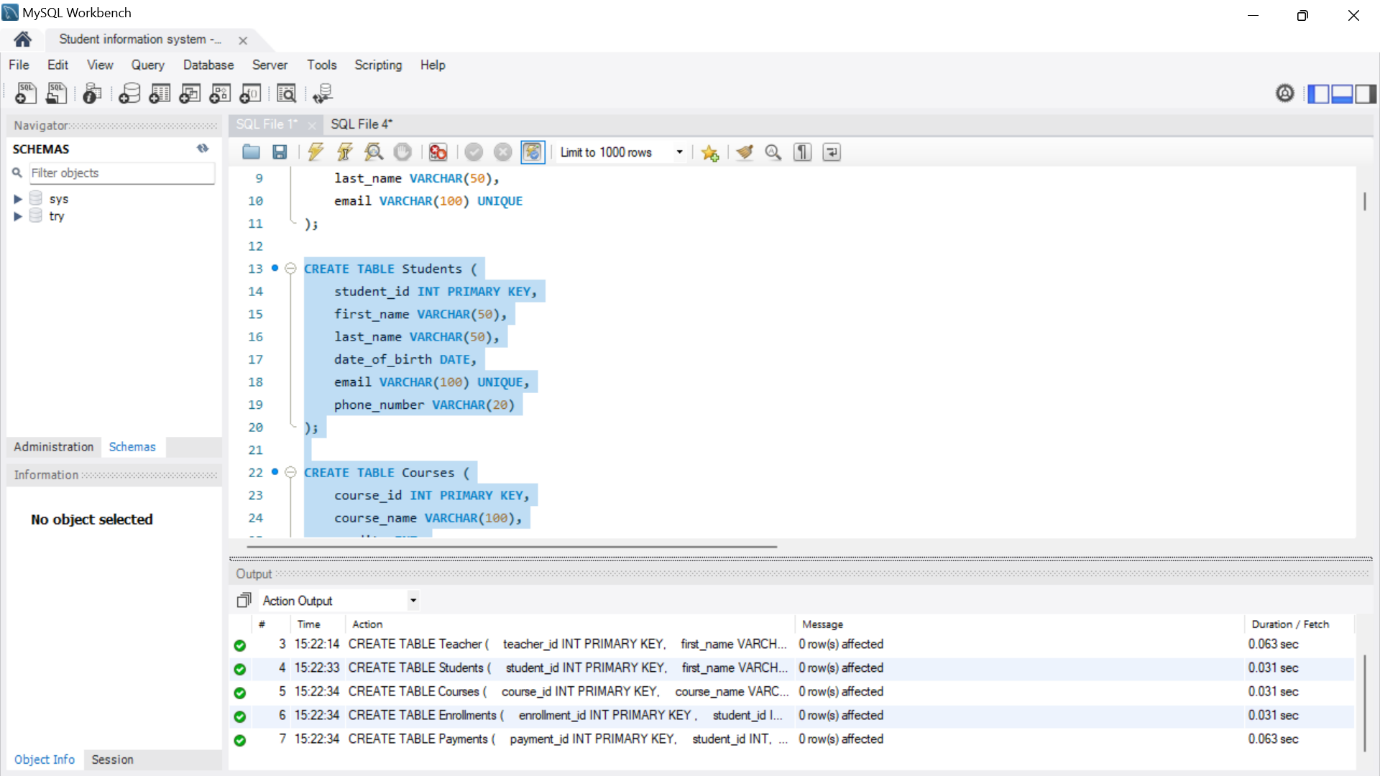
student\_id INT,

amount DECIMAL(10, 2),

payment\_date DATE,

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

);

****

**NORMALIZATION-**

/\*

the first three normal forms (1NF, 2NF, 3NF).

1. Students Table:

1NF:

The table is in 1NF as it contains only atomic values in each cell.

2NF:

The table is in 2NF since it has a single-column primary key, and there are no partial dependencies.

3NF:

The table is in 3NF. All attributes are functionally dependent on the primary key (student\_id).

2. Courses Table:

1NF:

The table is in 1NF as it contains only atomic values in each cell.

2NF:

The table is in 2NF since it has a single-column primary key, and there are no partial dependencies.

3NF:

The table is in 3NF. All non-prime attributes are functionally dependent on the primary key (course\_id).

3. Enrollments Table:

1NF:

The table is in 1NF as it contains only atomic values in each cell.

2NF:

The table is in 2NF since it has a single-column primary key, and there are no partial dependencies.

3NF:

The table is in 3NF. All non-prime attributes are functionally dependent on the primary key (enrollment\_id).

4. Teacher Table:

1NF:

The table is in 1NF as it contains only atomic values in each cell.

2NF:

The table is in 2NF since it has a single-column primary key, and there are no partial dependencies.

3NF:

The table is in 3NF. All attributes are functionally dependent on the primary key (teacher\_id).

5. Payments Table:

1NF:

The table is in 1NF as it contains only atomic values in each cell.

2NF:

The table is in 2NF since it has a single-column primary key, and there are no partial dependencies.

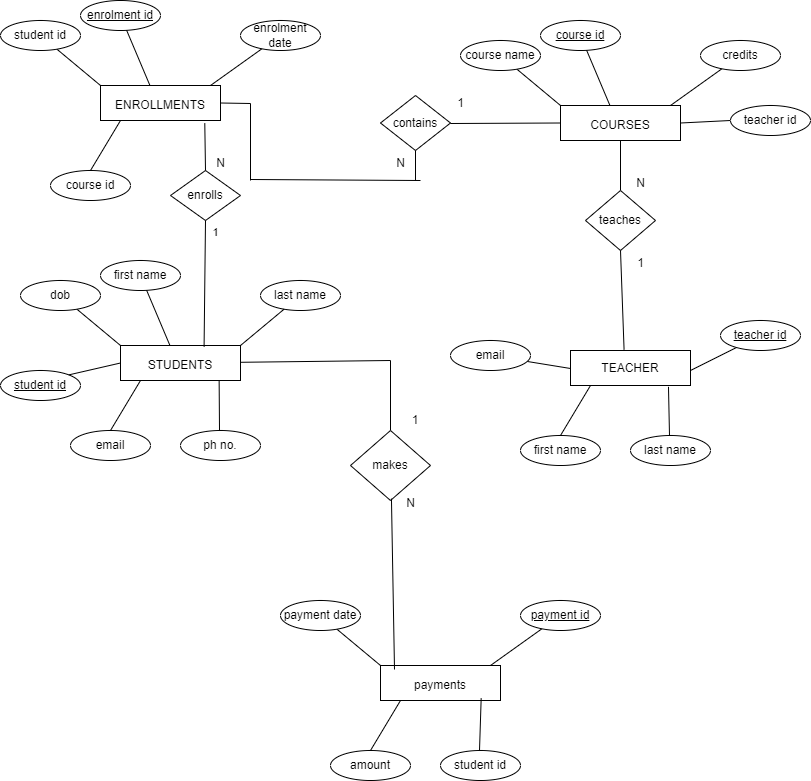
3NF:

The table is in 3NF. All non-prime attributes are functionally dependent on the primary key (payment\_id).

all tables have atomic values, no partial dependencies on the primary key, and no transitive dependencies.

\*/

**ER DIAGRAM**



**TASK2)**

**-- values insertion**

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

VALUES

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

(302, 'Isha', 'Singh', '1998-07-12', 'isha.singh@example.com', '8765432109'),

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

(304, 'Anaya', 'Patel', '1996-12-02', 'anaya.patel@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, 'Physics', 4, 101),

(402, 'Art', 3, 102),

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

(404, 'Literature', 4, 104),

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

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

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

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

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

(410, 'Economics', 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'),

(531, 302, 406, '2023-06-20'),

(532, 302, 407, '2023-07-05'),

(533, 303, 408, '2023-07-15'),

(534, 304, 409, '2023-08-01'),

(535, 305, 410, '2023-08-10');

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

VALUES

(601, 301, 80.00, '2023-01-10'),

(602, 302, 60.50, '2023-02-15'),

(603, 303, 90.00, '2023-03-01'),

(604, 304, 75.00, '2023-02-20'),

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

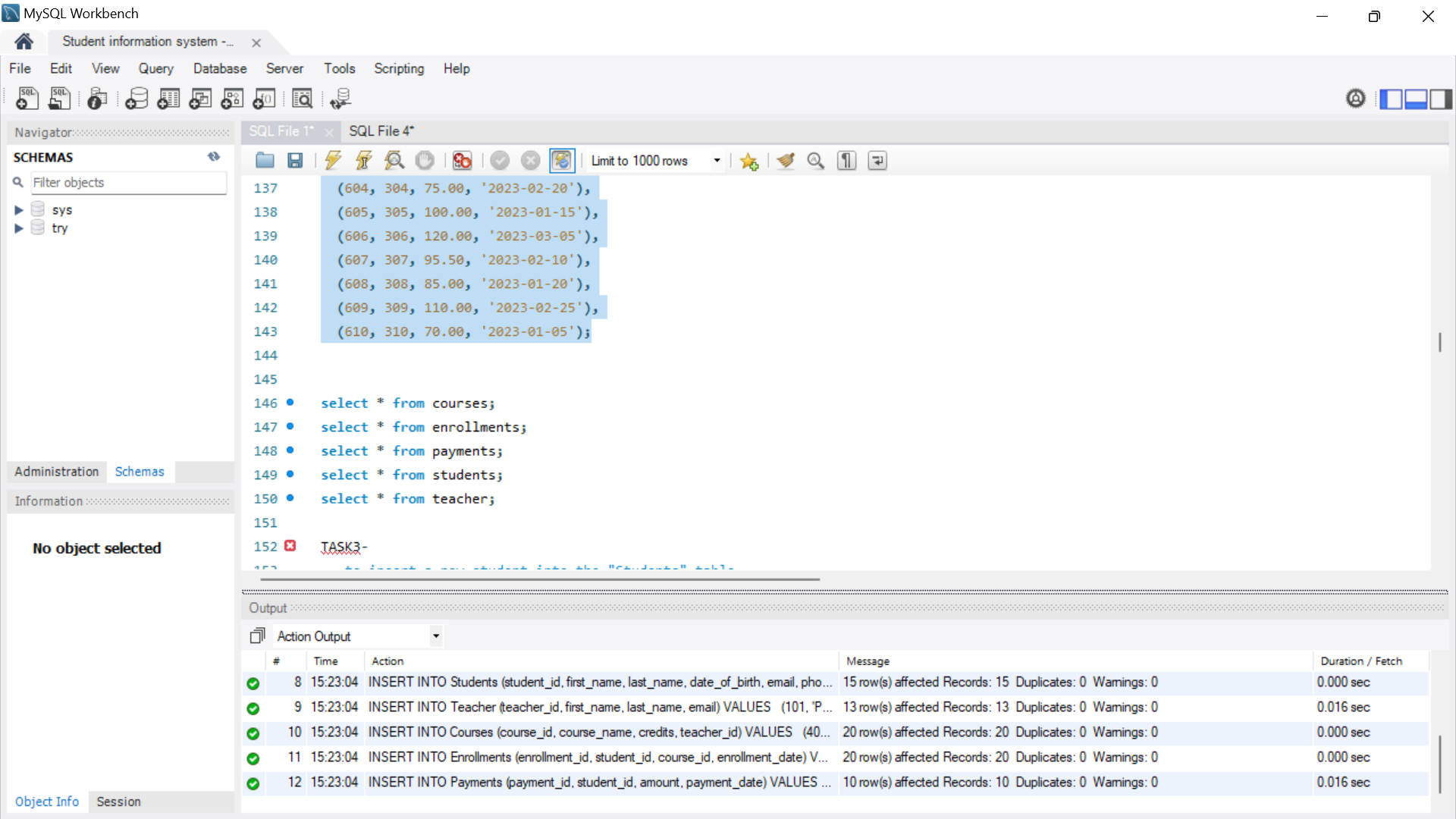
(606, 306, 120.00, '2023-03-05'),

(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');

