PYTHON -HEXAWARE FOUNDATION TRAINING

ASSIGNMENT

MYSQL

STUDENT INFORMATION SYSTEM

Name: Mathew P

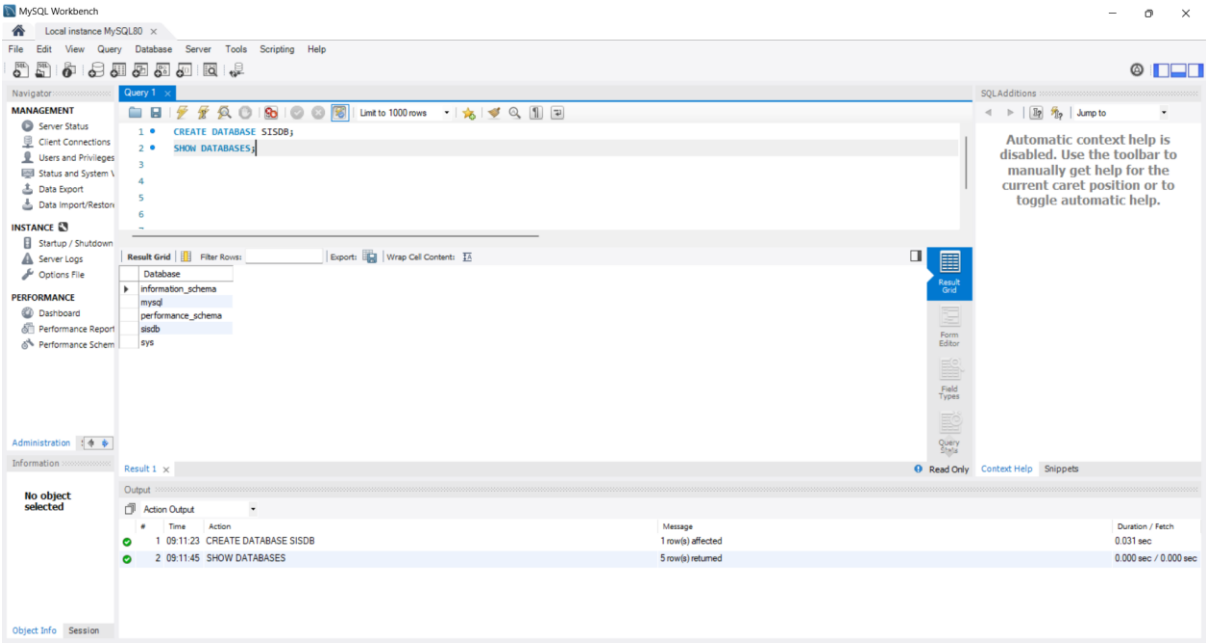
Batch: 04

TASK -1

DATABASE DESIGN

1.Create the database named "SISDB"

Query: CREATE DATABASE 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

Query:

USE SISDB;

CREATE TABLE Teacher(

teacher_id VARCHAR(10),

first_name VARCHAR(255) NOT NULL,

last_name VARCHAR(255),

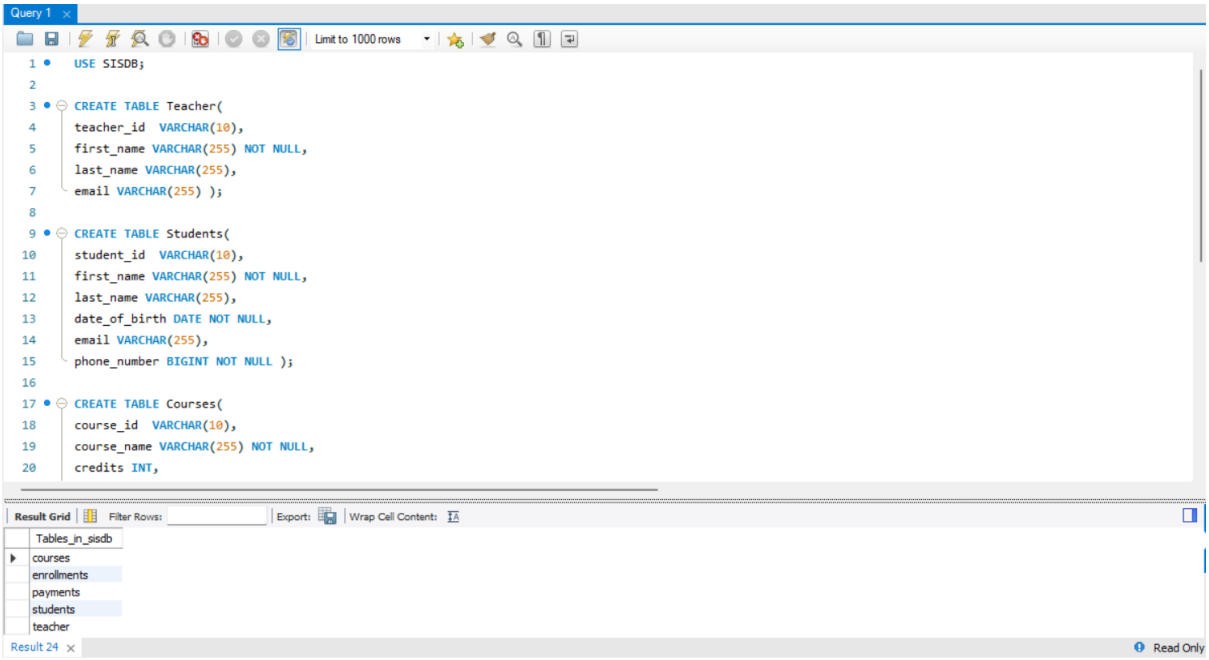
email VARCHAR(255));

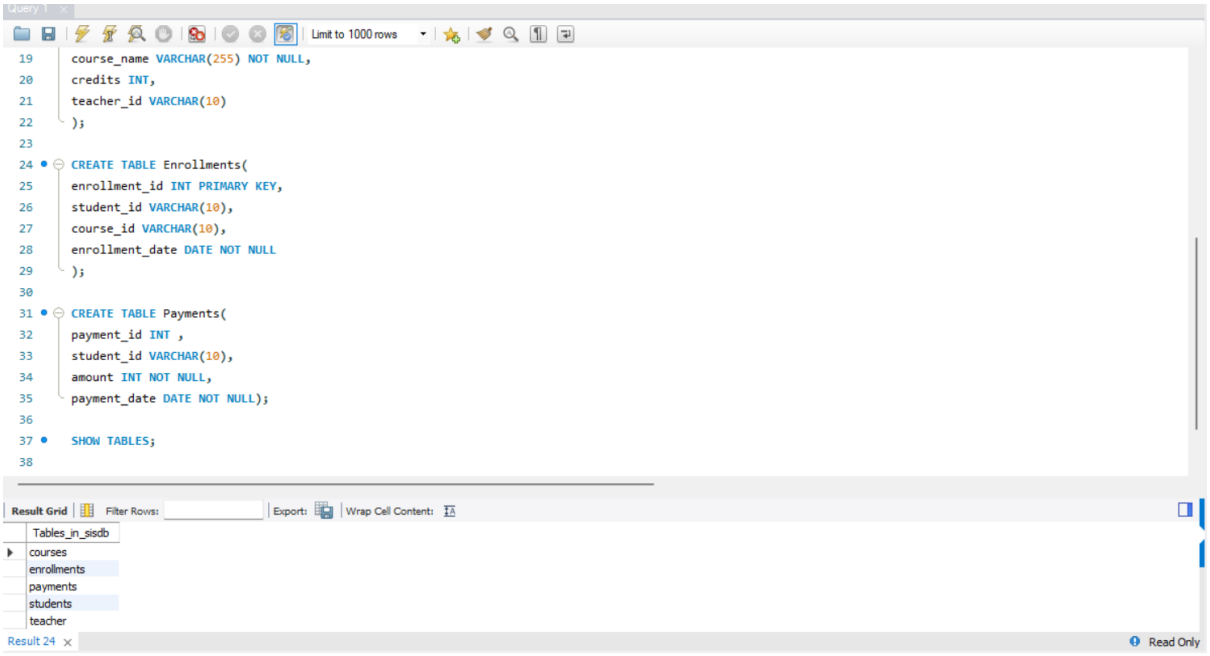
```
CREATE TABLE Students(
student_id  VARCHAR(10),
first_name VARCHAR(255) NOT NULL,
last_name VARCHAR(255),
date_of_birth DATE NOT NULL,
email VARCHAR(255),
phone_number BIGINT NOT NULL );

CREATE TABLE Courses(
course_id  VARCHAR(10),
course_name VARCHAR(255) NOT NULL,
credits INT,
teacher_id VARCHAR(10)
);

CREATE TABLE Enrollments(
enrollment_id INT PRIMARY KEY,
student_id VARCHAR(10),
course_id VARCHAR(10),
enrollment_date DATE NOT NULL
);

CREATE TABLE Payments(
payment_id INT ,
student_id VARCHAR(10),
```

amount INT NOT NULL,

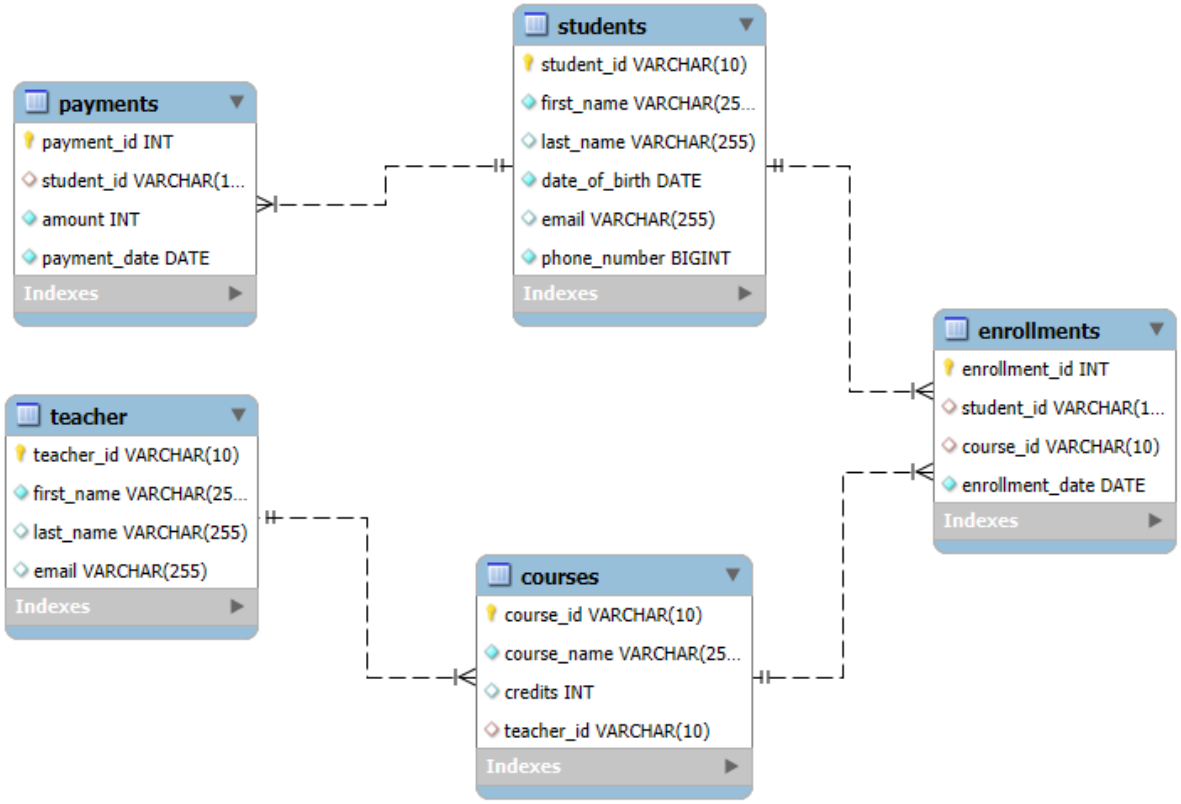
payment_date DATE NOT NULL);

SHOW TABLES;





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



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

Query:

Primary key constraint:

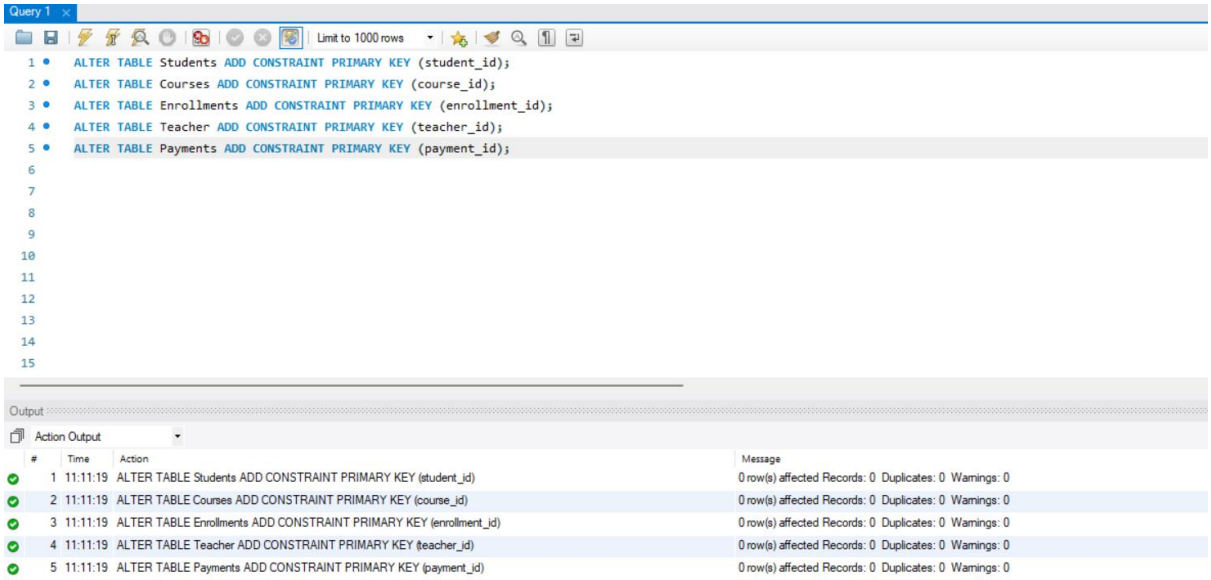
ALTER TABLE Students ADD CONSTRAINT PRIMARY KEY (student_id);

ALTER TABLE Courses ADD CONSTRAINT PRIMARY KEY (course_id);

ALTER TABLE Enrollments ADD CONSTRAINT PRIMARY KEY (enrollment_id);

ALTER TABLE Teacher ADD CONSTRAINT PRIMARY KEY (teacher_id);