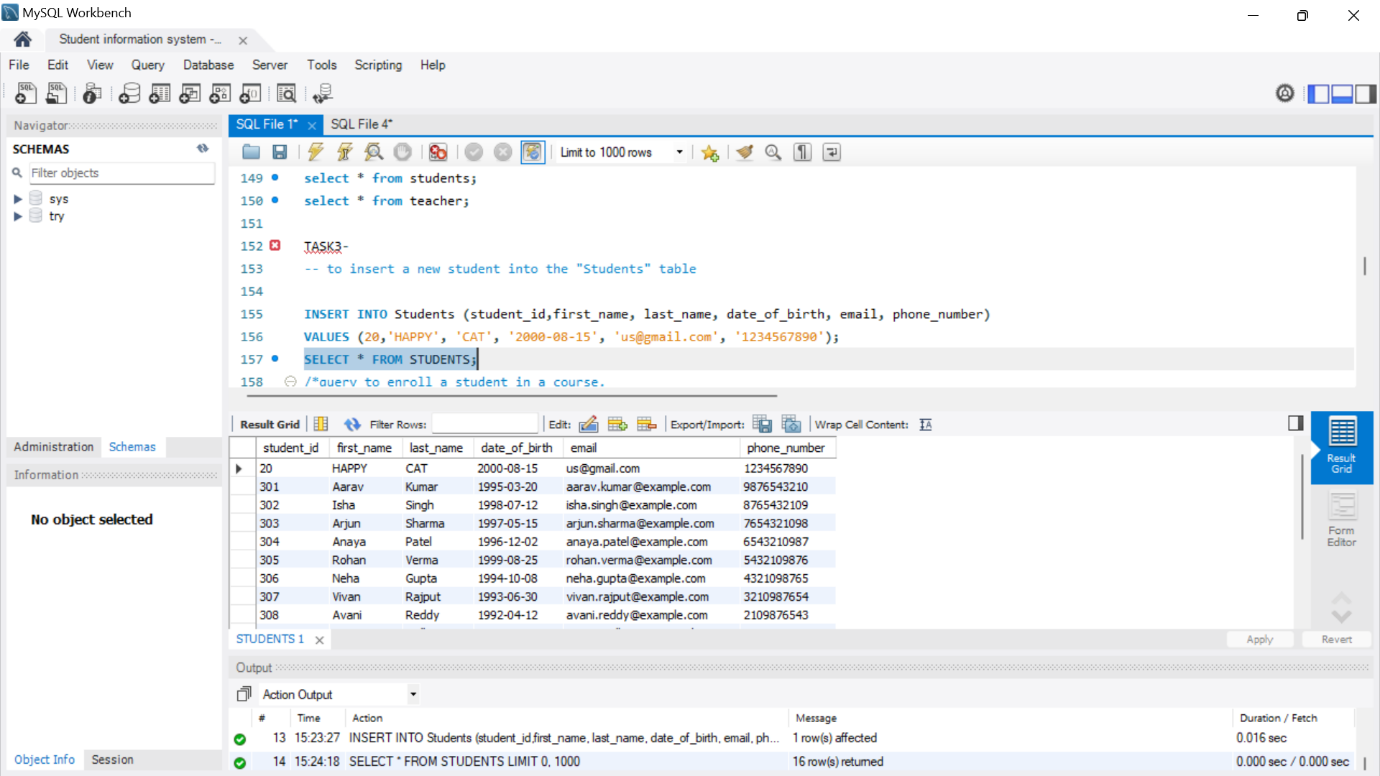


**TASK3-**

**-- to insert a new student into the "Students" table**

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

VALUES (20,'HAPPY', 'CAT', '2000-08-15', 'us@gmail.com', '1234567890');



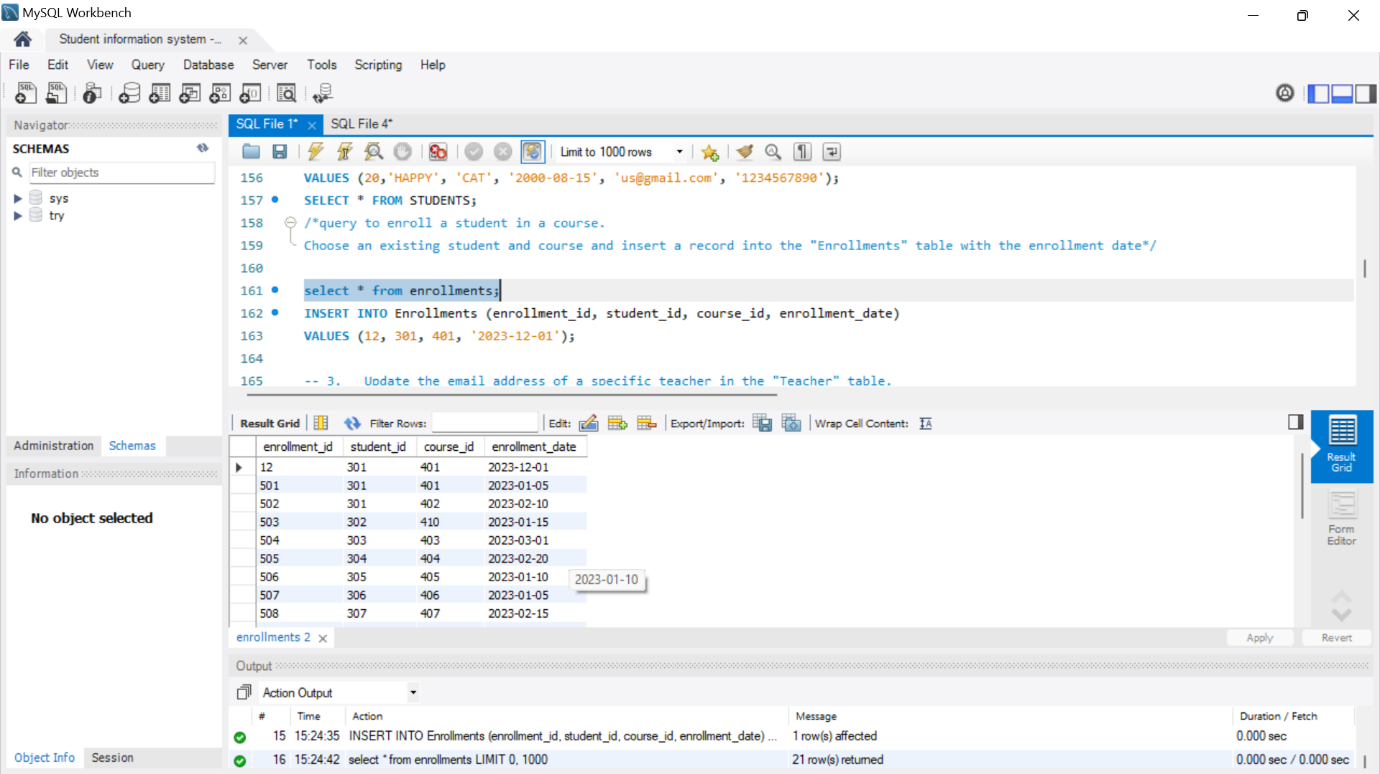
**/\*query to enroll a student in a course.**

**Choose an existing student and course and insert a record into the "Enrollments" table with the enrollment date\*/**

select \* from enrollments;

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

VALUES (12, 301, 401, '2023-12-01');



**-- 3. Update the email address of a specific teacher in the "Teacher" table.**

UPDATE Teacher SET email = 'mew@gmailcom' WHERE teacher\_id = 107;

select \* from teacher;

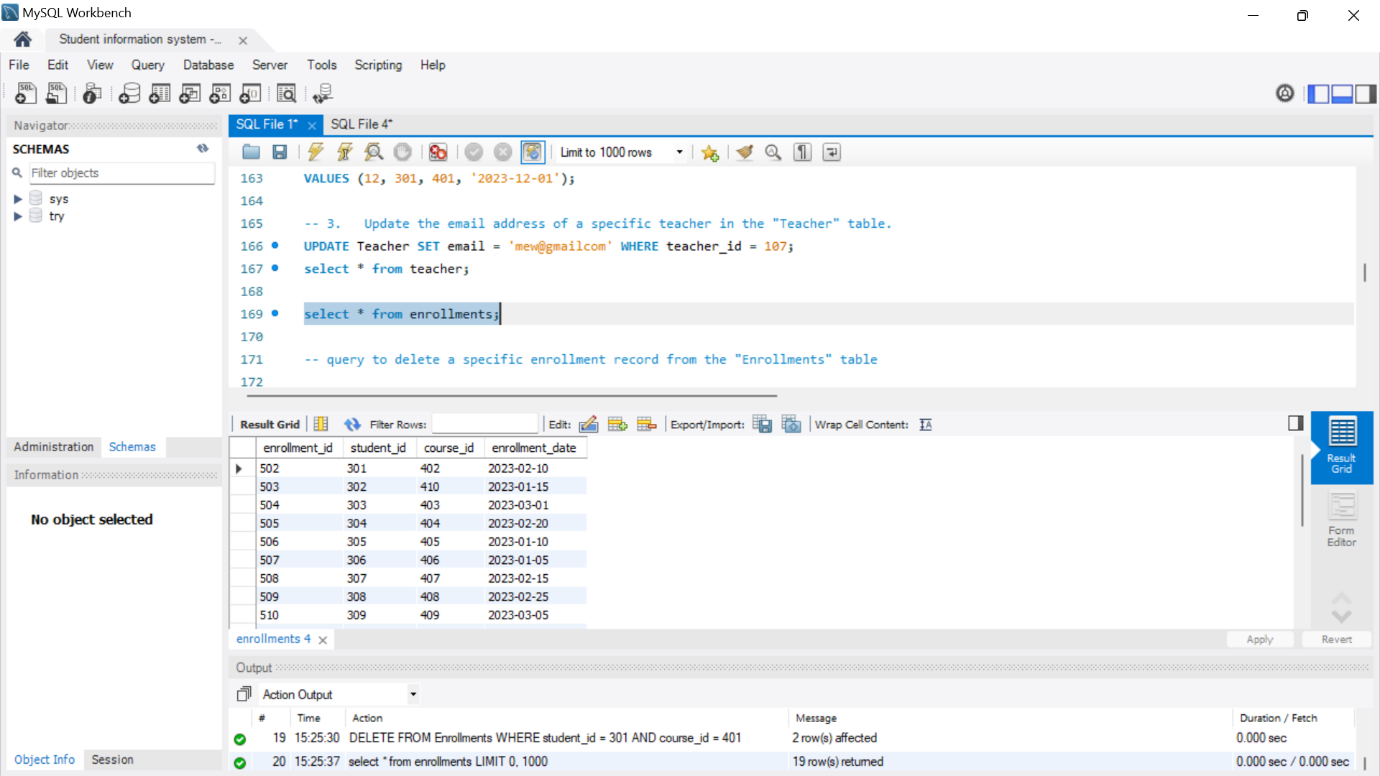


**-- query to delete a specific enrollment record from the "Enrollments" table**

DELETE FROM Enrollments

WHERE student\_id = 301 AND course\_id = 401;

select \* from enrollments;