ALTER TABLE Payments ADD CONSTRAINT PRIMARY KEY (payment_id);



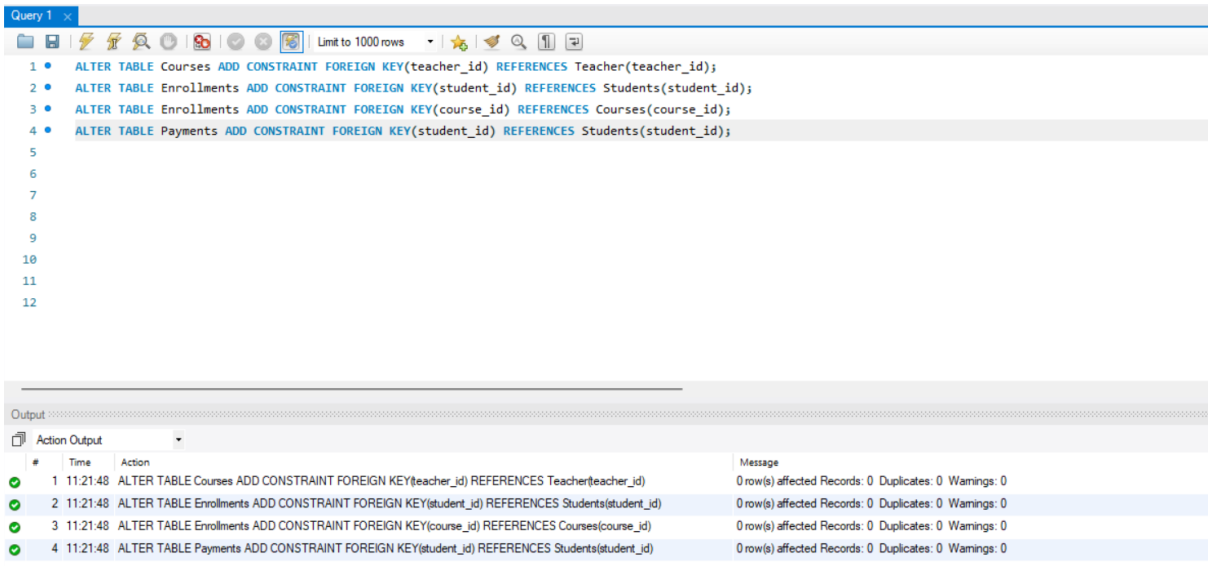
Foreign Key Constraint:

ALTER TABLE Courses ADD CONSTRAINT FOREIGN KEY(teacher_id) REFERENCES Teacher(teacher_id);

ALTER TABLE Enrollments ADD CONSTRAINT FOREIGN KEY(student_id) REFERENCES Students(student_id);

ALTER TABLE Enrollments ADD CONSTRAINT FOREIGN KEY(course_id) REFERENCES Courses(course_id);

ALTER TABLE Payments ADD CONSTRAINT FOREIGN KEY(student_id) REFERENCES Students(student_id);



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

Query:

5.i. Students Table:

INSERT INTO Students(student_id, first_name, last_name, date_of_birth, email, phone_number) VALUES(1, 'Angella', 'Thomson', '1995-08-12', 'angella@example.com', '9078563412'),

(2, 'Bellatrix', 'Lestrange', '1993-10-11', 'bella@example.com', '8079685746'),

(3,'Hermoine', 'Granger', '1994-11-17', 'hermo@exampl.com','3524138079'),

(4,'Harry','Potter','1994-10-12','potter@example.com','0987654321'),

(5,'Ron','Weasly','1994-05-15','ron@example.com','0099887766'),

(6,'Draco','Malfoy','1994-12-12','draco@example.com','9988776655'),

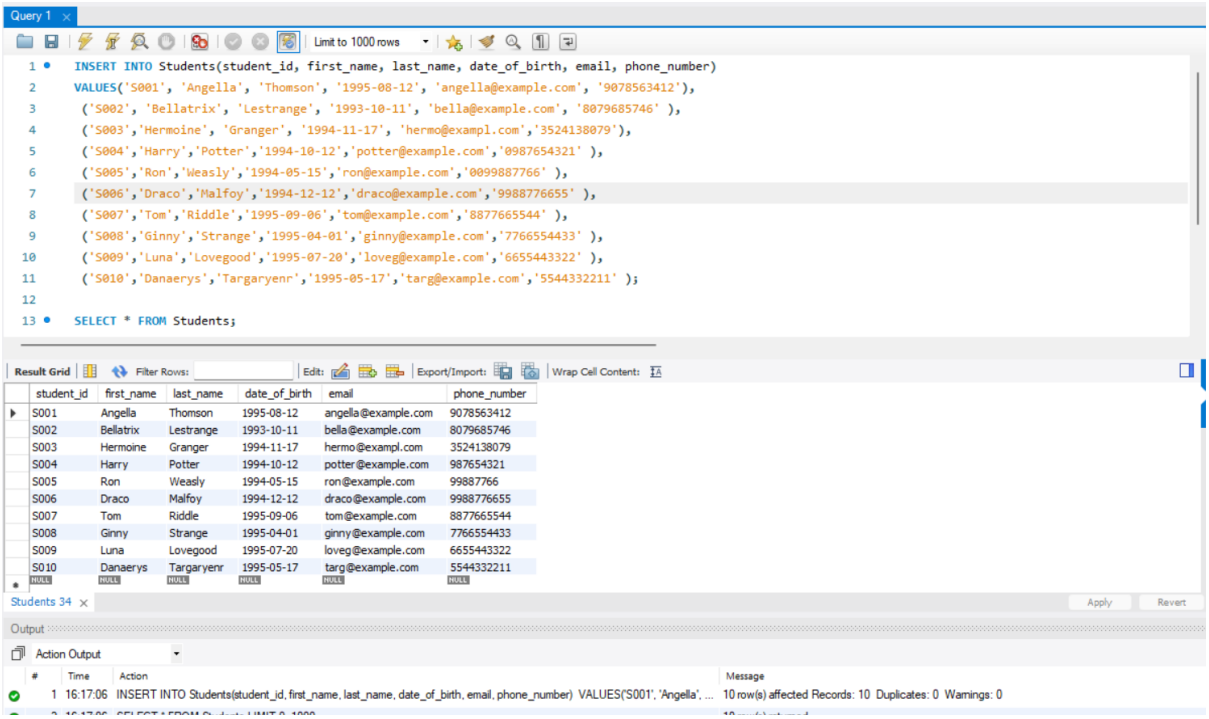
(7,'Tom','Riddle','1995-09-06','tom@example.com','8877665544'),

(8,'Ginny','Strange','1995-04-01','potter@example.com','7766554433'),

(9,'Luna','Lovegood','1995-07-20','potter@example.com','6655443322'),

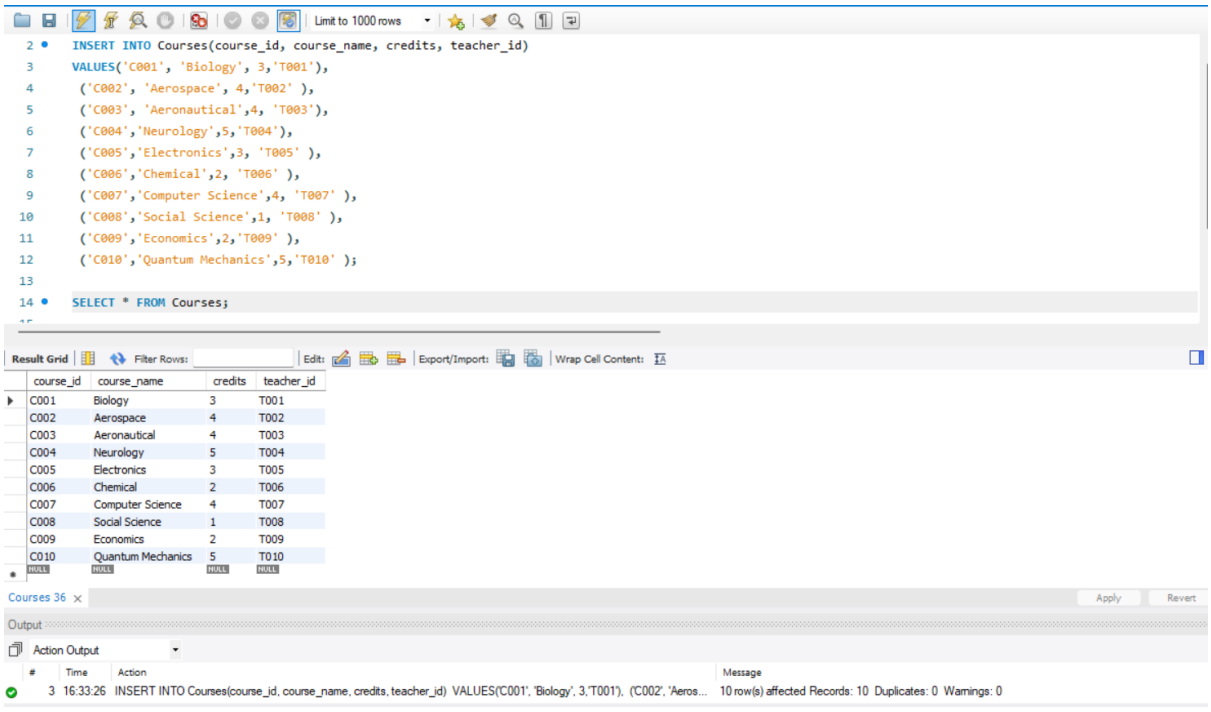
(10,'Danaerys','Targaryenr','1995-05-17','potter@example.com','5544332211');

SELECT * FROM Students;



5.ii. Courses Table:

```
INSERT INTO Courses(course_id, course_name, credits, teacher_id)
VALUES('C001', 'Biology', 3,'T001'),
 ('C002', 'Aerospace', 4,'T002' ),
 ('C003', 'Aeronautical',4, 'T003'),
 ('C004','Neurology',5,'T004'),
 ('C005','Electronics',3, 'T005' ),
 ('C006','Chemical',2, 'T006' ),
 ('C007','Computer Science',4, 'T007' ),
 ('C008','Social Science',1, 'T008' ),
 ('C009','Economics',2,'T009' ),
 ('C010','Quantum Mechanics',5,'T010' );

 SELECT * FROM Courses;
```



5.iii.Enrollments Table:

INSERT INTO Enrollments (enrollment_id, student_id, course_id, enrollment_date) VALUES

(1, 'S001', 'C001', '2024-06-01'),

(2, 'S002', 'C002', '2024-06-02'),

(3, 'S003', 'C003', '2024-06-03'),

(4, 'S004', 'C004', '2024-06-04'),

(5, 'S005', 'C005', '2024-06-05'),

(6, 'S006', 'C006', '2024-06-06'),

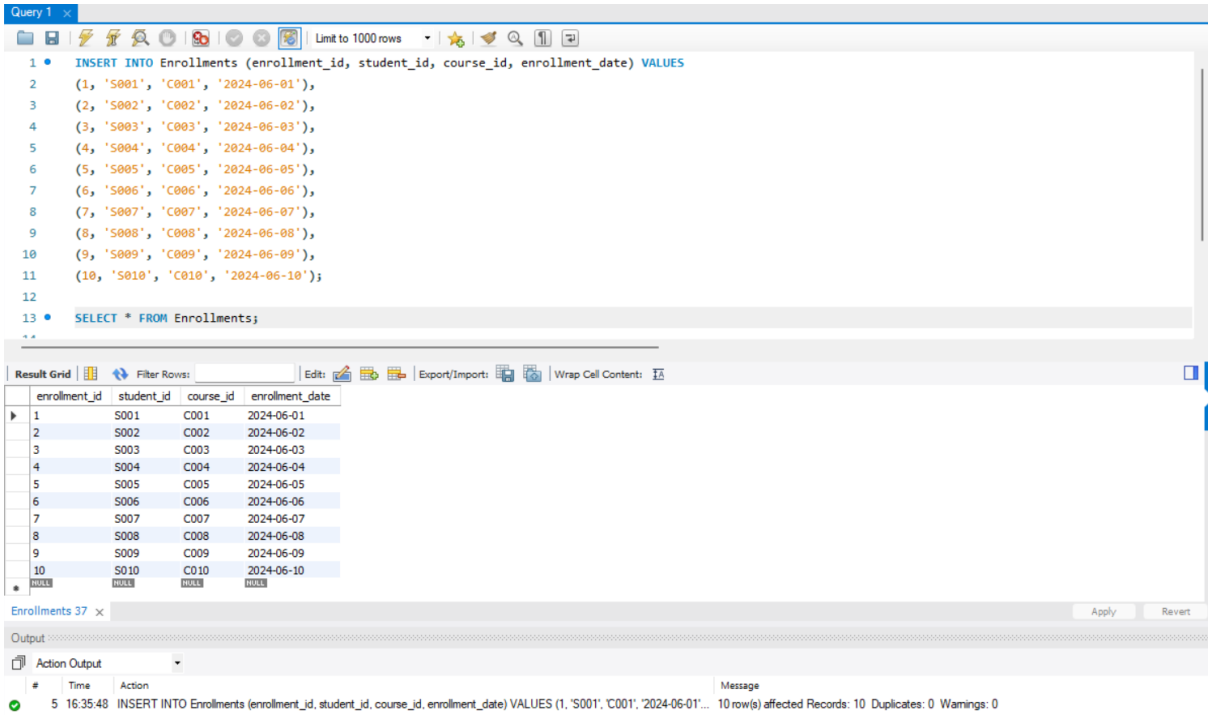
(7, 'S007', 'C007', '2024-06-07'),

(8, 'S008', 'C008', '2024-06-08'),

(9, 'S009', 'C009', '2024-06-09'),

(10, 'S010', 'C010', '2024-06-10');

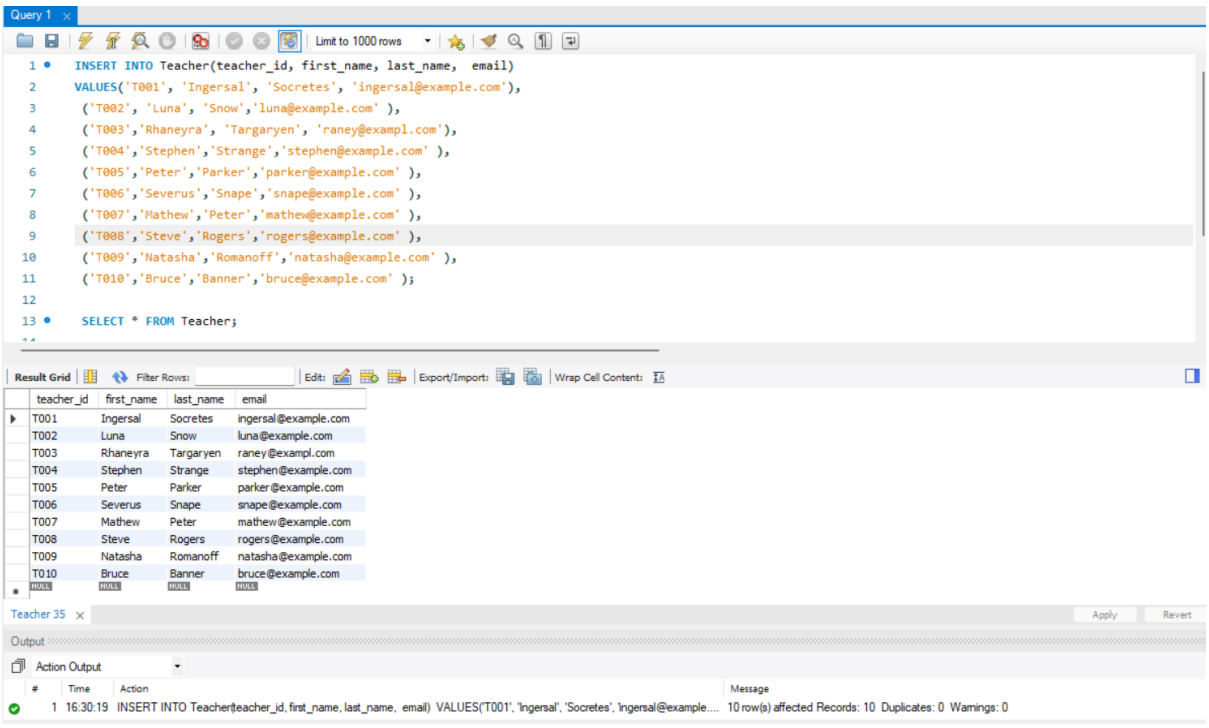
SELECT*FROM Enrollments;



5.iv. Teacher table:

```
INSERT INTO Teacher(teacher_id, first_name, last_name,  email)
VALUES('T001', 'Ingersal', 'Socretes', 'ingersal@example.com'),
 ('T002', 'Luna', 'Snow','luna@example.com' ),
 ('T003','Rhaneyra', 'Targaryen', 'raney@exampl.com'),
 ('T004','Stephen','Strange','stephen@example.com' ),
 ('T005','Peter','Parker','parker@example.com' ),
 ('T006','Severus','Snape','snape@example.com' ),
 ('T007','Mathew','Peter','mathew@example.com' ),
 ('T008','Steve','Rogers','rogers@example.com' ),
 ('T009','Natasha','Romanoff','natasha@example.com' ),
 ('T010','Bruce','Banner','bruce@example.com' );
```

```
SELECT * FROM Teacher;
```



5.v. Payments Table:

INSERT INTO Payments (payment_id, student_id, amount, payment_date) VALUES

(1, 'S001', 1000, '2024-06-01'),

(2, 'S002', 2000, '2024-06-02'),

(3, 'S003', 1500, '2024-06-03'),

(4, 'S004', 2500, '2024-06-04'),

(5, 'S005', 1800, '2024-06-05'),

(6, 'S006', 2100, '2024-06-06'),

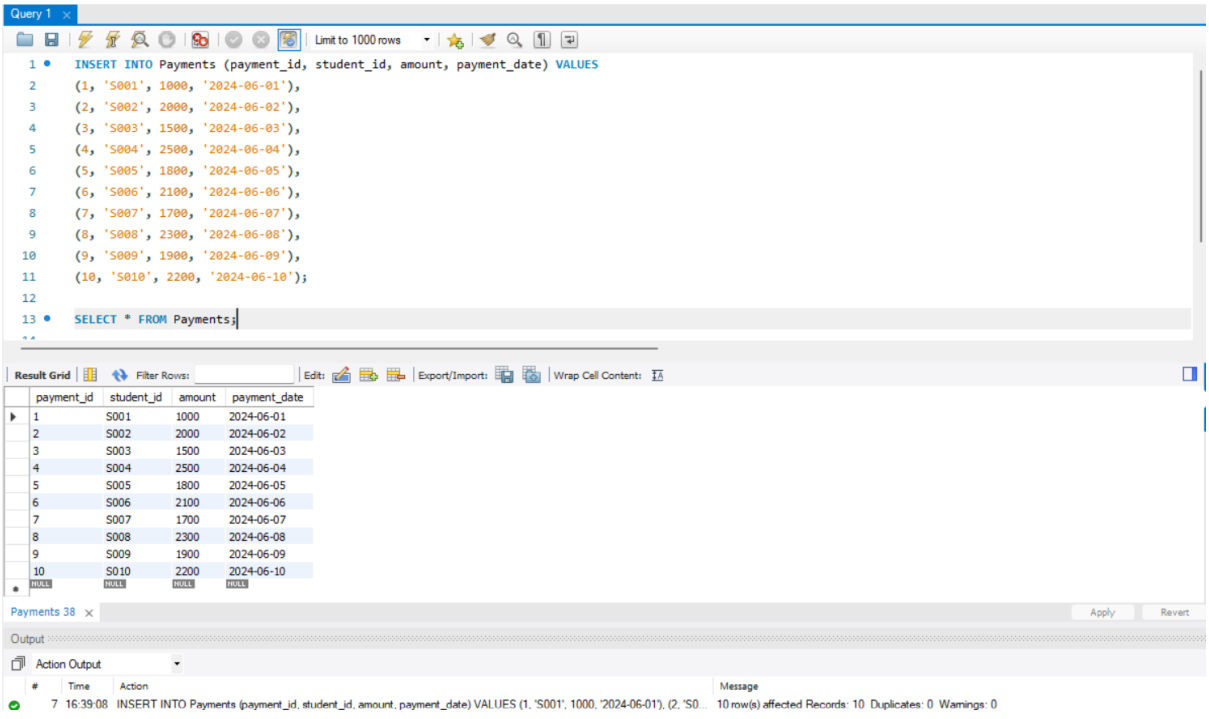
(7, 'S007', 1700, '2024-06-07'),

(8, 'S008', 2300, '2024-06-08'),

(9, 'S009', 1900, '2024-06-09'),

(10, 'S010', 2200, '2024-06-10');

SELECT * FROM Payments;



TASK-02

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

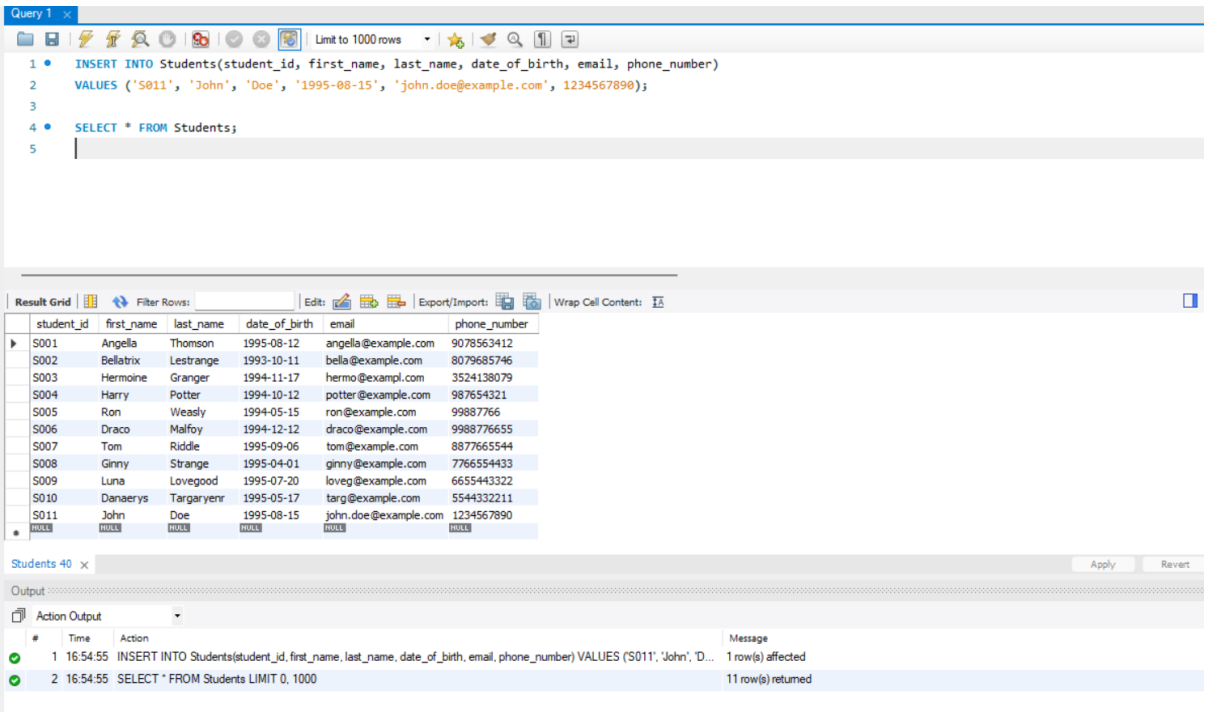
d. Email: john.doe@example.com

e. Phone Number: 1234567890

Query:

INSERT INTO Students(student_id, first_name, last_name, date_of_birth, email, phone_number) VALUES ('S011', 'John', 'Doe', '1995-08-15', 'john.doe@example.com', 1234567890);

SELECT * FROM Students;



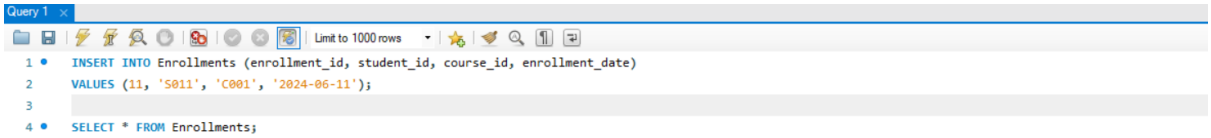
2. Write an SQL query to enroll a student in a course. Choose an existing student and course and insert a record into the "Enrollments" table with the enrollment date.

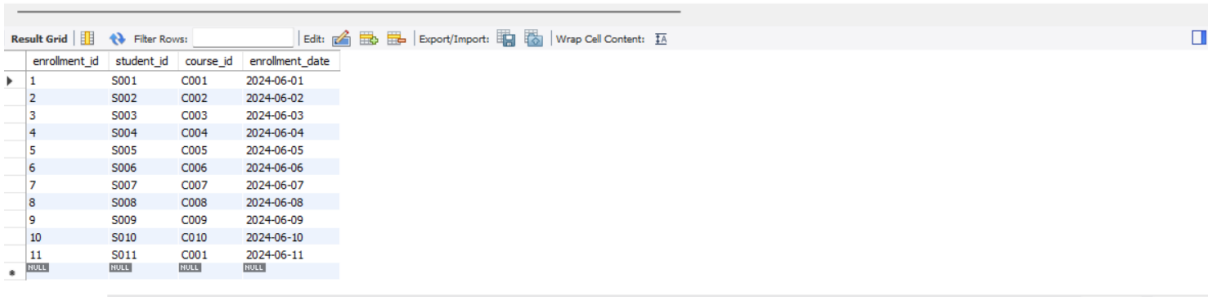
Query:

INSERT INTO Enrollments (enrollment_id, student_id, course_id, enrollment_date)

VALUES (11, 'S011', 'C001', '2024-06-11');

SELECT * FROM Enrollments;





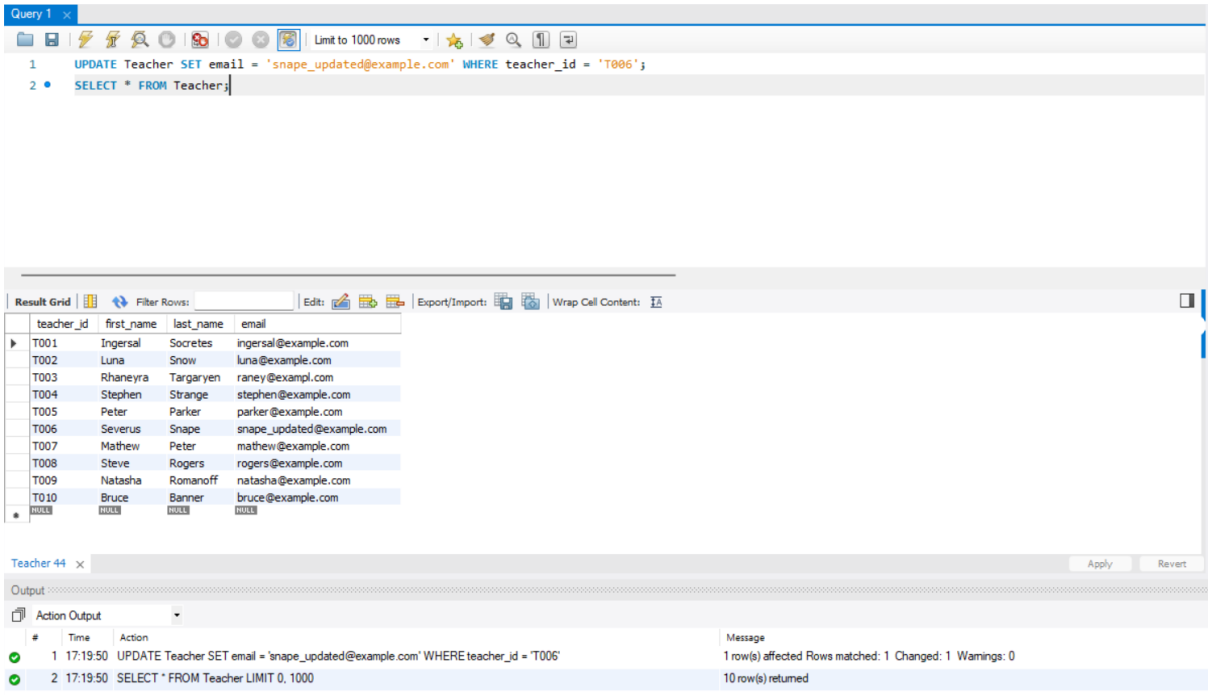
enrollment_id	student_id	course_id	enrollment_date
1	S001	C001	2024-06-01
2	S002	C002	2024-06-02
3	S003	C003	2024-06-03
4	S004	C004	2024-06-04
5	S005	C005	2024-06-05
6	S006	C006	2024-06-06
7	S007	C007	2024-06-07
8	S008	C008	2024-06-08
9	S009	C009	2024-06-09
10	S010	C010	2024-06-10
11	S011	C001	2024-06-11
NULL	NULL	NULL	NULL

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

Query:

UPDATE Teacher SET email = 'snape_updated@example.com' WHERE teacher_id = 'T006';

SELECT * FROM Teacher;