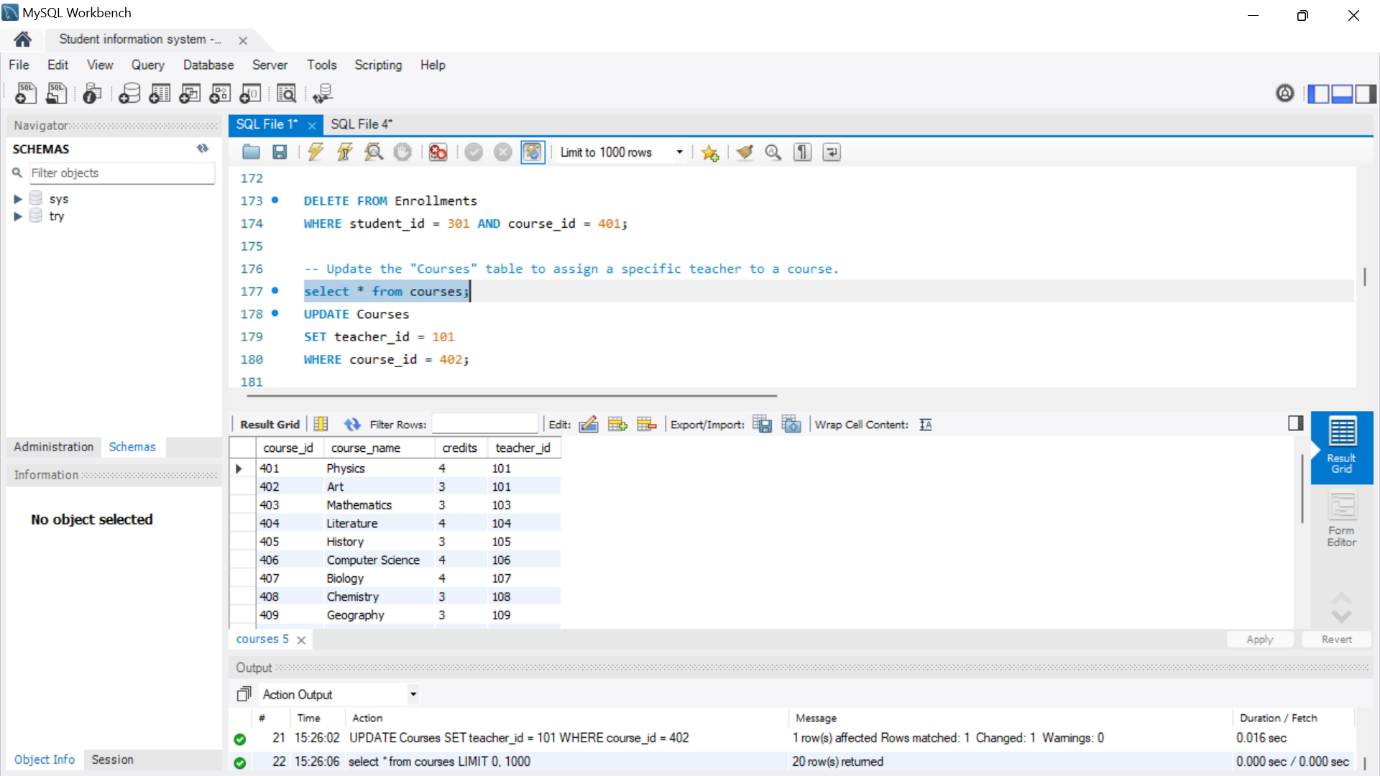


**-- Update the "Courses" table to assign a specific teacher to a course.**

select \* from courses;

UPDATE Courses

SET teacher\_id = 101

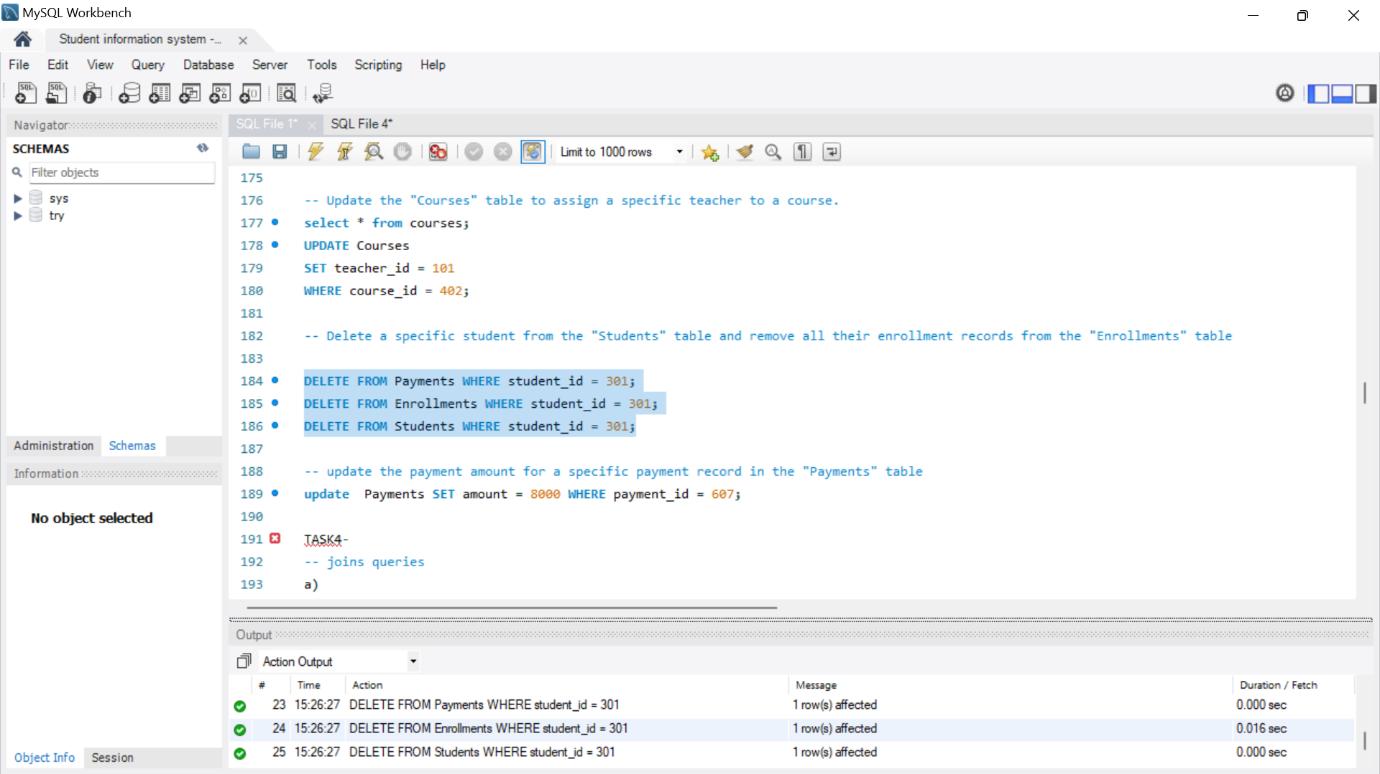
WHERE course\_id = 402; 

**-- Delete a specific student from the "Students" table and remove all their enrollment records from the "Enrollments" table**

DELETE FROM Payments WHERE student\_id = 301;

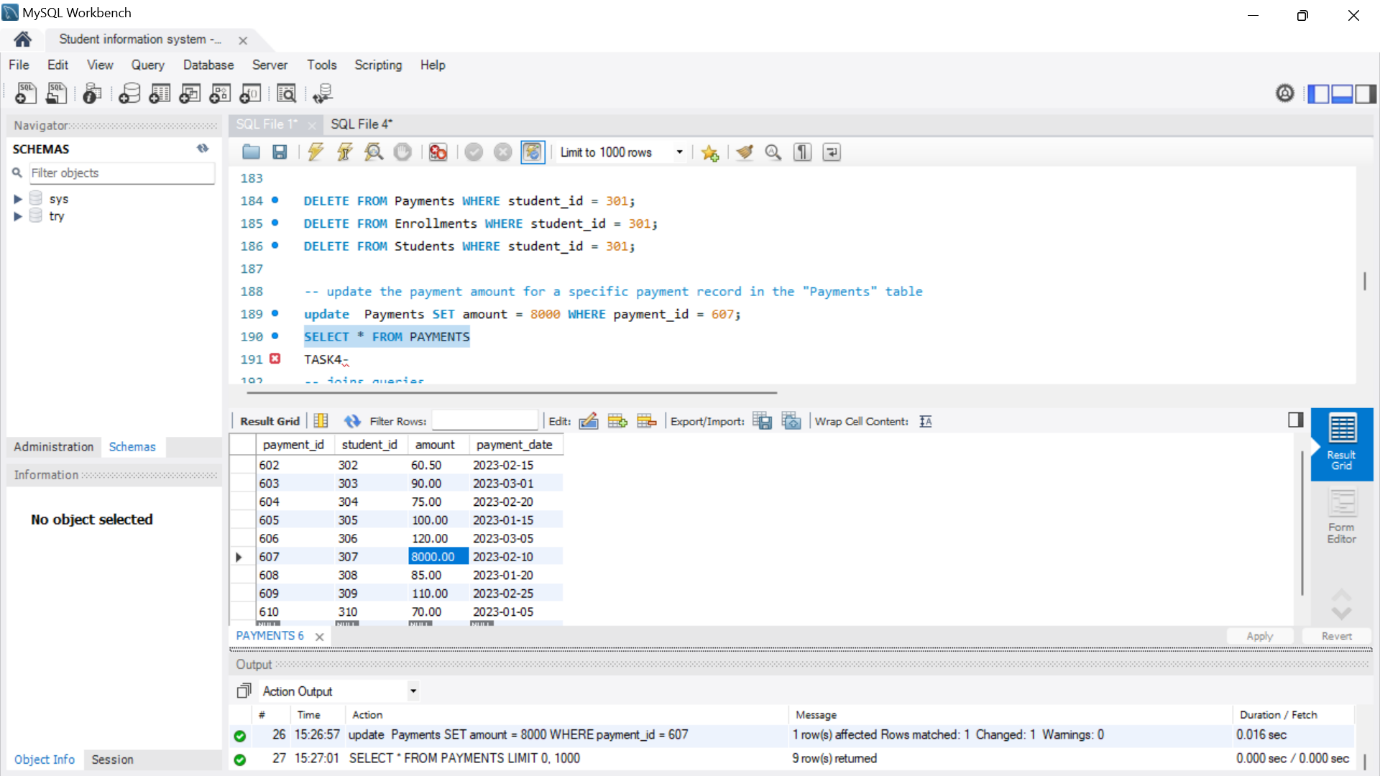
DELETE FROM Enrollments WHERE student\_id = 301;

DELETE FROM Students WHERE student\_id = 301;



**-- update the payment amount for a specific payment record in the "Payments" table**

update Payments SET amount = 8000 WHERE payment\_id = 607;



**TASK4-**

-- joins queries

**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.**

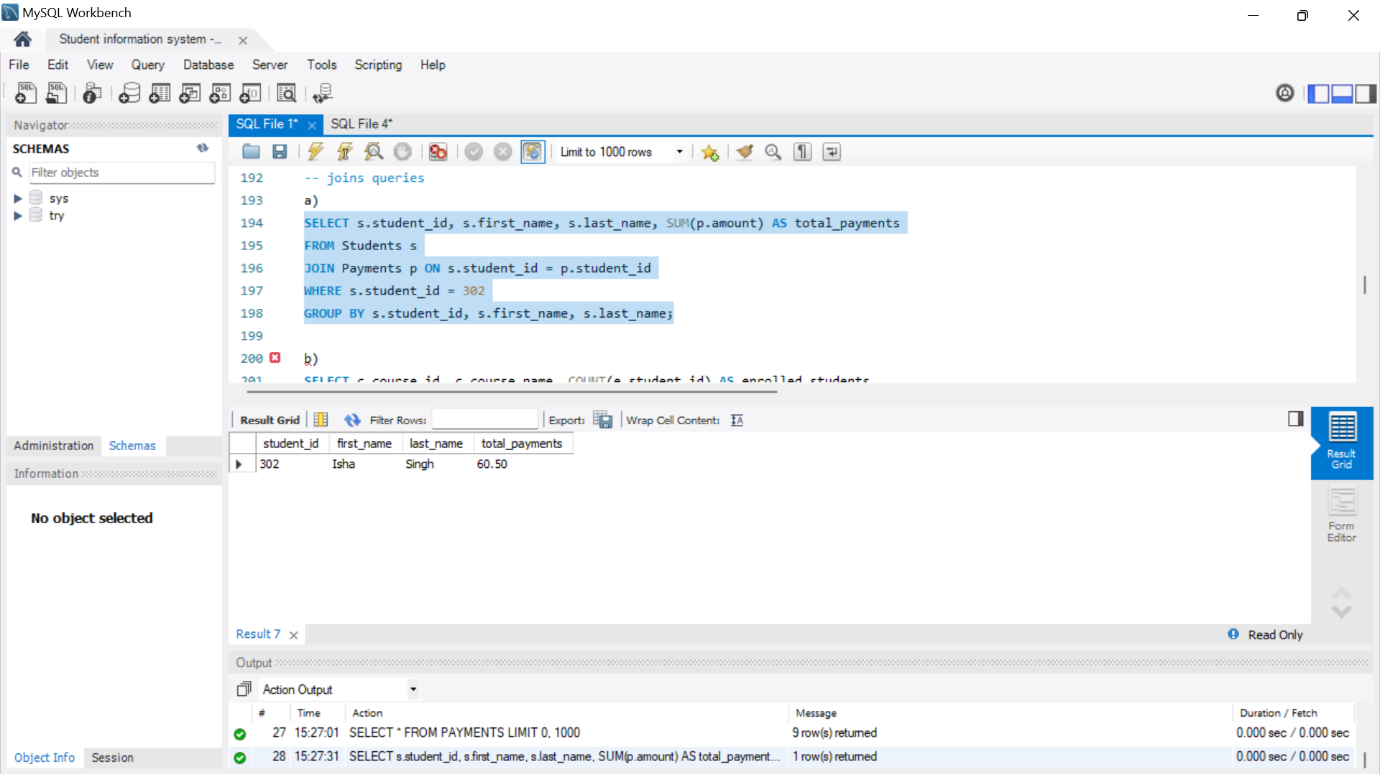
SELECT s.student\_id, s.first\_name, s.last\_name, SUM(p.amount) AS total\_payments