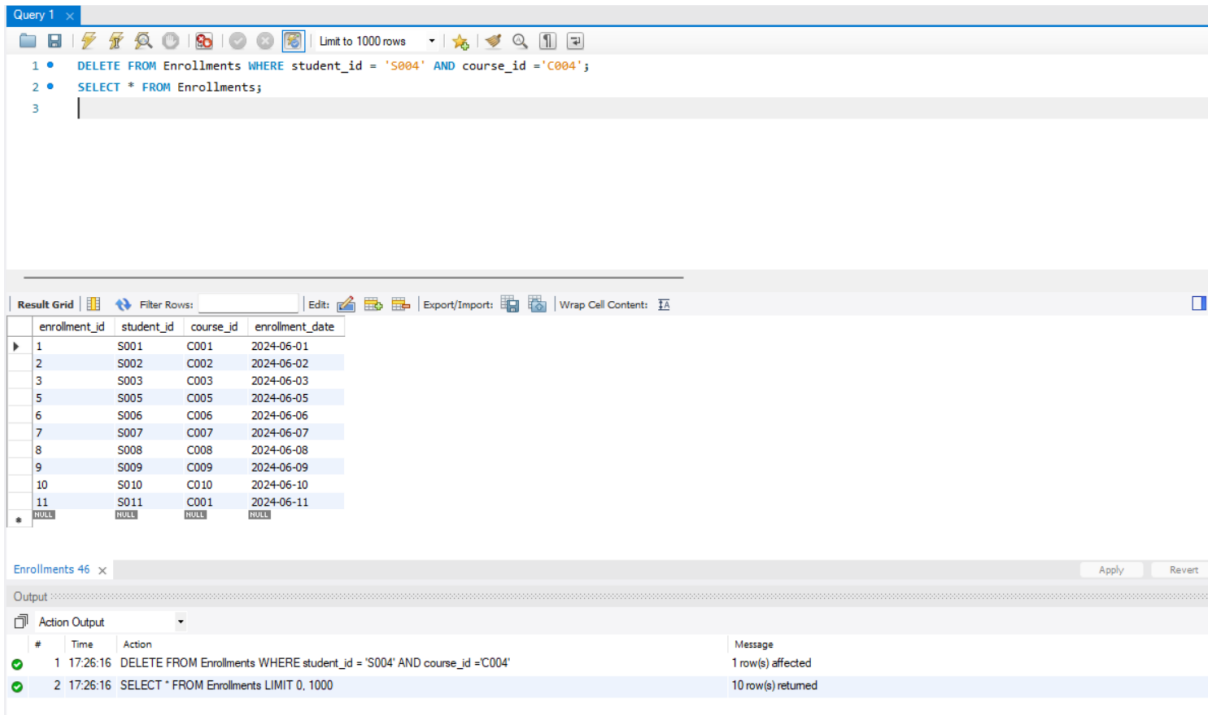


4. Write an SQL query to delete a specific enrollment record from the "Enrollments" table. Select an enrollment record based on the student and course.

Query:

DELETE FROM Enrollments WHERE student_id = 'S004' AND course_id ='C004';

SELECT * FROM Enrollments;

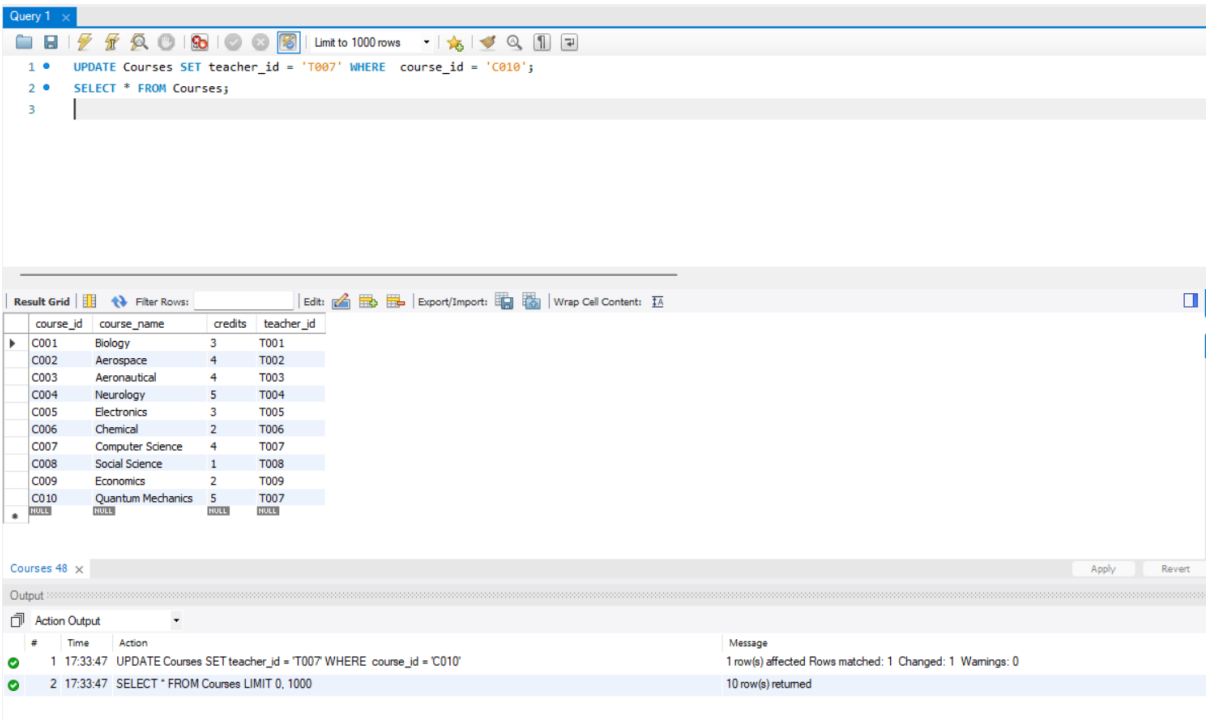


5. Update the "Courses" table to assign a specific teacher to a course. Choose any course and teacher from the respective tables.

Query:

UPDATE Courses SET teacher_id = 'T007' WHERE course_id = 'C010';

SELECT * FROM Courses;



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.

Explanation:

To delete a specific student, by considering “Referential Integrity”, we need to delete the child table data of the student and then finally delete the Parent table data of the particular student.

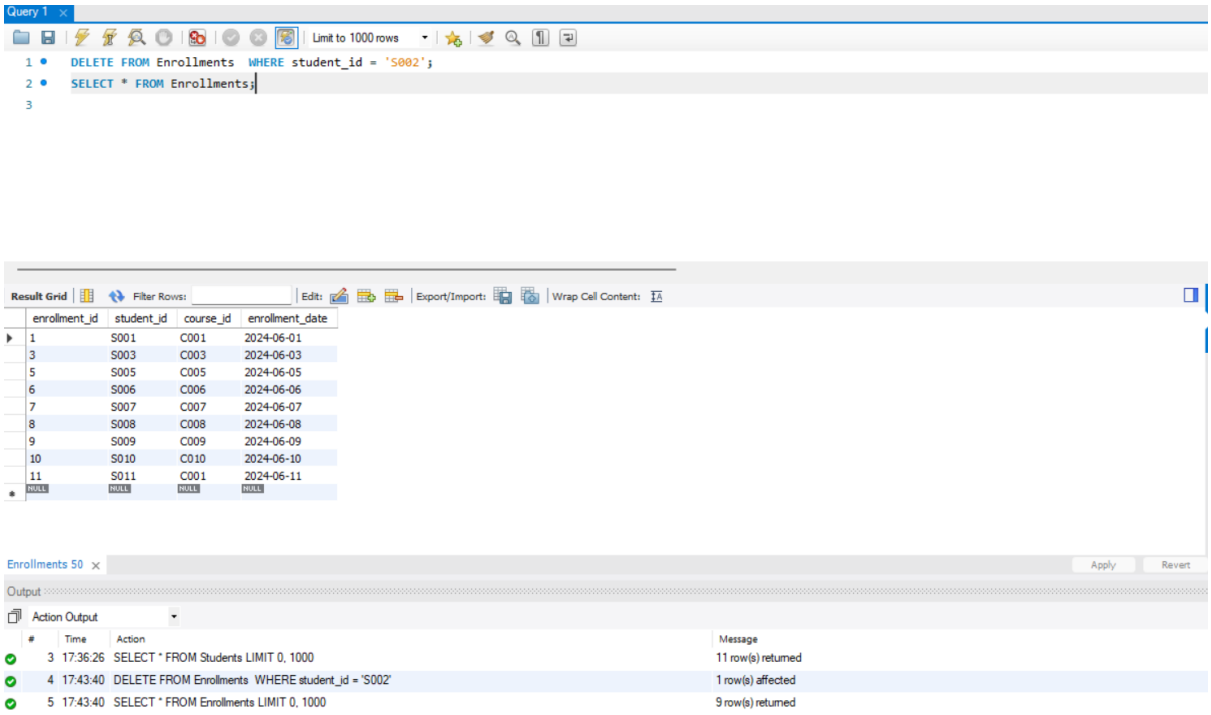
For an illustration, lets consider the deletion of S002 student “Bellatrix Lestrange”.

Step 1: we need to delete the student’s data from Enrollments table (Child table).

Query:

DELETE FROM Enrollments WHERE student_id = 'S002';

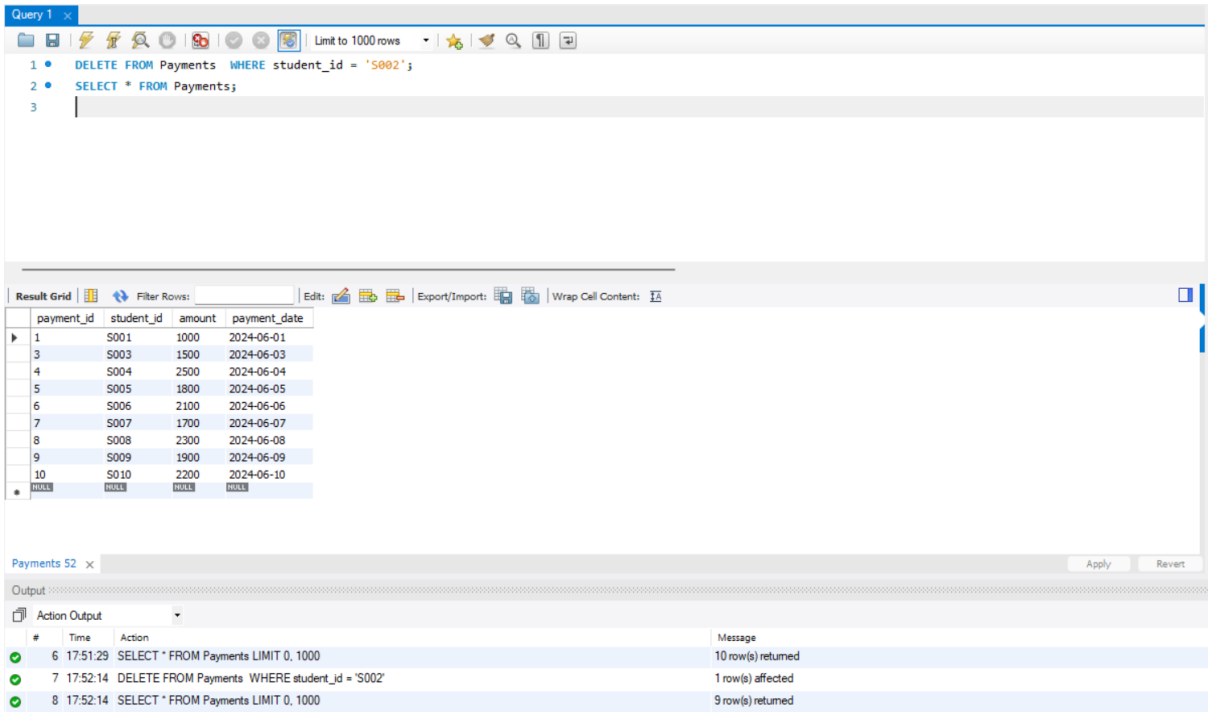
SELECT * FROM Enrollments;



Step 2: Deletion of Student ‘S002’ from the Payments Table (also a Child Table).

Query: DELETE FROM Payments WHERE student_id = 'S002';

SELECT * FROM Payments;

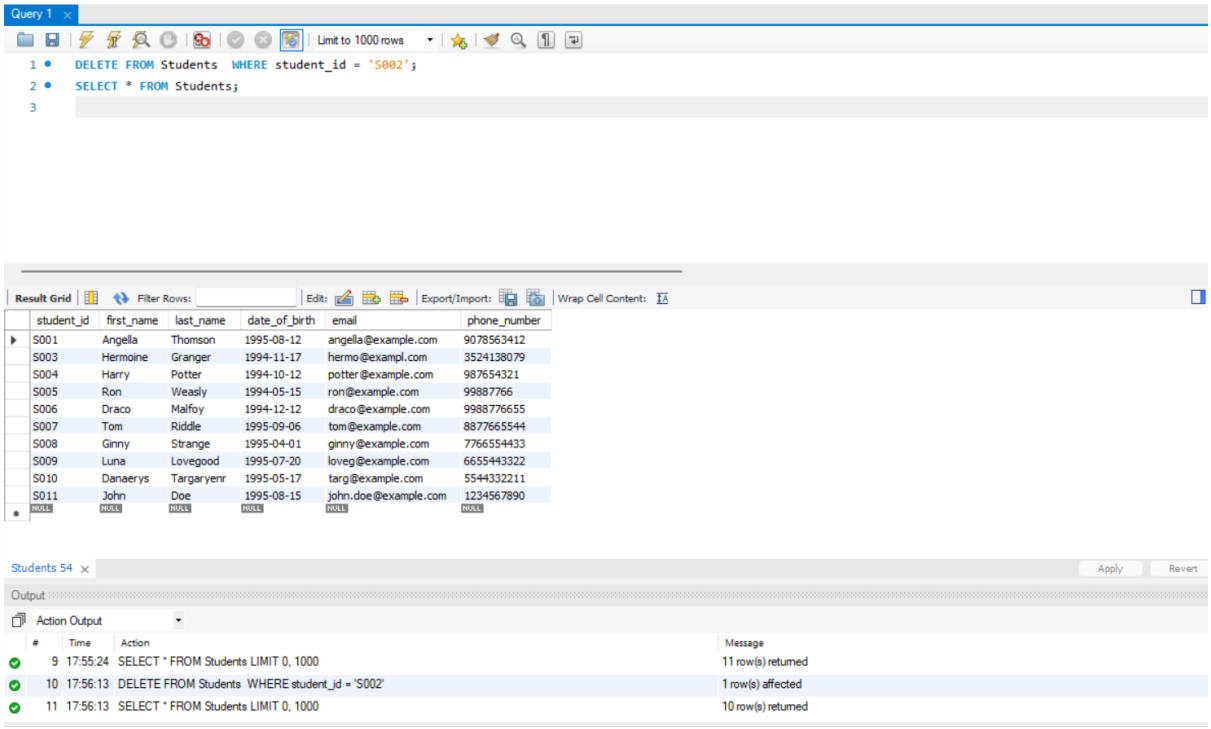


Step 3: Finally we can delete Student ‘S002’ from the parent table “Students table”

Query:

DELETE FROM Students WHERE student_id = 'S002';

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.