FROM Students s

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

WHERE s.student\_id = 302

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



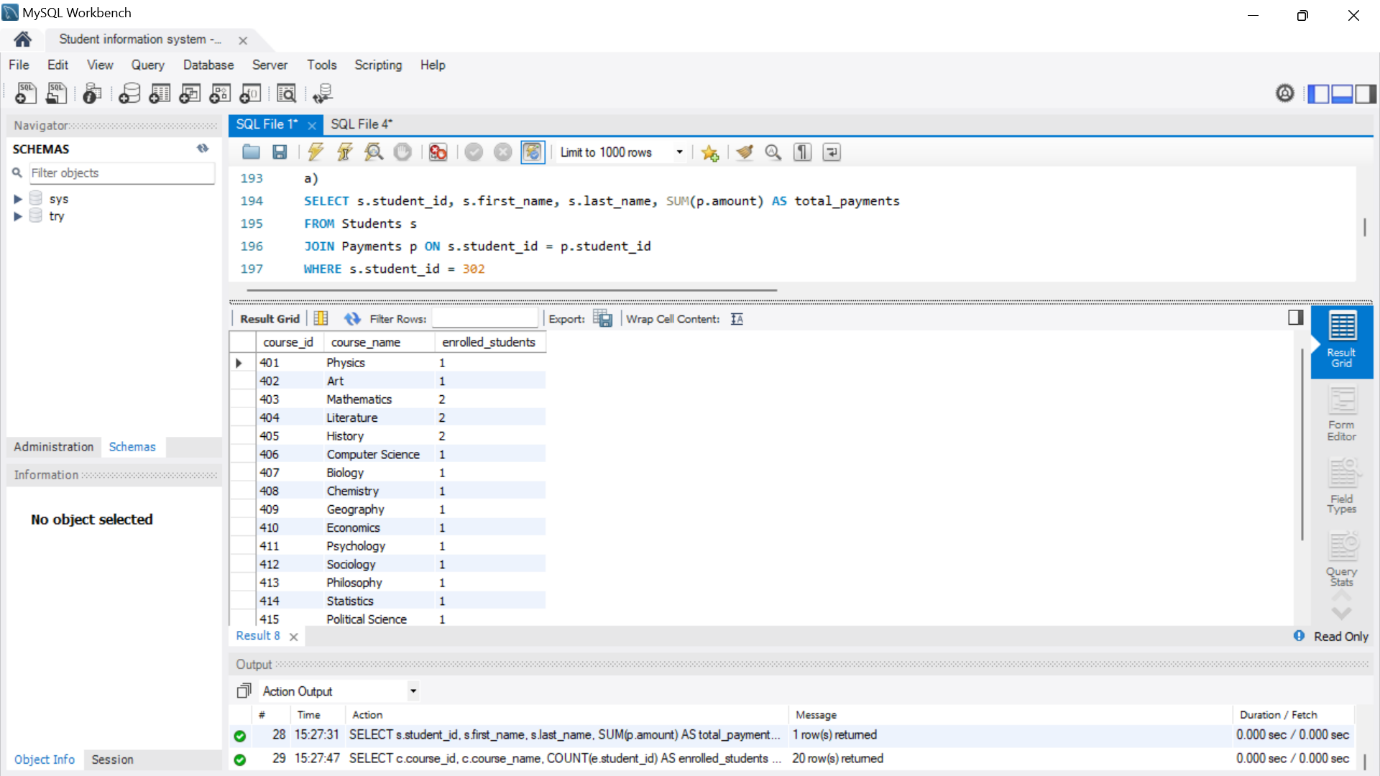
**2. Write an SQL query to retrieve a list of courses along with the count of students enrolled in each course. Use a JOIN operation between the "Courses" table and the "Enrollments" table.**

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

FROM Courses c

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

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



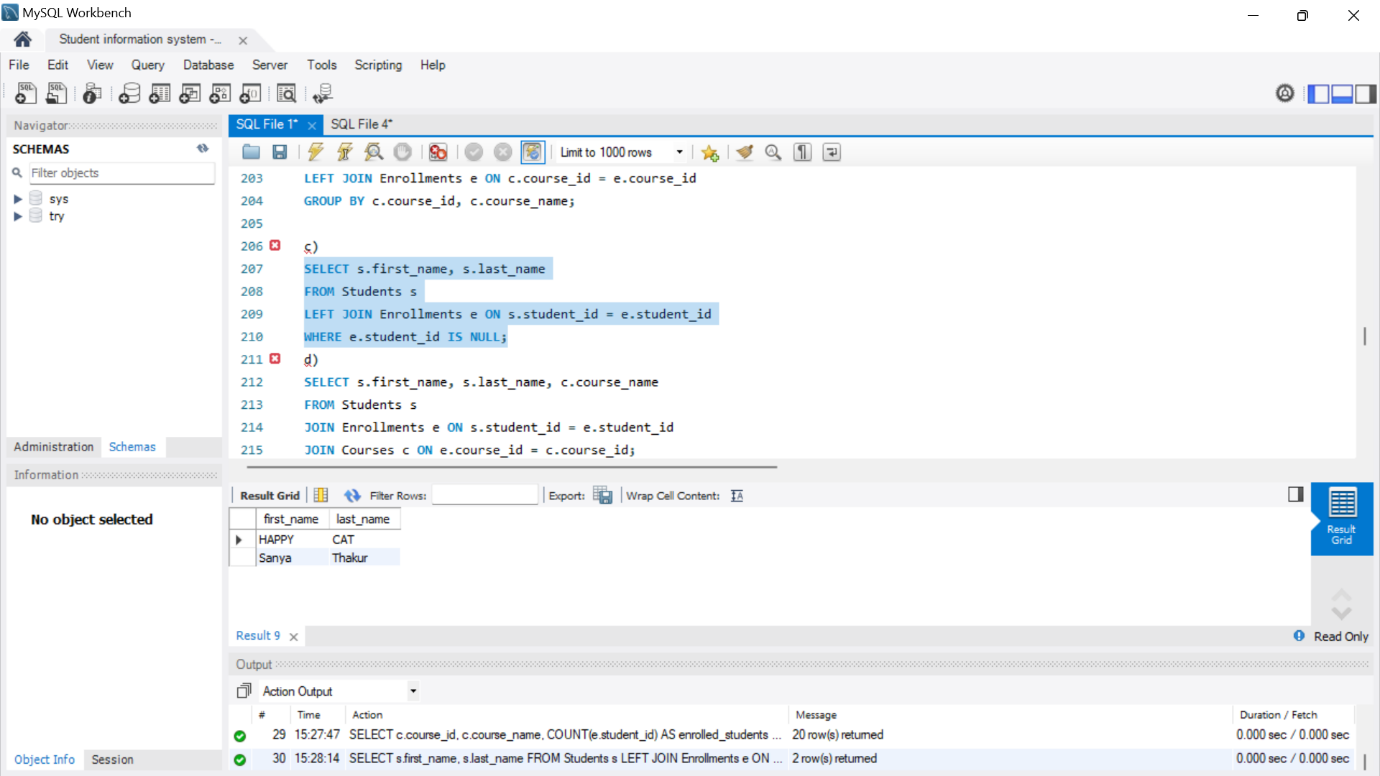
**3. Write an SQL query to find the names of students who have not enrolled in any course. Use a LEFT JOIN between the "Students" table and the "Enrollments" table to identify students without enrollments.**

SELECT 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;



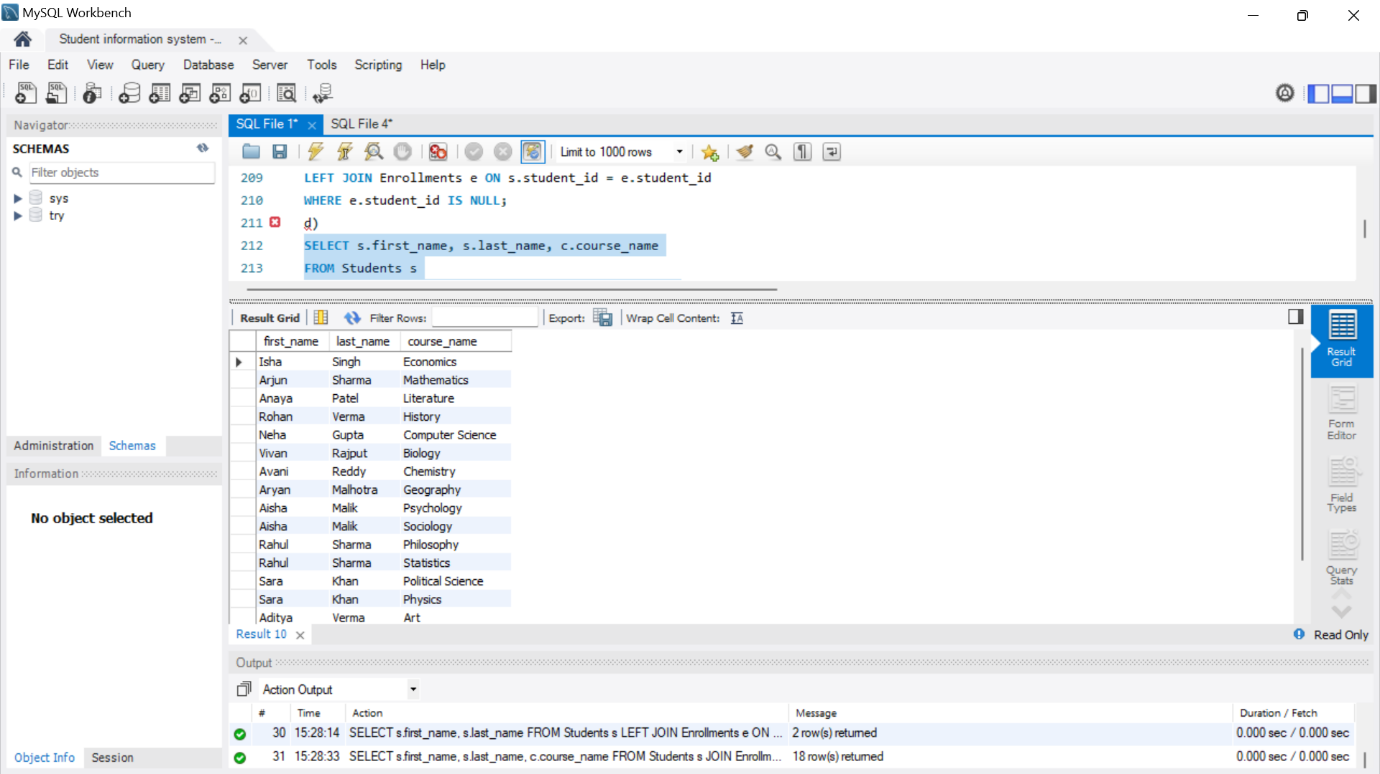
**4. Write an SQL query to retrieve the first name, last name of students, and the names of the courses they are enrolled in. Use JOIN operations between the "Students" table and the "Enrollments" and "Courses" tables.**

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;