Query:

UPDATE Payments SET amount = 2000 WHERE payment_id = 5;

SELECT * FROM Payments;

Query 1

```
1 • UPDATE Payments SET amount = 2000 WHERE payment_id = 5;
2 • SELECT * FROM Payments;
3
```

Result Grid

payment_id	student_id	amount	payment_date
1	S001	1000	2024-06-01
3	S003	1500	2024-06-03
4	S004	2500	2024-06-04
5	S005	2000	2024-06-05
6	S006	2100	2024-06-06
7	S007	1700	2024-06-07
8	S008	2300	2024-06-08
9	S009	1900	2024-06-09
10	S010	2200	2024-06-10
NULL	NULL	NULL	NULL

Payments 56

Output

Action Output

#	Time	Action	Message
1	18:00:41	UPDATE Payments SET amount = 2000 WHERE payment_id = 5	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
2	18:00:41	SELECT * FROM Payments LIMIT 0, 1000	9 row(s) returned

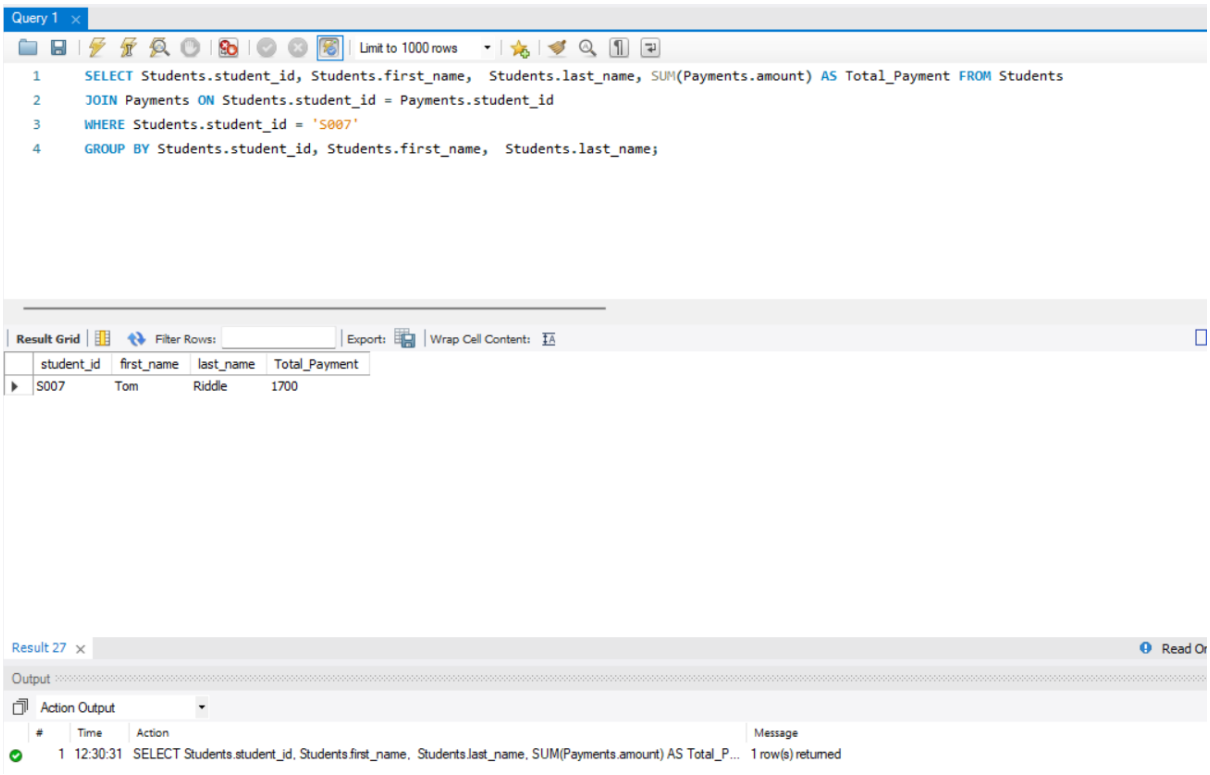
TASK – 03

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.

Query:

```
SELECT Students.student_id, Students.first_name, Students.last_name, SUM(Payments.amount) AS Total_Payment FROM Students
JOIN Payments ON Students.student_id = Payments.student_id
WHERE Students.student_id = 'S007'
GROUP BY Students.student_id, Students.first_name, Students.last_name;
```



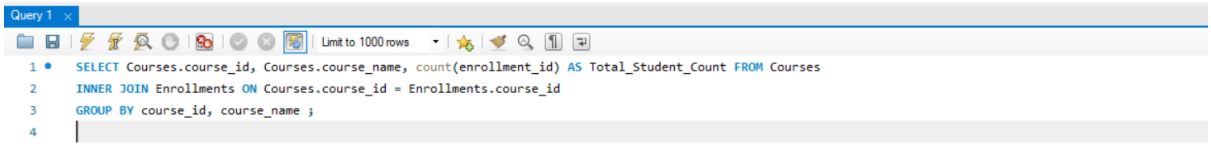
2. Write an SQL query to retrieve a list of courses along with the count of students enrolled in each course. Use a JOIN operation between the "Courses" table and the "Enrollments" table.

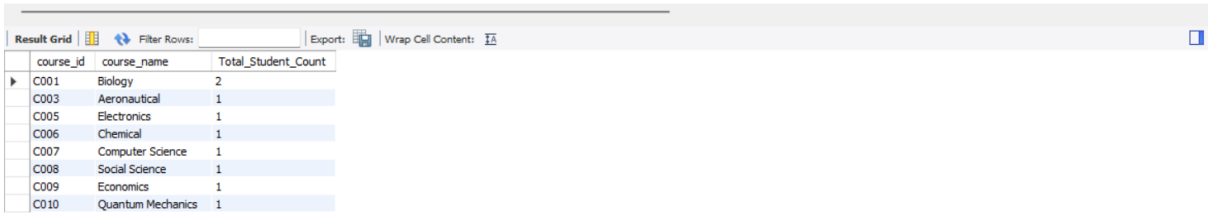
Query:

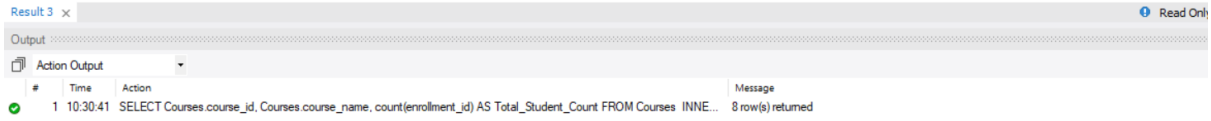
SELECT Courses.course_id, Courses.course_name, count(enrollment_id) AS Total_Student_Count FROM Courses

INNER JOIN Enrollments ON Courses.course_id = Enrollments.course_id

GROUP BY course_id, course_name ;







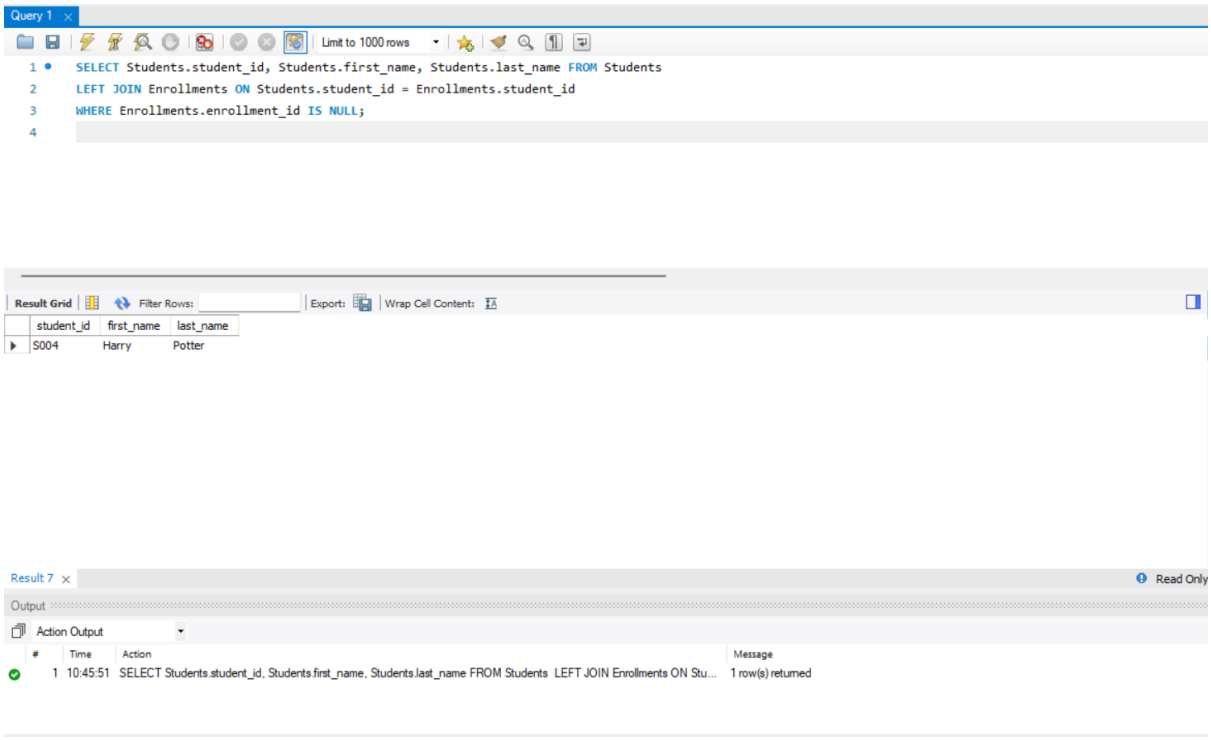
3. Write an SQL query to find the names of students who have not enrolled in any course. Use a LEFT JOIN between the "Students" table and the "Enrollments" table to identify students without enrollments.

Query:

SELECT Students.student_id, Students.first_name, Students.last_name FROM Students

LEFT JOIN Enrollments ON Students.student_id = Enrollments.student_id

WHERE Enrollments.enrollment_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.

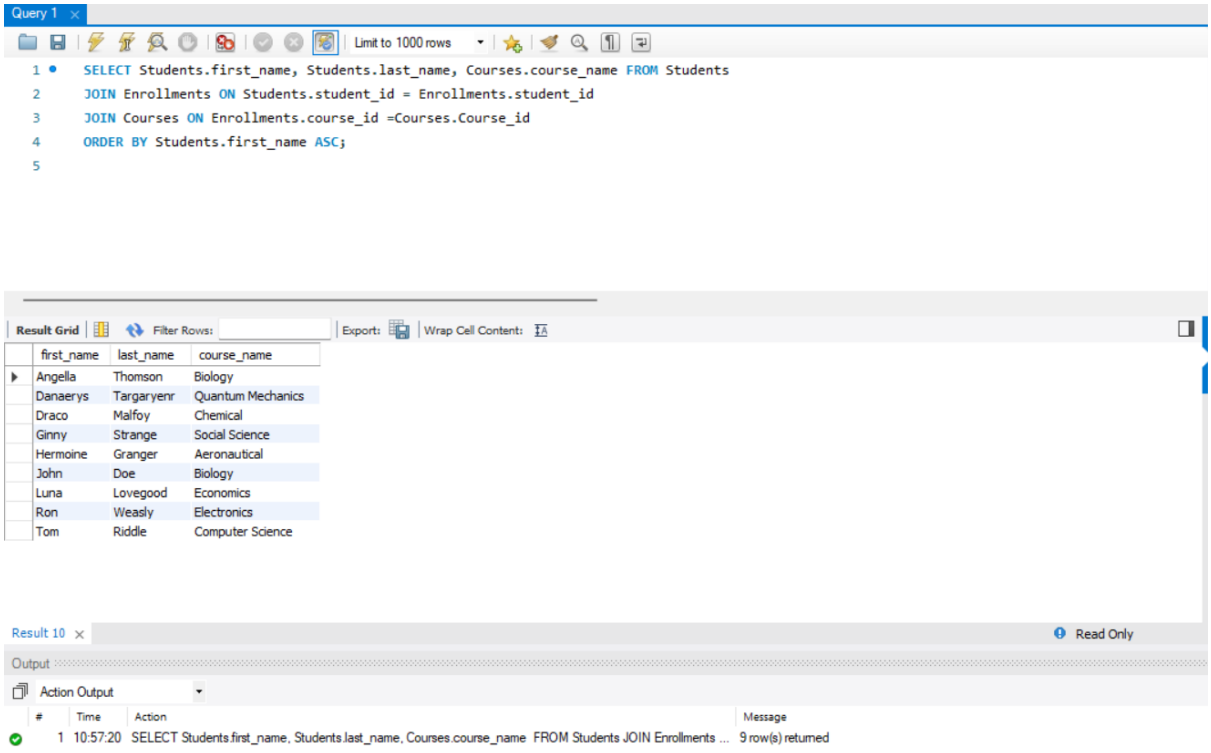
Query:

SELECT Students.first_name, Students.last_name, Courses.course_name FROM Students

JOIN Enrollments ON Students.student_id = Enrollments.student_id

JOIN Courses ON Enrollments.course_id =Courses.Course_id

ORDER BY Students.first_name ASC;