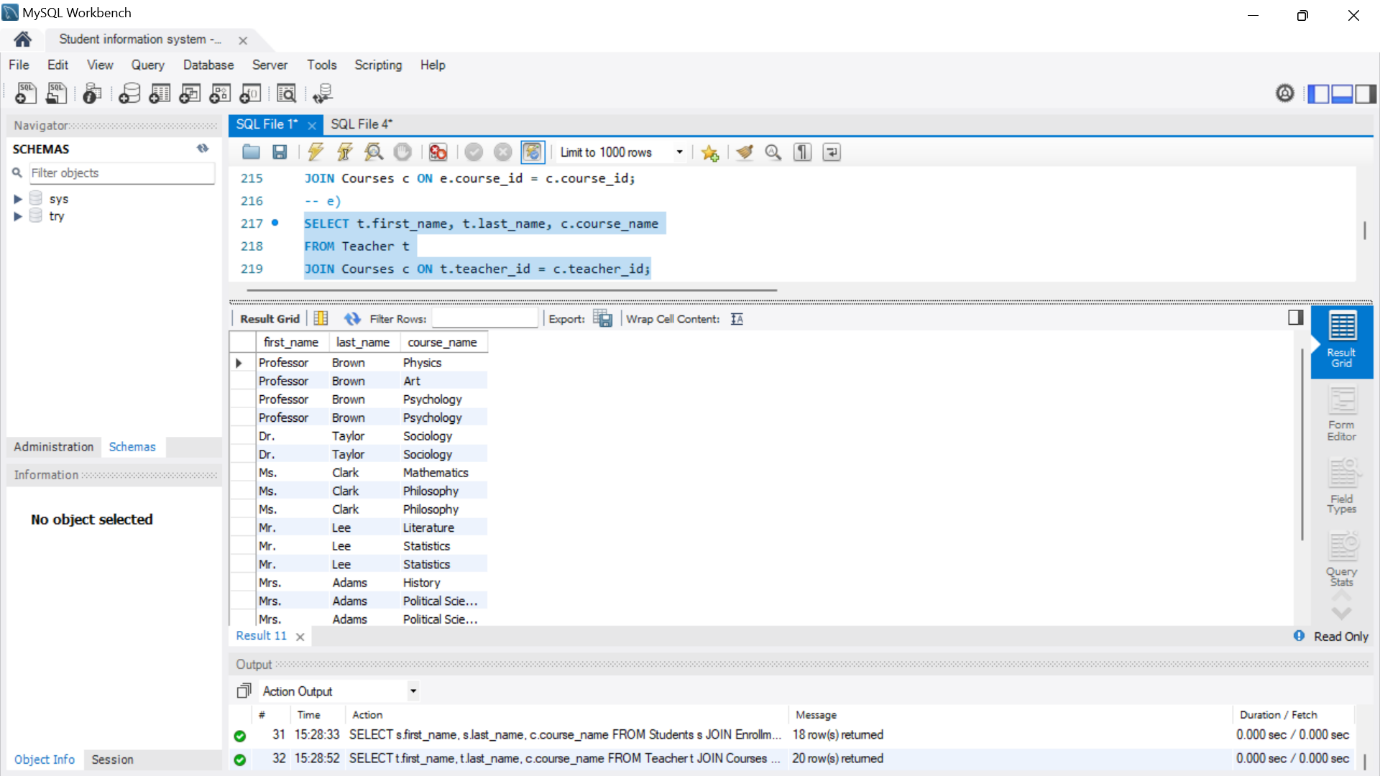


**5. Create a query to list the names of teachers and the courses they are assigned to. Join the "Teacher" table with the "Courses" table.**

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

FROM Teacher t

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



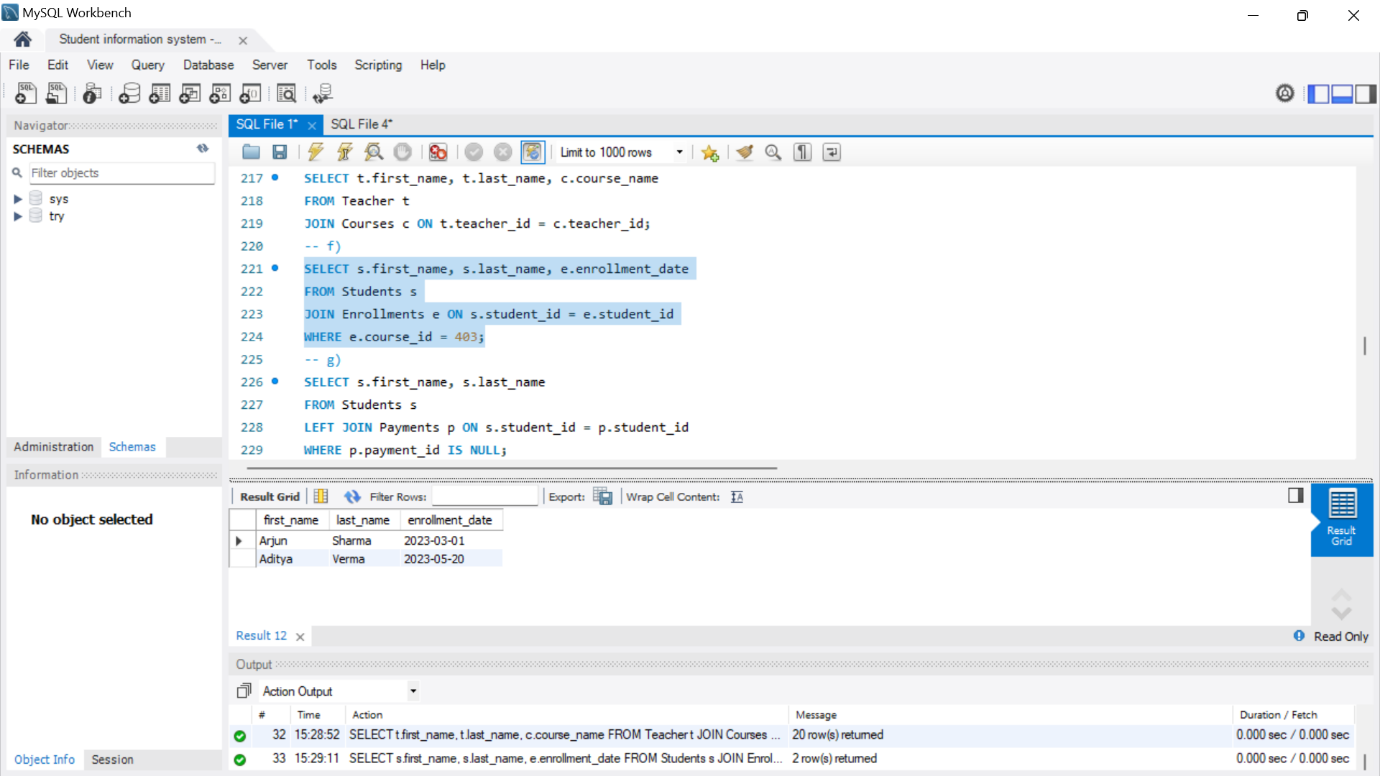
**6. Retrieve a list of students and their enrollment dates for a specific course. You'll need to join the "Students" table with the "Enrollments" and "Courses" tables..**

SELECT s.first\_name, s.last\_name, e.enrollment\_date

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 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;

-- Insert additional values into Courses table



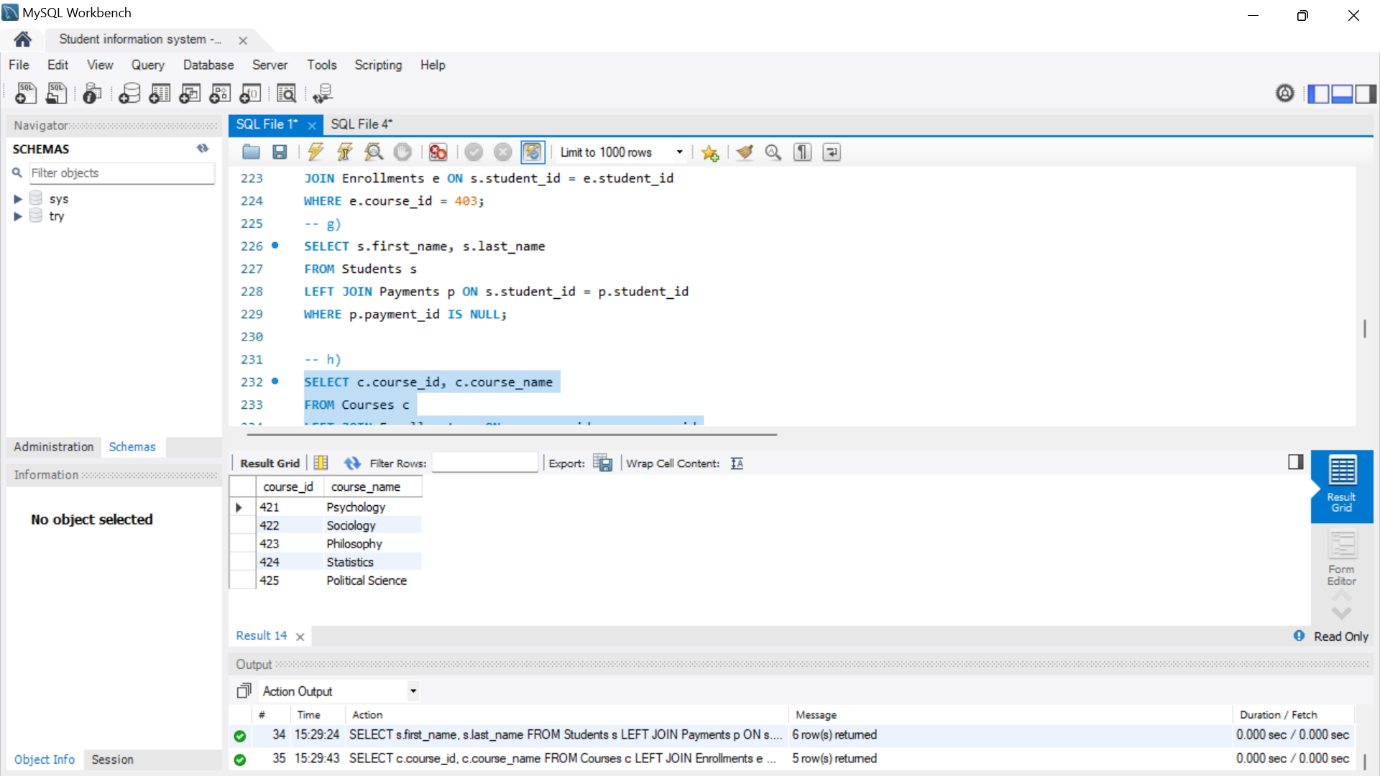
**8. Write a query to identify courses that have no enrollments. You'll need to use a LEFT JOIN between the "Courses" table and the "Enrollments" table and filter for courses with NULL enrollment records.**

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.**

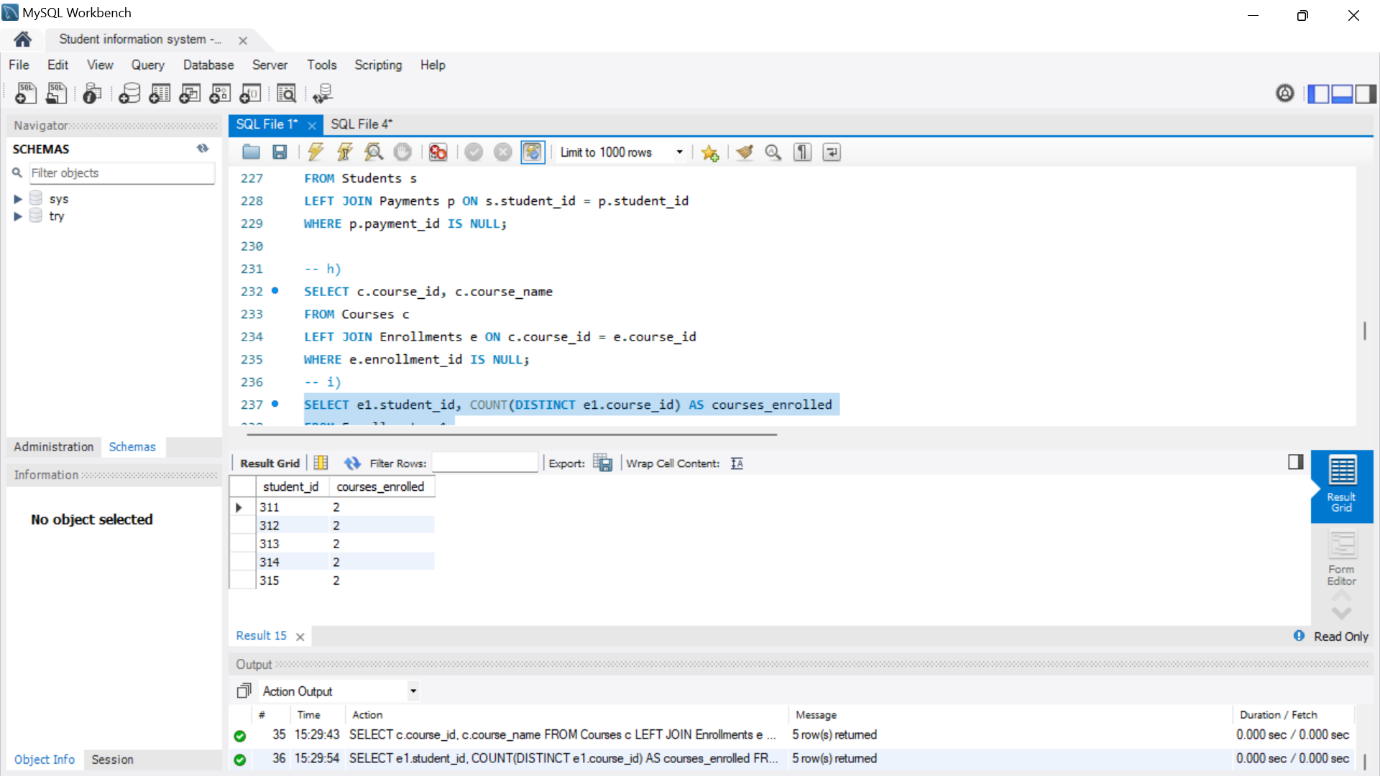
SELECT e1.student\_id, COUNT(DISTINCT e1.course\_id) AS courses\_enrolled

FROM Enrollments e1

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

GROUP BY e1.student\_id

HAVING COUNT(DISTINCT e1.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.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;



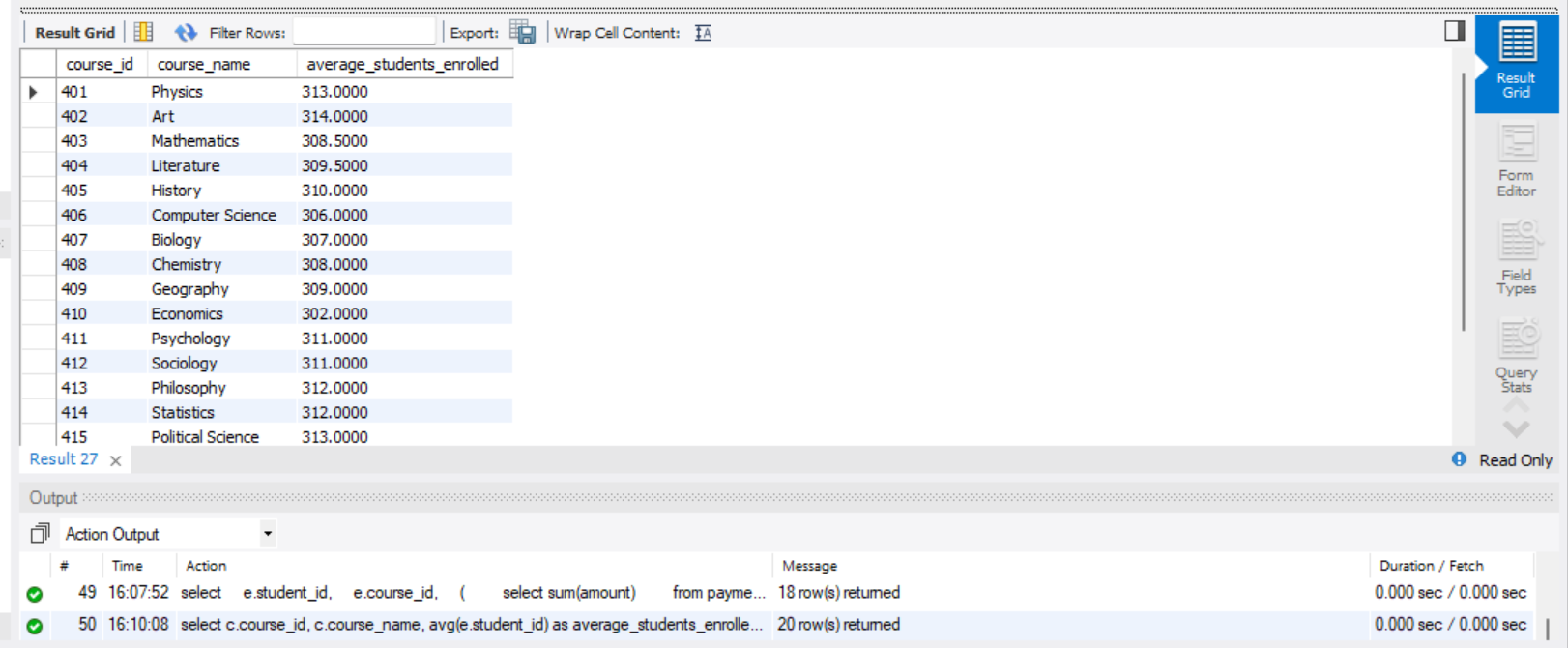
**TASK5-**

**-- 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(e.student\_id) as average\_students\_enrolled

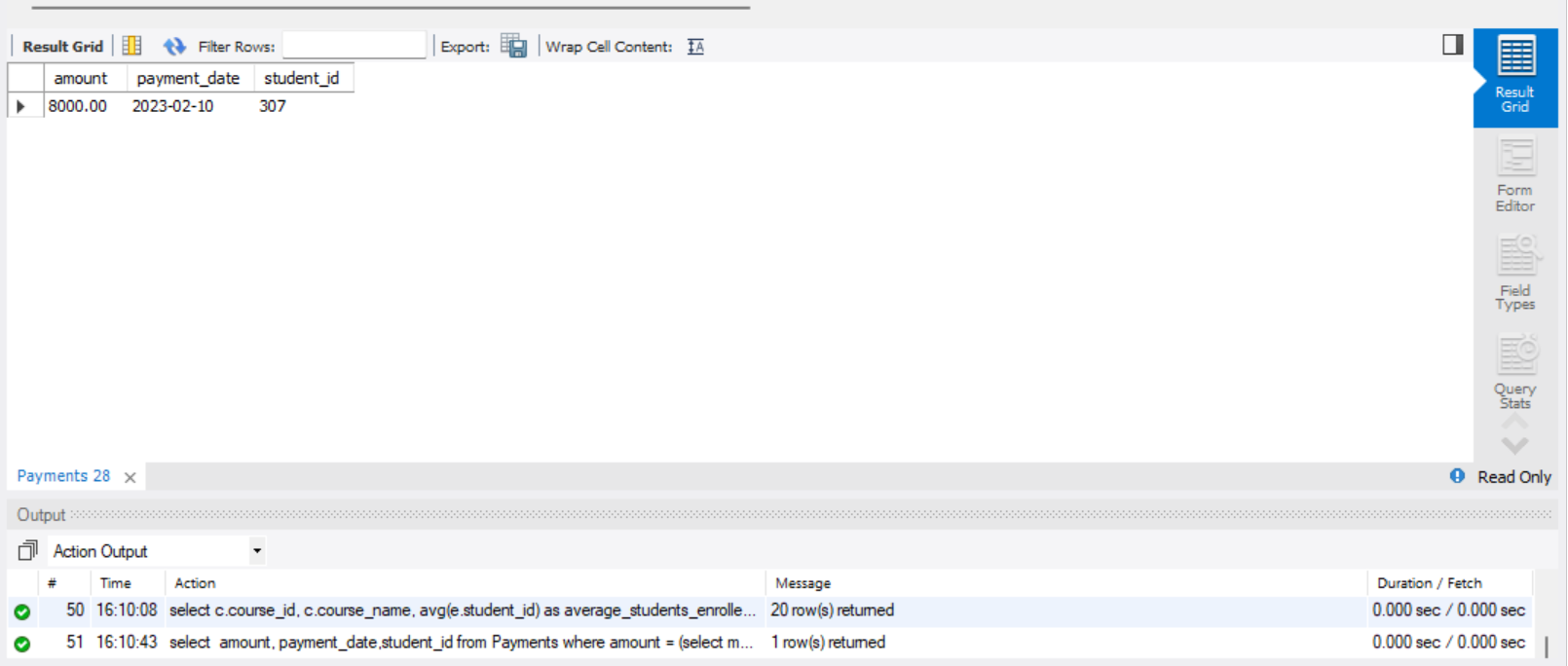
from Courses c left join Enrollments e on c.course\_id = e.course\_id group by c.course\_id, c.course\_name;



**-- 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 amount, payment\_date,student\_id from Payments where amount = (select max(amount) from Payments);