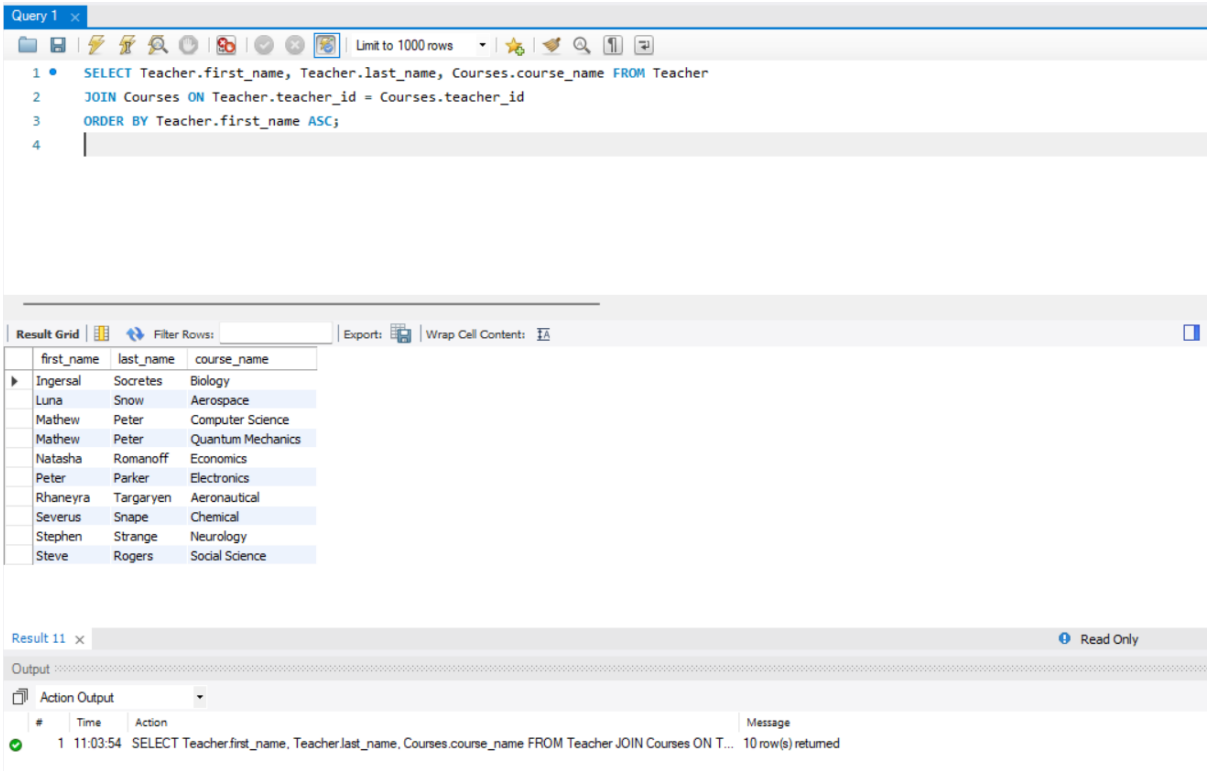
5. Create a query to list the names of teachers and the courses they are assigned to. Join the "Teacher" table with the "Courses" table.

Query:

SELECT Teacher.first_name, Teacher.last_name, Courses.course_name FROM Teacher

JOIN Courses ON Teacher.teacher_id = Courses.teacher_id

ORDER BY Teacher.first_name ASC;



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.

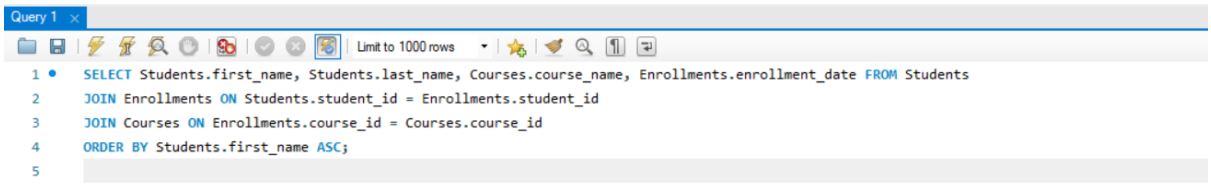
Query:

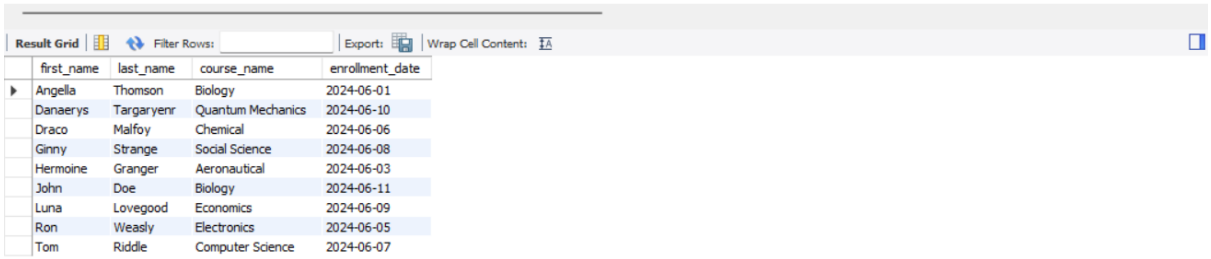
SELECT Students.first_name, Students.last_name, Courses.course_name, Enrollments.enrollment_date FROM Students

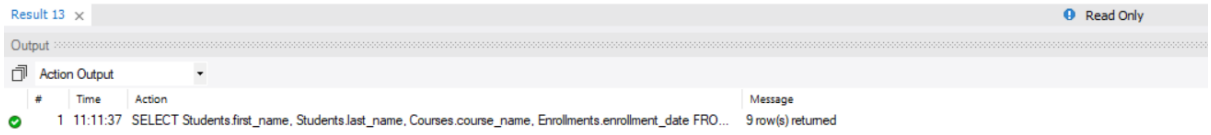
JOIN Enrollments ON Students.student_id = Enrollments.student_id

JOIN Courses ON Enrollments.course_id = Courses.course_id

ORDER BY Students.first_name ASC;







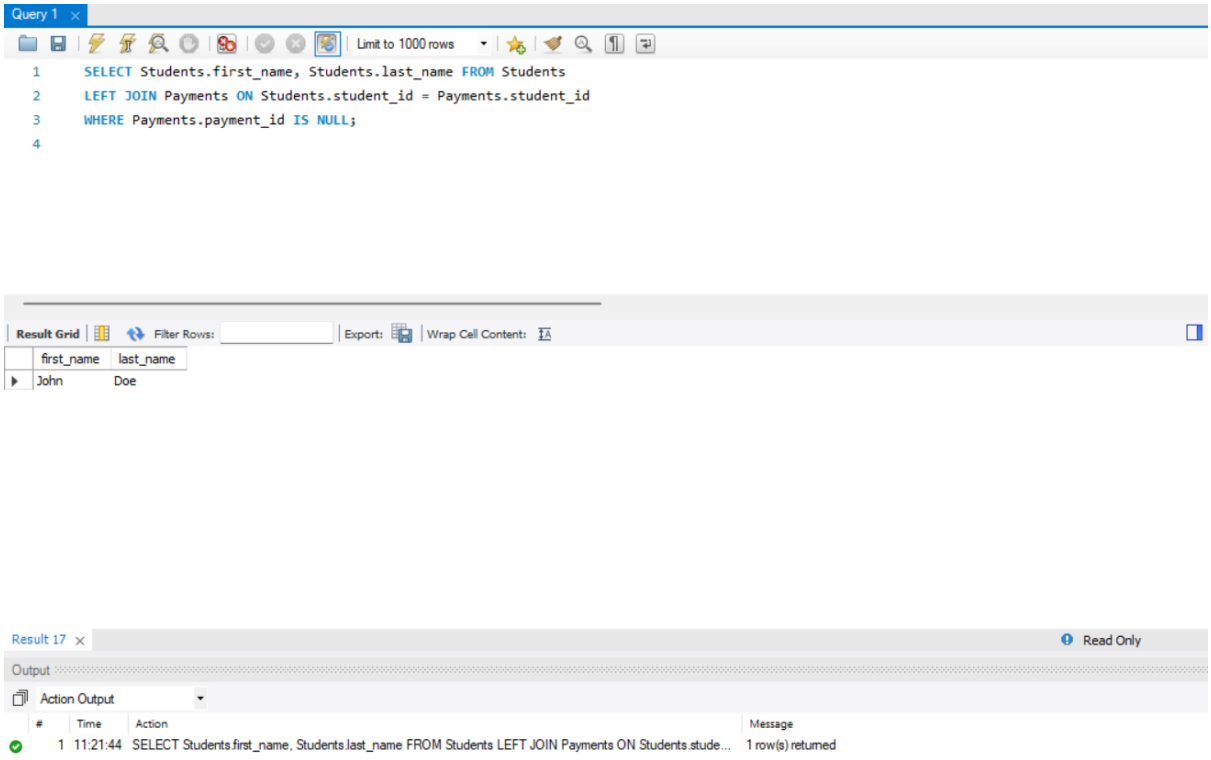
7. Find the names of students who have not made any payments. Use a LEFT JOIN between the "Students" table and the "Payments" table and filter for students with NULL payment records.

Query:

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;



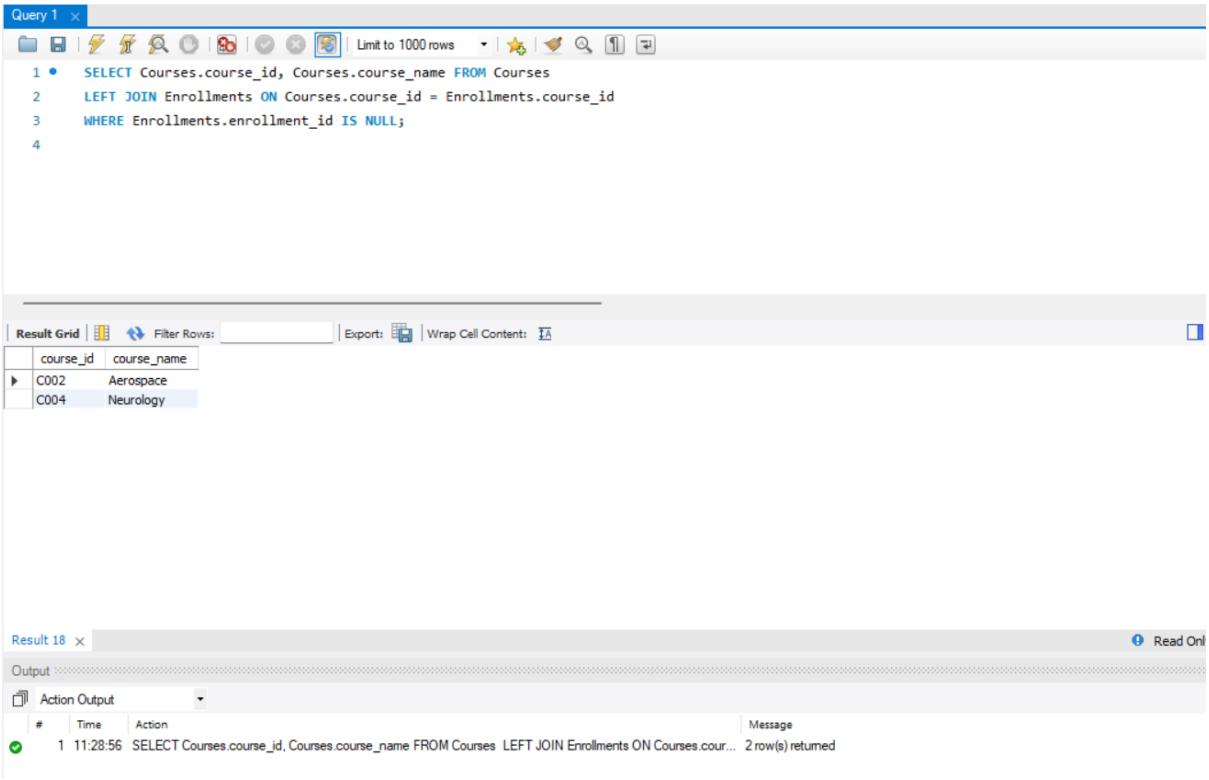
8. Write a query to identify courses that have no enrollments. You'll need to use a LEFT JOIN between the "Courses" table and the "Enrollments" table and filter for courses with NULL enrollment records.

Query:

SELECT Courses.course_id, Courses.course_name FROM Courses

LEFT JOIN Enrollments ON Courses.course_id = Enrollments.course_id

WHERE Enrollments.enrollment_id IS NULL;



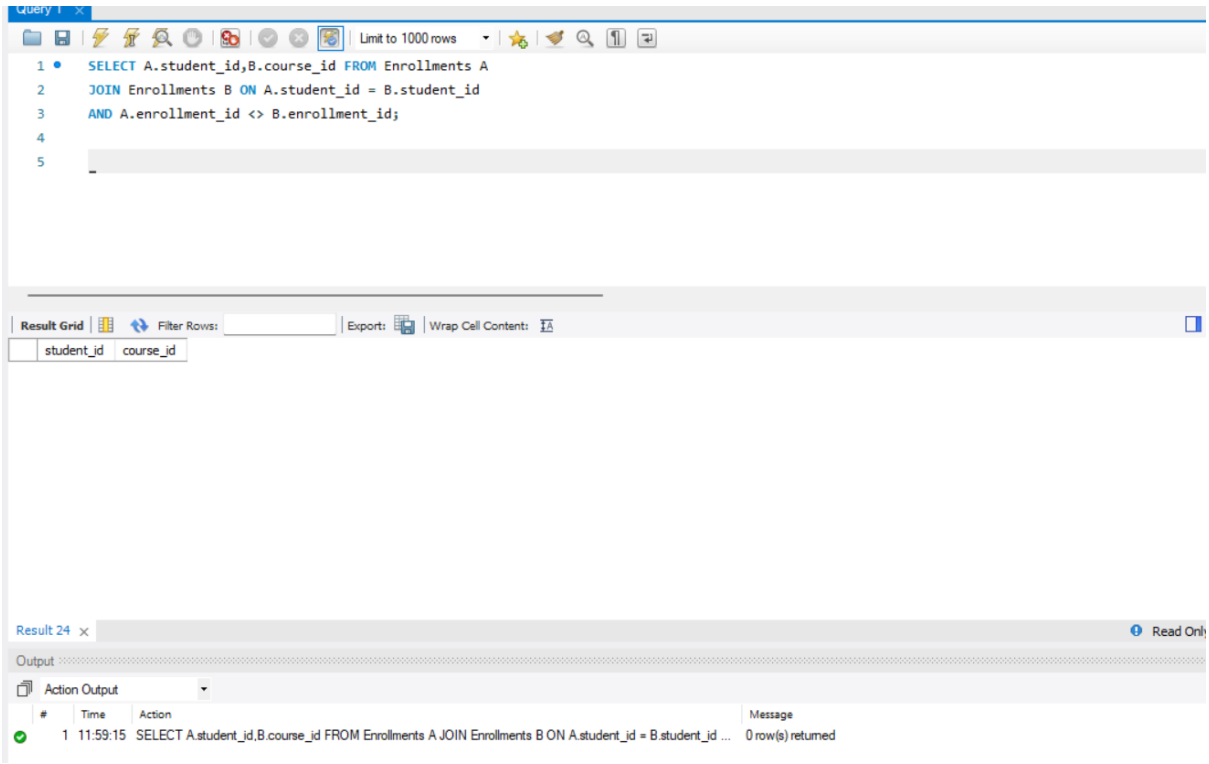
9. Identify students who are enrolled in more than one course. Use a self-join on the "Enrollments" table to find students with multiple enrollment records.

Query:

SELECT A.student_id,B.course_id FROM Enrollments A

JOIN Enrollments B ON A.student_id = B.student_id

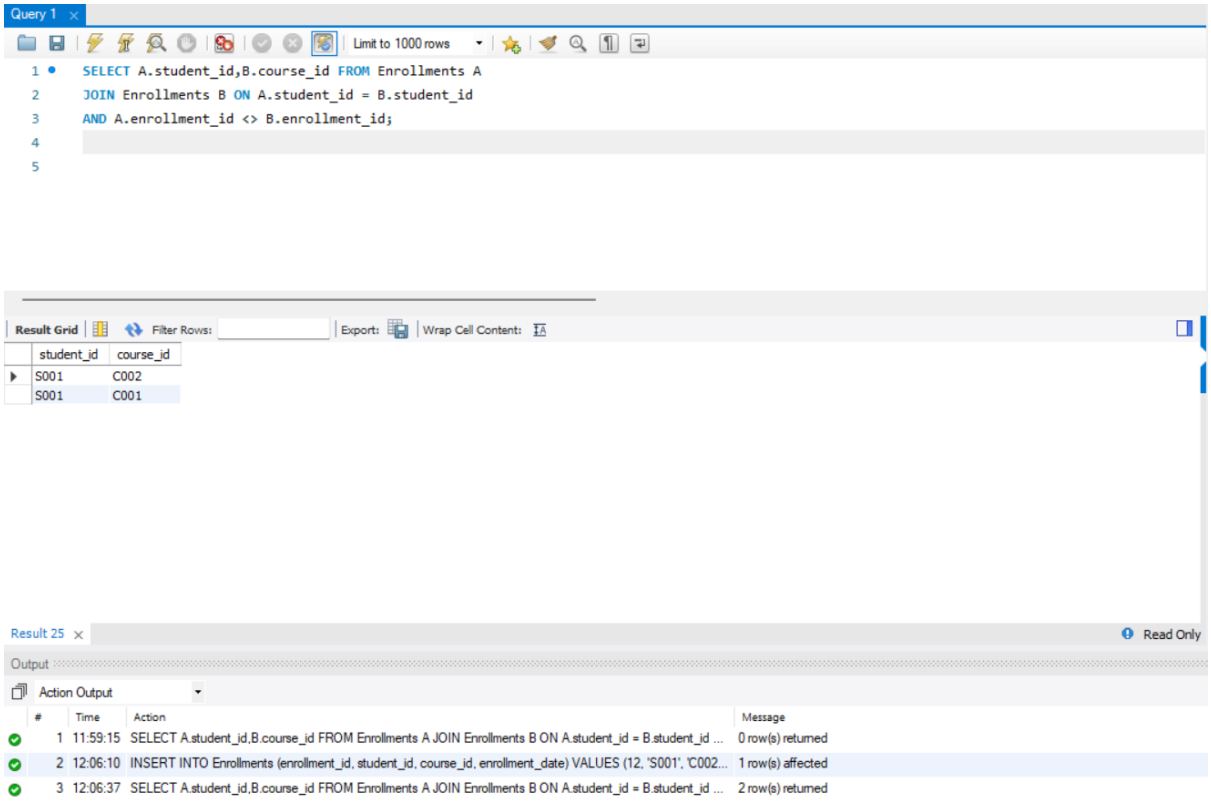
AND A.enrollment_id <> B.enrollment_id;



At my current data entry, the query result shows empty. So let us insert a extra enrollment for same student.

INSERT INTO Enrollments (enrollment_id, student_id, course_id, enrollment_date)

VALUES (12, 'S001', 'C002', '2024-06-12');



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.

Query:

SELECT Teacher.teacher_id , Teacher.first_name, Teacher.last_name From Teacher

LEFT JOIN Courses ON Teacher.teacher_id = Courses.teacher_id

WHERE Courses.course_id IS NULL;

Query 1

```
1  SELECT Teacher.teacher_id , Teacher.first_name, Teacher.last_name From Teacher
2  LEFT JOIN Courses ON Teacher.teacher_id = Courses.teacher_id
3  WHERE Courses.course_id IS NULL;
4
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

teacher_id	first_name	last_name
T010	Bruce	Banner

Result 26 | Read Only

Output

Action Output

#	Time	Action	Message
1	12:21:12	SELECT Teacher.teacher_id , Teacher.first_name, Teacher.last_name From Teacher LEFT JOIN Courses O...	1 row(s) returned

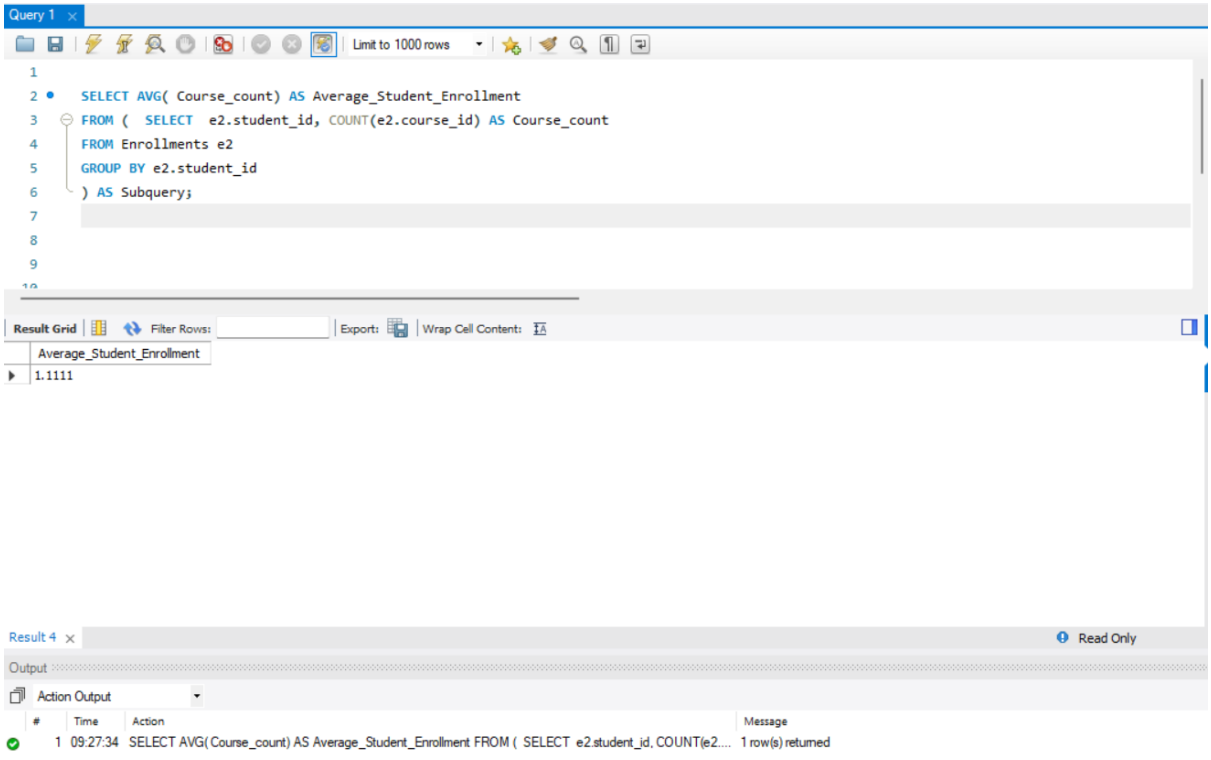
TASK – 04

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.

Query:

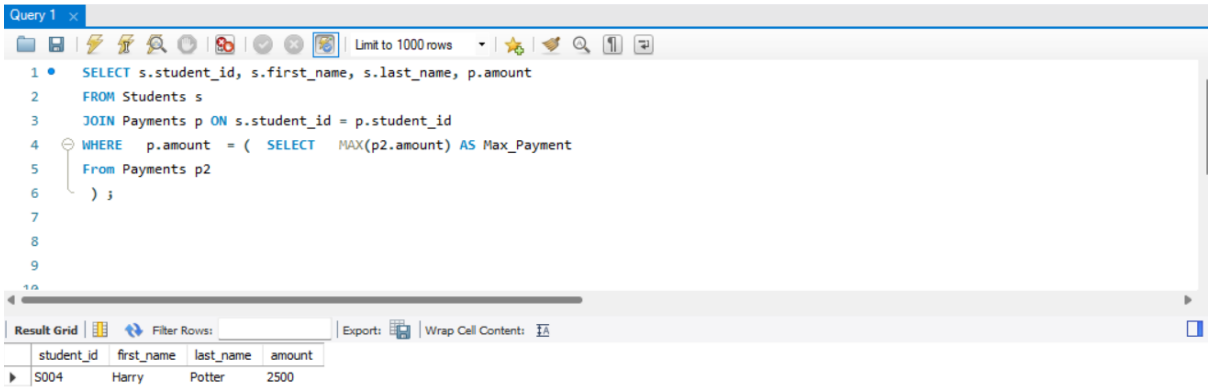
```
SELECT AVG(Course_count) AS Average_Student_Enrollment
FROM ( SELECT  e2.student_id, COUNT(e2.course_id) AS Course_count
FROM Enrollments e2
GROUP BY e2.student_id
) AS Subquery;
```

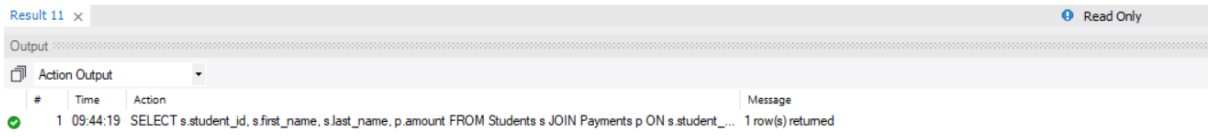


2. Identify the student(s) who made the highest payment. Use a subquery to find the maximum payment amount and then retrieve the student(s) associated with that amount.

Query:

```
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(p2.amount) AS Max_Payment
From Payments p2
) ;
```





3. Retrieve a list of courses with the highest number of enrollments. Use subqueries to find the course(s) with the maximum enrollment count.

Query :

```
SELECT c.course_id, c.course_name, COUNT(e.enrollment_id) AS enroll_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.enrollment_id) = (

SELECT MAX(course_count) FROM (

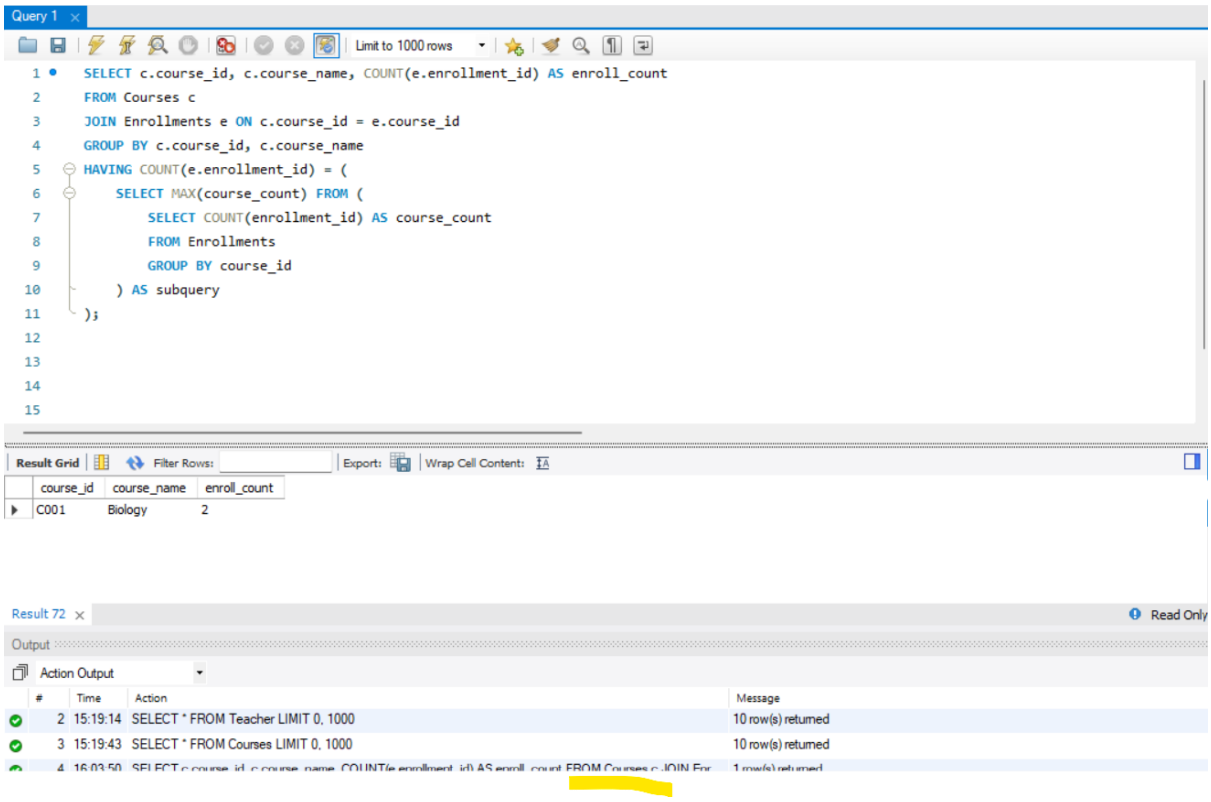
SELECT COUNT(enrollment_id) AS course_count

FROM Enrollments

GROUP BY course_id

) AS subquery

);

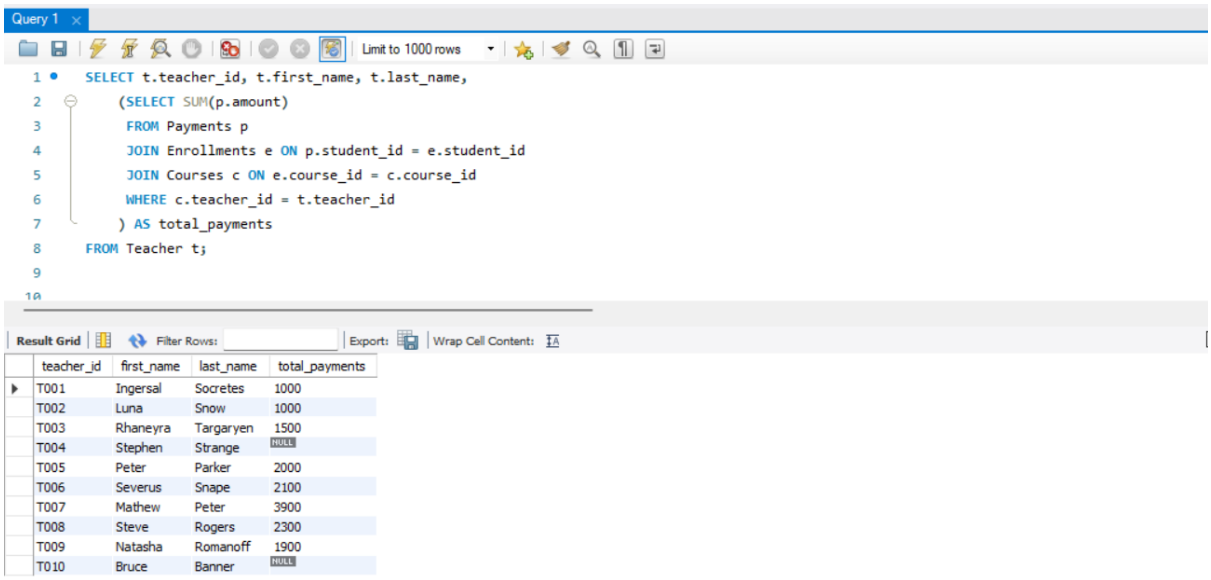


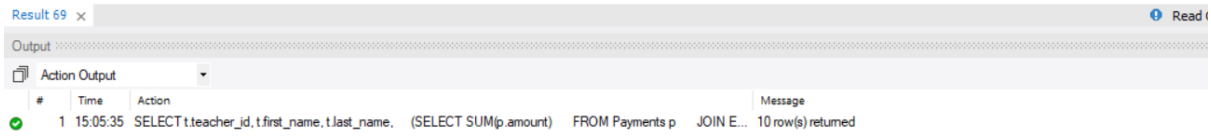
4. Calculate the total payments made to courses taught by each teacher. Use subqueries to sum payments for each teacher's courses.