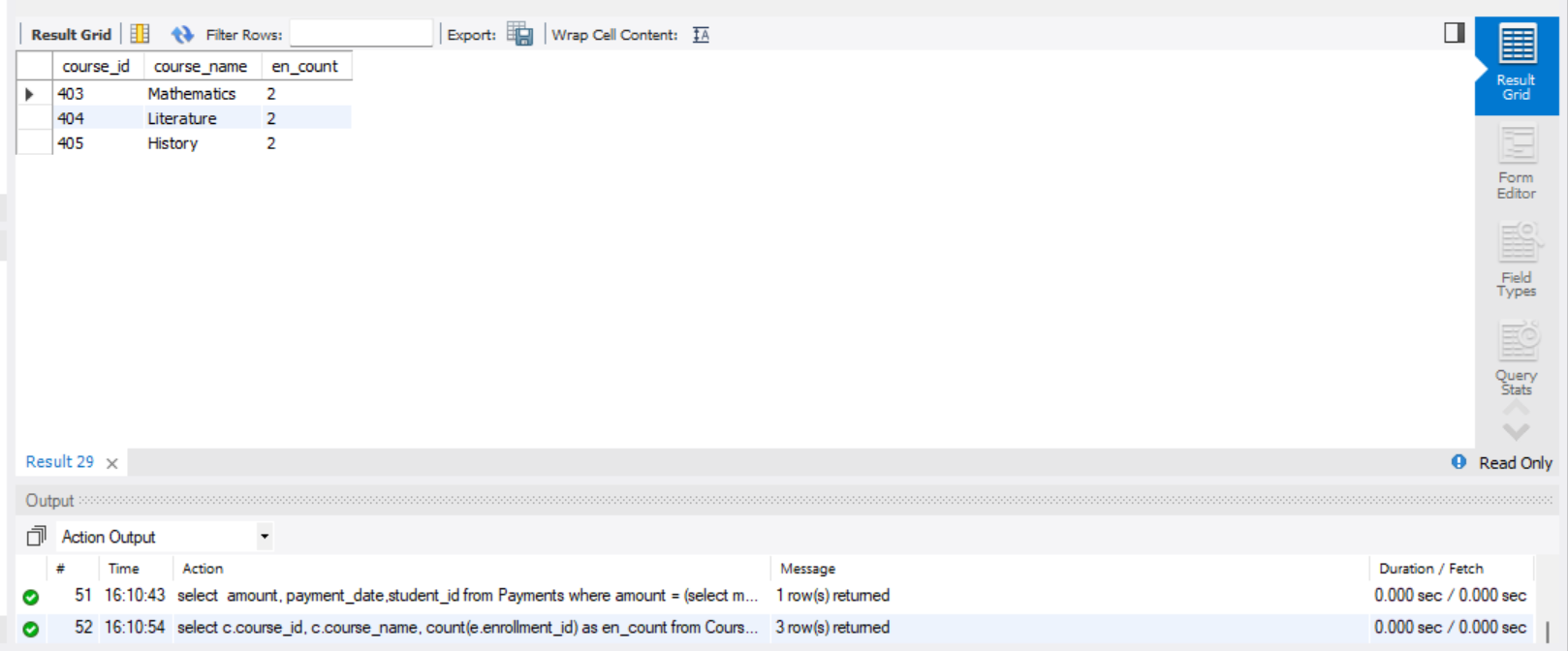


**-- 3 Retrieve a list of courses with the highest number of enrollments.**

**-- Use subqueries to find the course(s) with the maximum enrollment count.**

select c.course\_id, c.course\_name, count(e.enrollment\_id) as en\_count

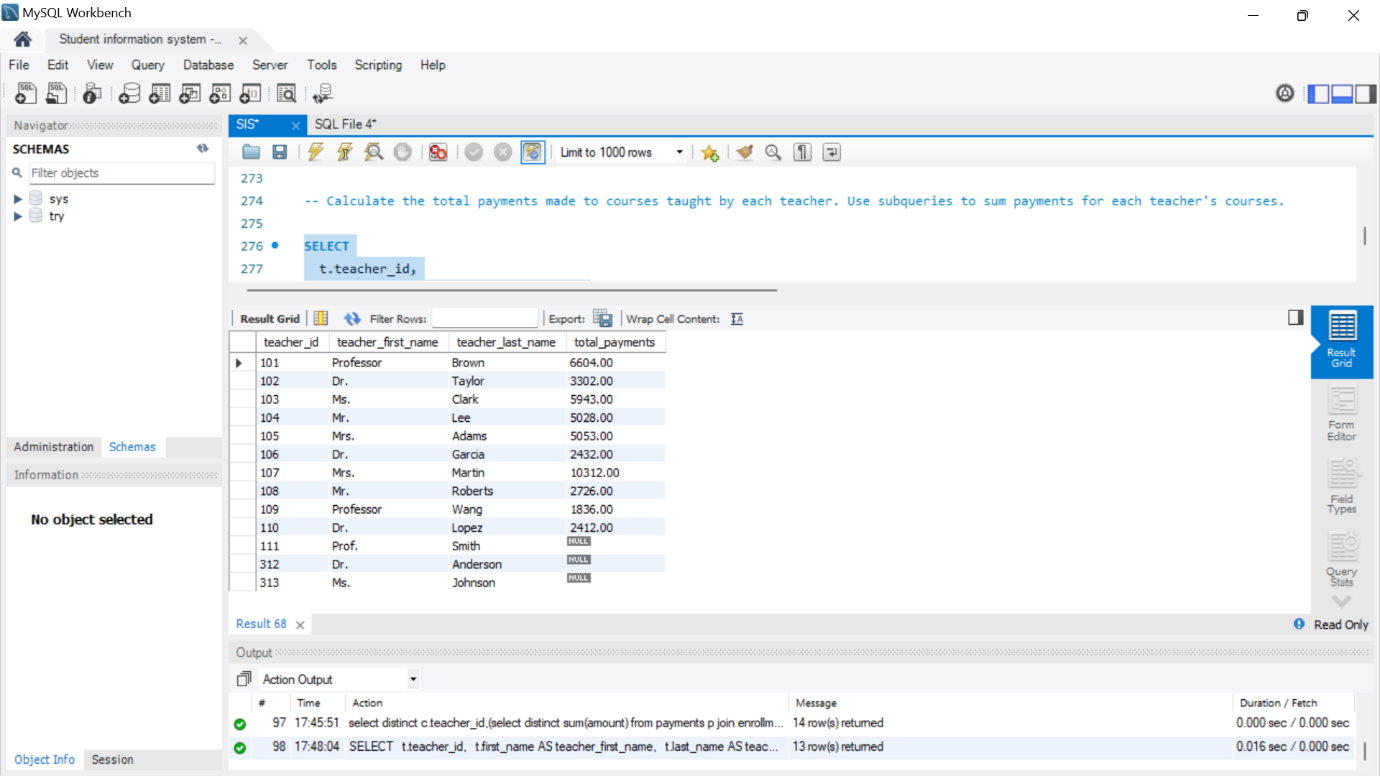
from Courses c left join Enrollments e on c.course\_id = e.course\_id group by c.course\_id, c.course\_name having en\_count = (select max(en\_count) from (select count(enrollment\_id) as en\_count from Enrollments group by course\_id) as maximum\_enrolls);



**/\* 4. Calculate the total payments made to courses taught by each teacher. Use subqueries to sum**

**payments for each teacher's courses. \*/**

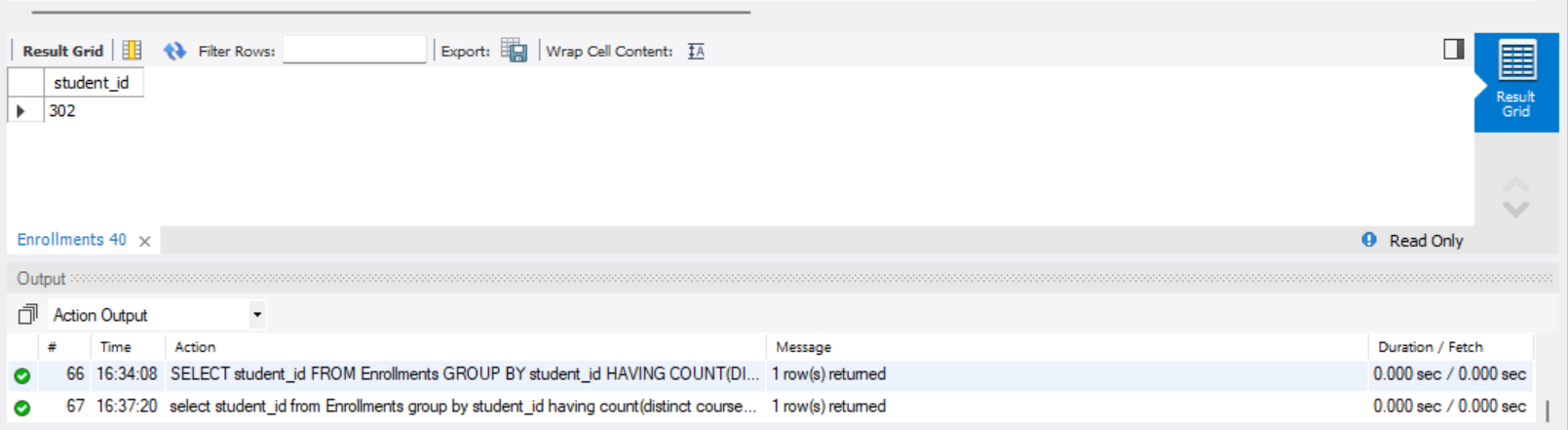
select c.teacher\_id,(select sum(amount) from payments p join enrollments e on p.student\_id = e.student\_id where e.course\_id = c.course\_id) as total\_payments from courses c;



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

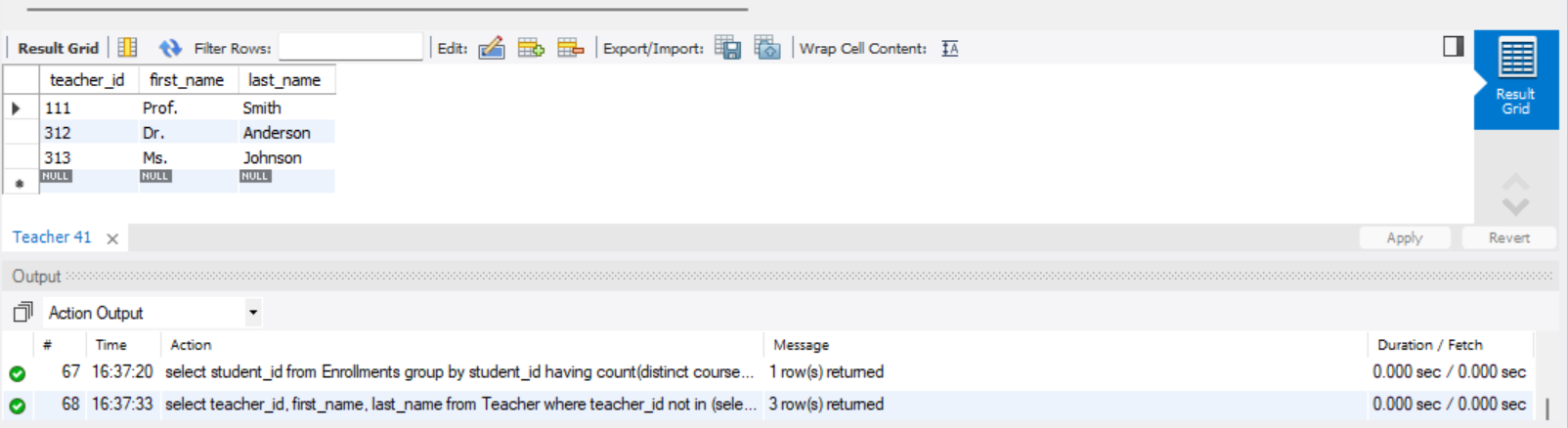
**student's enrollments with the total number of courses. \*/**

select student\_id from Enrollments group by 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, first\_name, last\_name from Teacher where teacher\_id not in (select distinct teacher\_id from Courses);

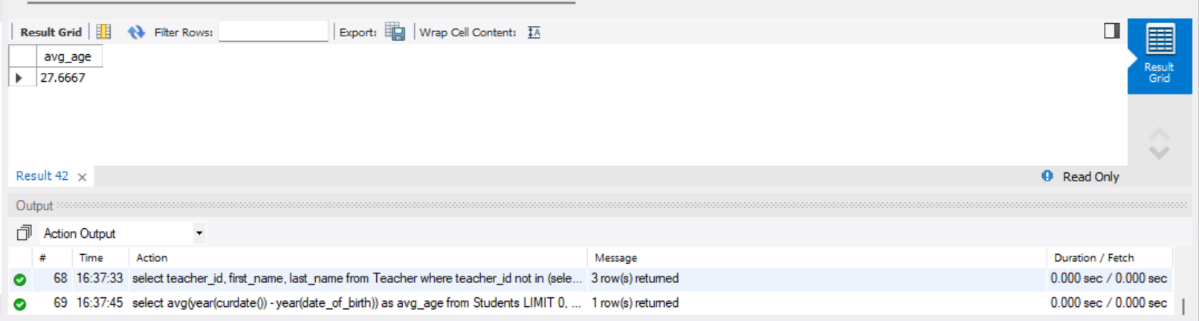


**/\* 7. Calculate the average age of all students. Use subqueries to calculate the age of each student**

**based on their date of birth. \*/**

select avg(student\_age) as average\_age

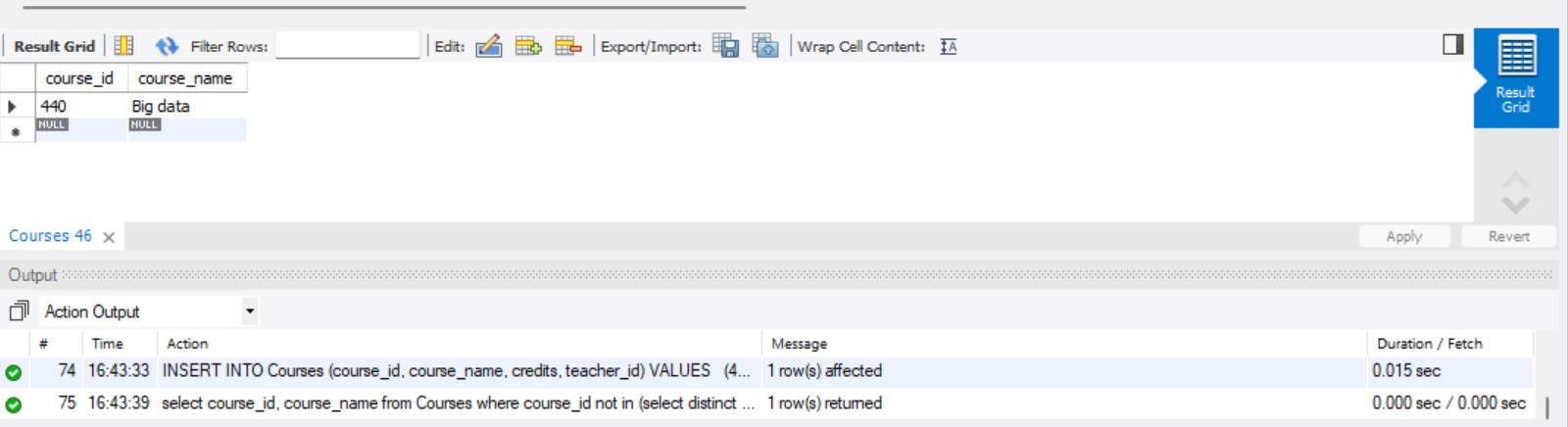
from (select datediff(curdate(), date\_of\_birth) / 365 as student\_age from Students) as avg\_ages;



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

**records. \*/**

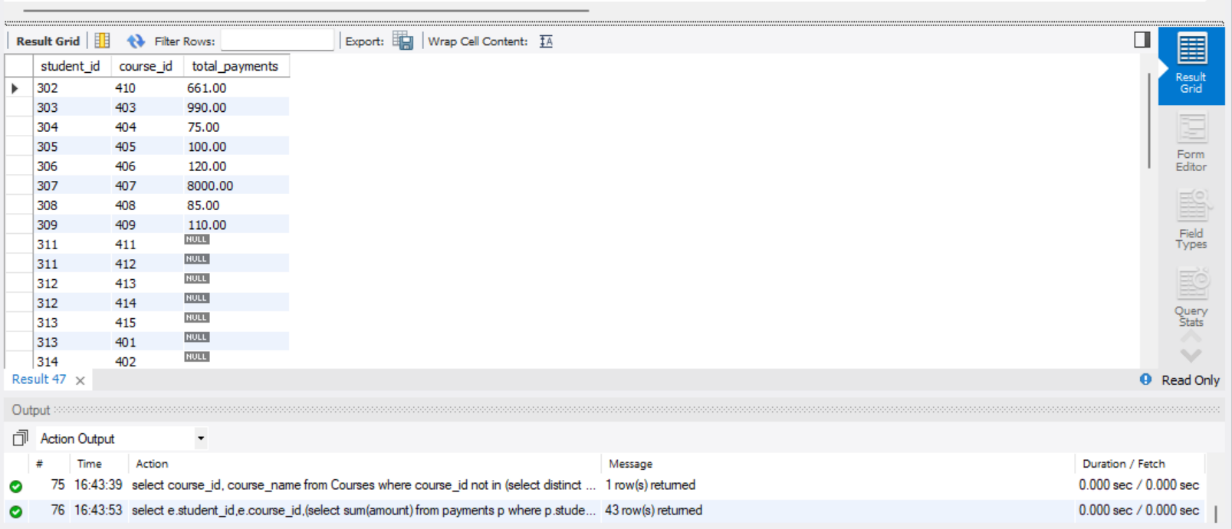
select course\_id, course\_name from Courses where course\_id not in (select distinct course\_id from Enrollments);



**-- 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,e.course\_id,(select sum(amount) from payments p where p.student\_id = e.student\_id) as total\_payments from enrollments e;



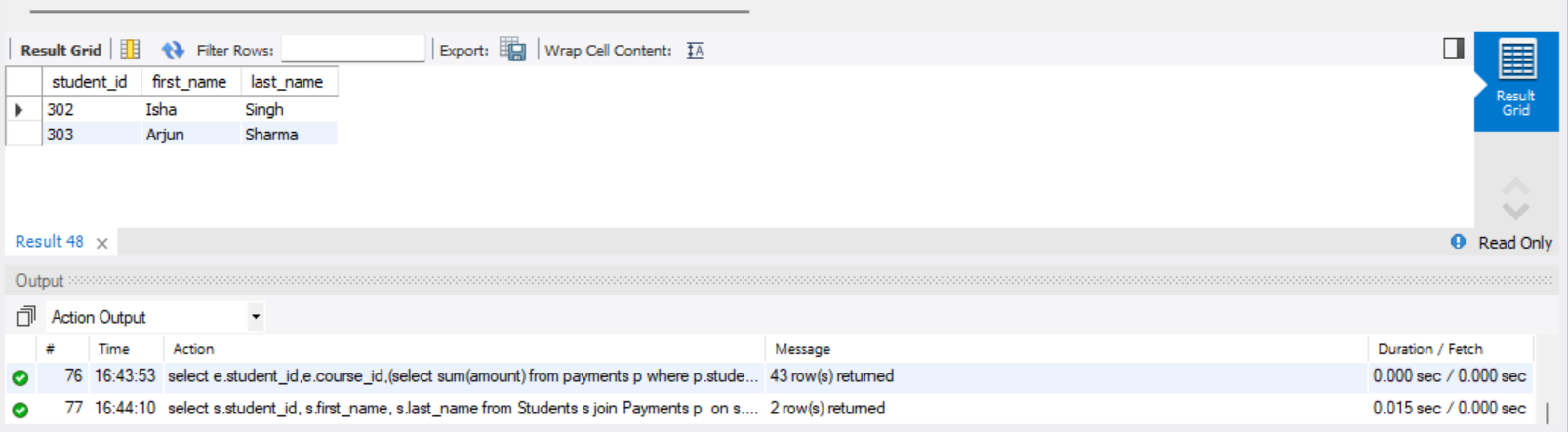
**-- 10. Identify students who have made more than one payment. Use subqueries and aggregate**

**-- functions to count payments per student and filter for those with counts greater than one.**

select s.student\_id, s.first\_name, s.last\_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;

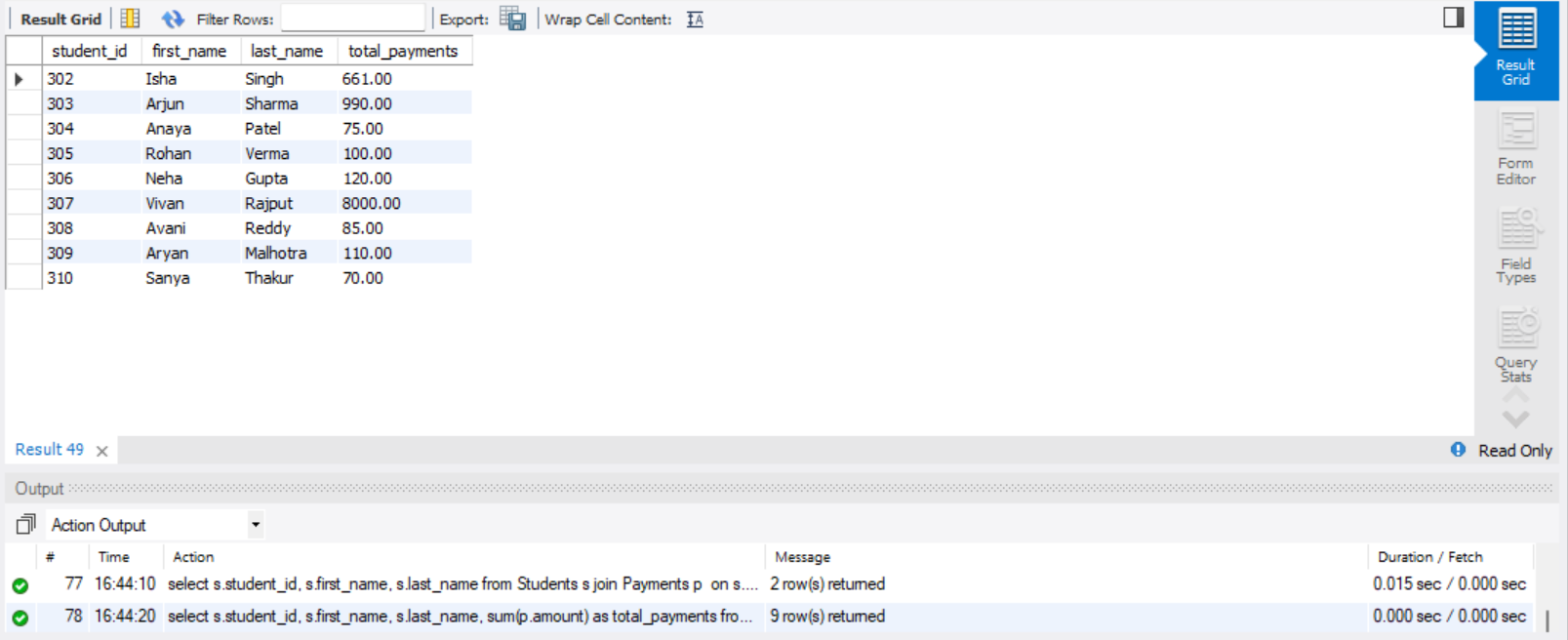
**-- 11. Write an SQL query to calculate the total payments made by each student. Join the "Students"**

**-- table with the "Payments" table and use GROUP BY to calculate the sum of payments for each**

**-- student.**

select s.student\_id, 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 group by s.student\_id;



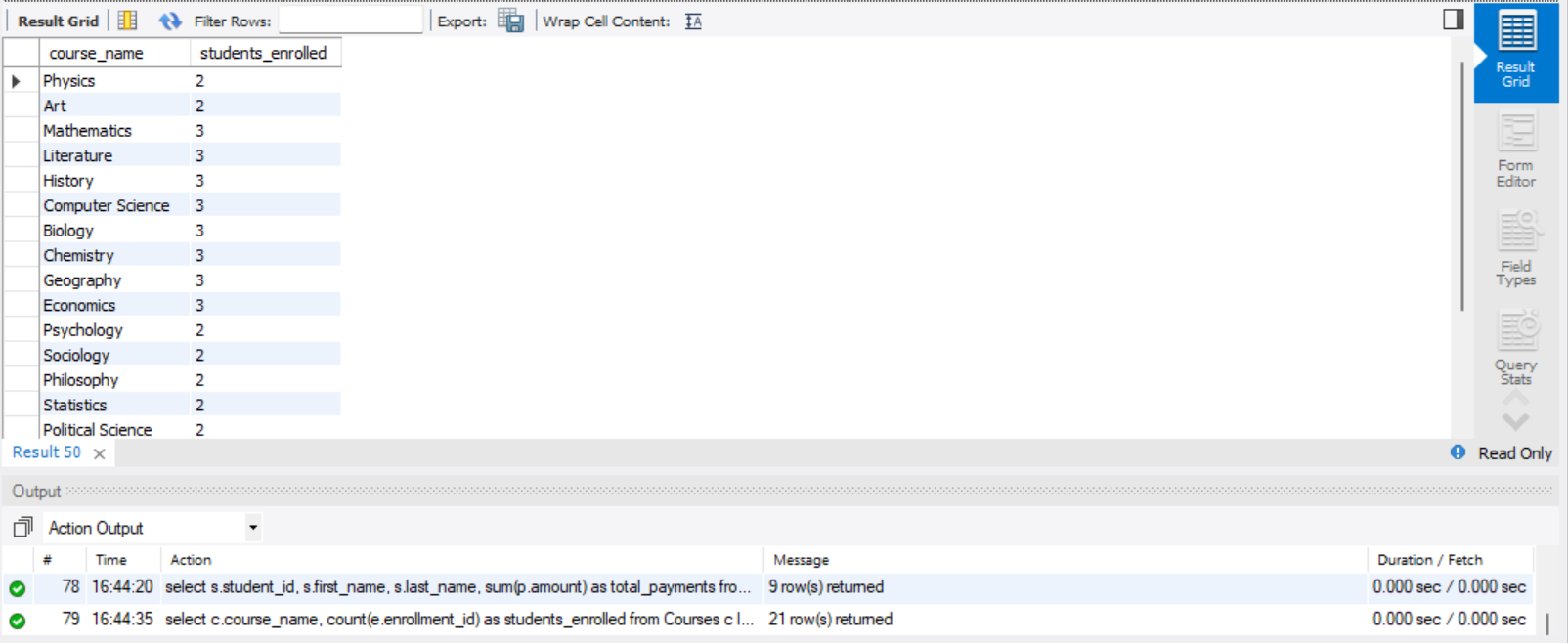
**-- 12. Retrieve a list of course names along with the count of students enrolled in each course. Use**

**-- JOIN operations between the "Courses" table and the "Enrollments" table and GROUP BY to**

**-- count enrollments.**

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, s.first\_name, s.last\_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;