Query:

SELECT t.teacher_id, t.first_name, t.last_name,

```
    (SELECT SUM(p.amount)
     FROM Payments p
     JOIN Enrollments e ON p.student_id = e.student_id
     JOIN Courses c ON e.course_id = c.course_id
     WHERE c.teacher_id = t.teacher_id
    ) AS total_payments
FROM Teacher t;
```





5. Identify students who are enrolled in all available courses. Use subqueries to compare a student's enrollments with the total number of courses.

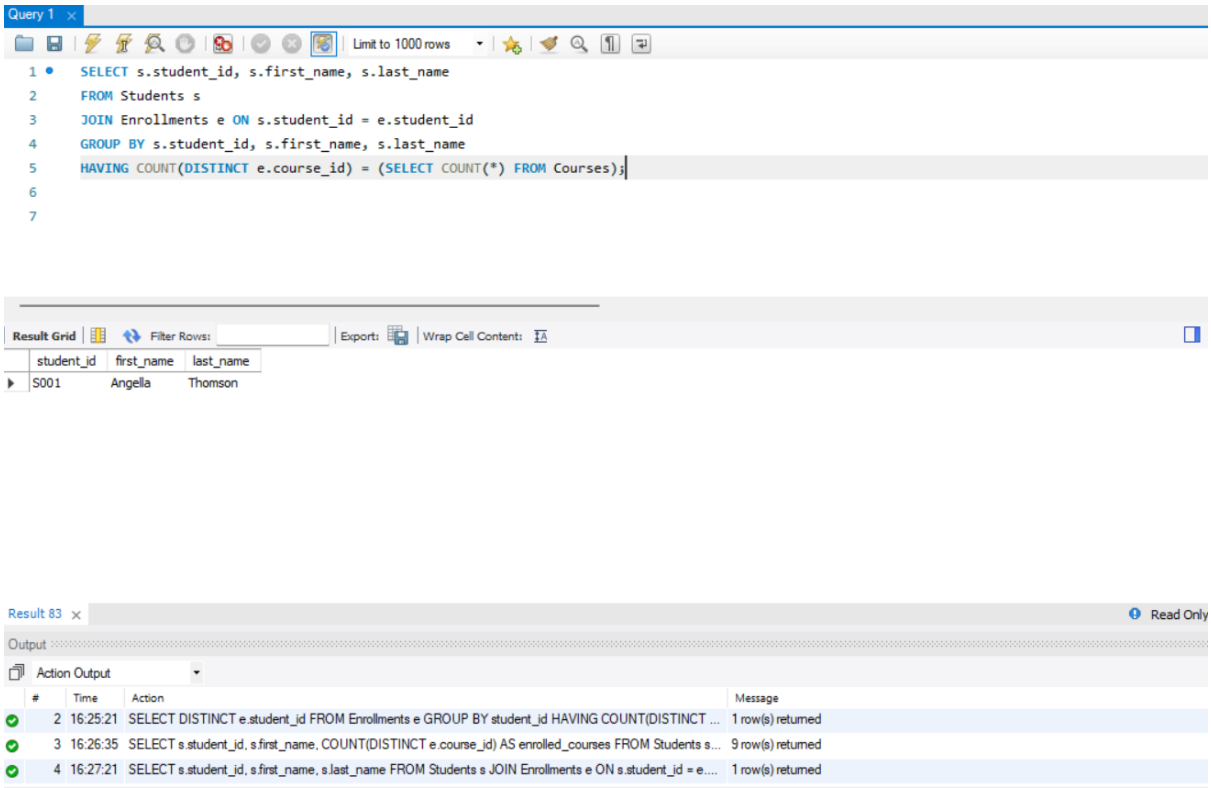
Query:

```
SELECT s.student_id, s.first_name, s.last_name
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(DISTINCT e.course_id) = (SELECT COUNT(*) 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.

Query:

SELECT t.teacher_id, t.first_name, t.last_name

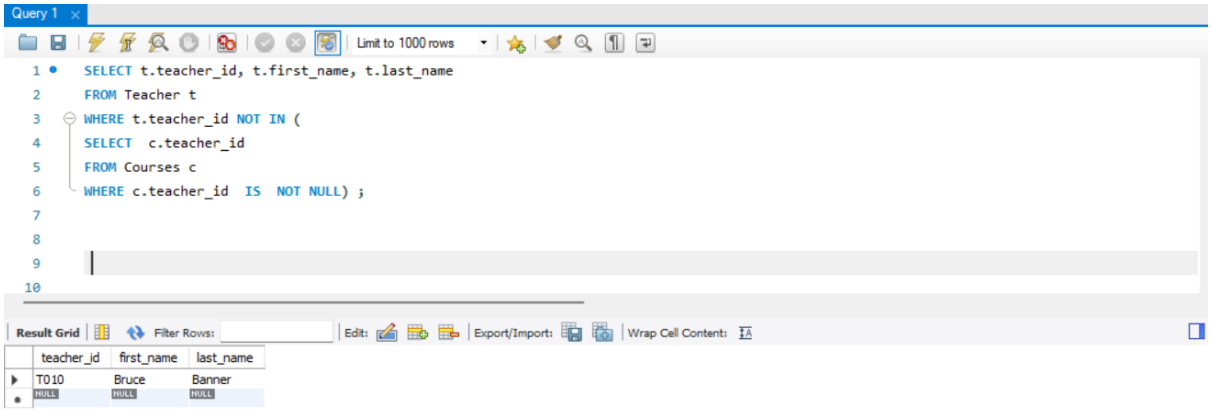
FROM Teacher t

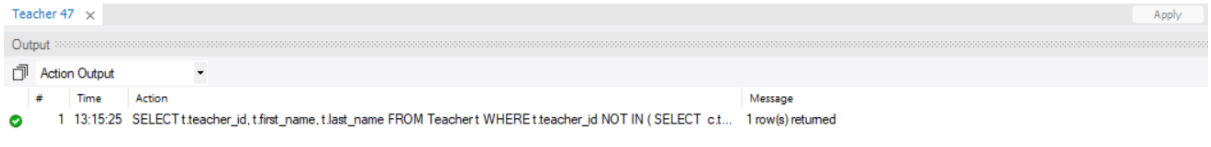
WHERE t.teacher_id NOT IN (

SELECT c.teacher_id

FROM Courses c

WHERE c.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.

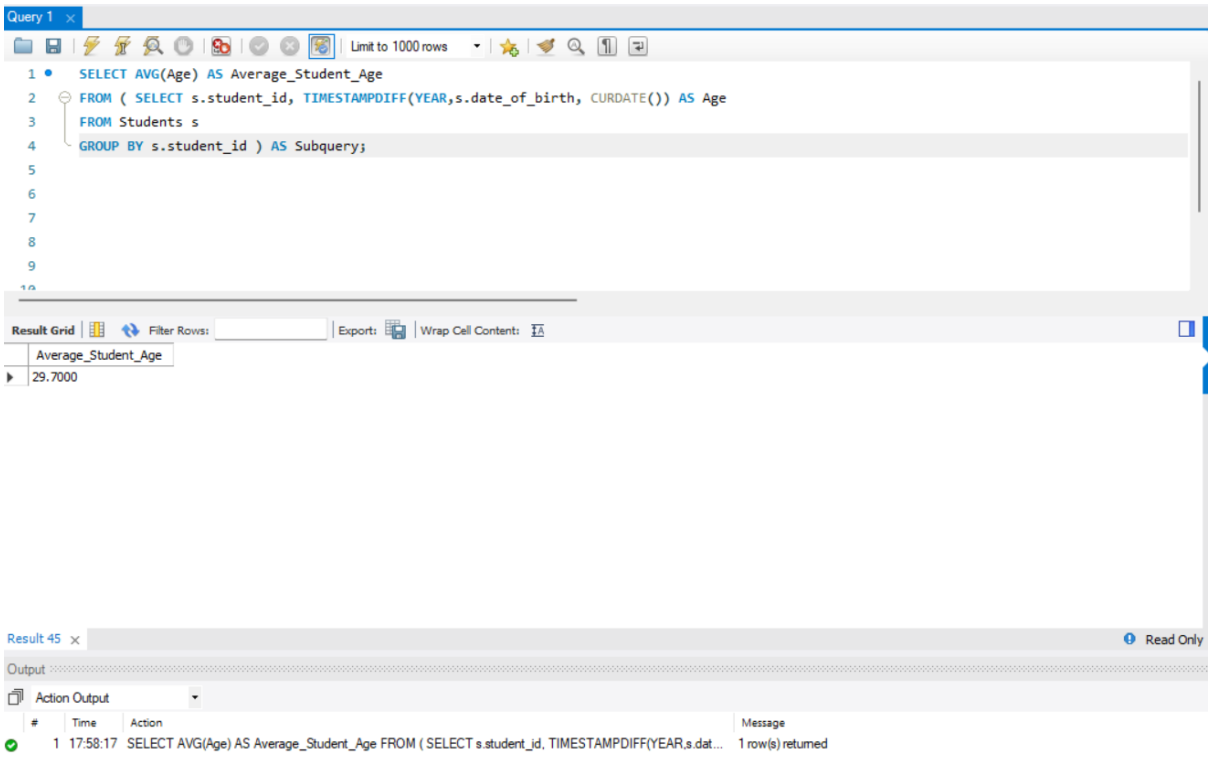
Query:

SELECT AVG(Age) AS Average_Student_Age

FROM (SELECT s.student_id, TIMESTAMPDIFF(YEAR,s.date_of_birth, CURDATE()) AS Age

FROM Students s

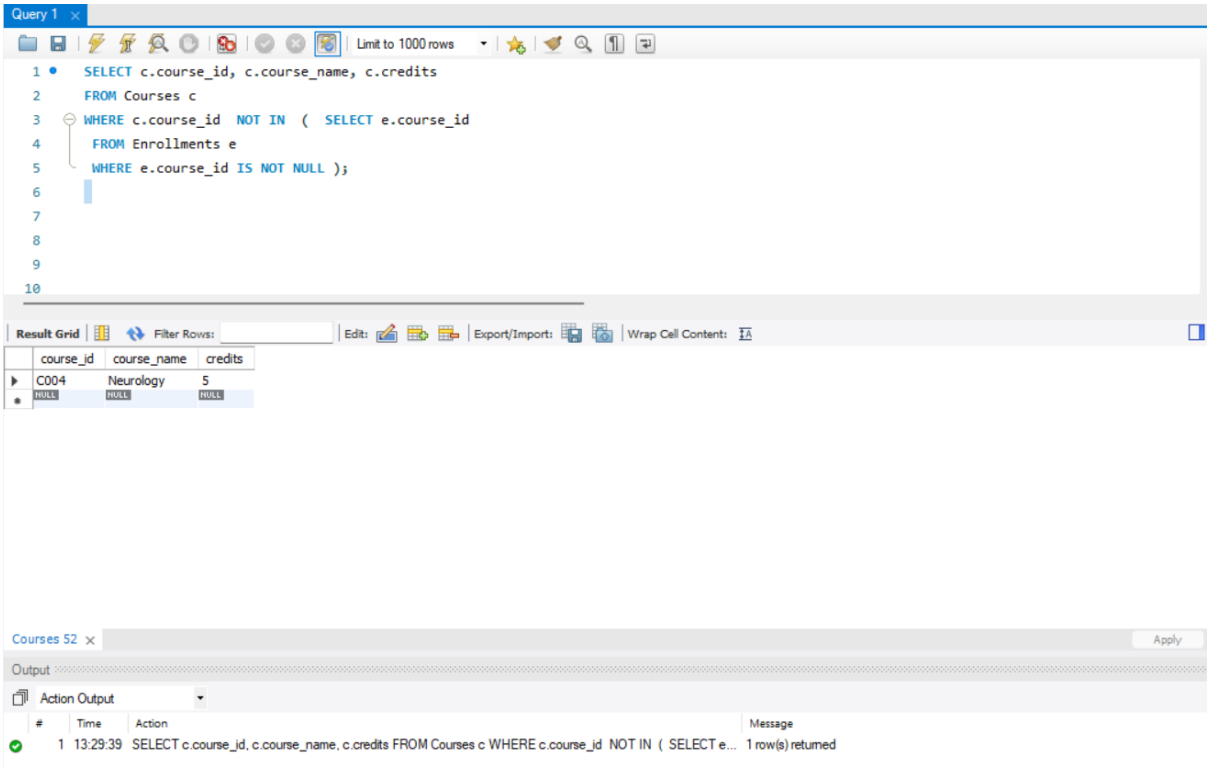
GROUP BY s.student_id) AS Subquery;



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

Query:

```
SELECT c.course_id, c.course_name, c.credits
FROM Courses c
WHERE c.course_id  NOT IN  (  SELECT e.course_id
 FROM Enrollments e
 WHERE e.course_id IS NOT NULL );
```



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

Query:

SELECT s.student_id, s.first_name, s.last_name, e.course_id, c.course_name, p.amount AS Amount_paid

FROM Students s

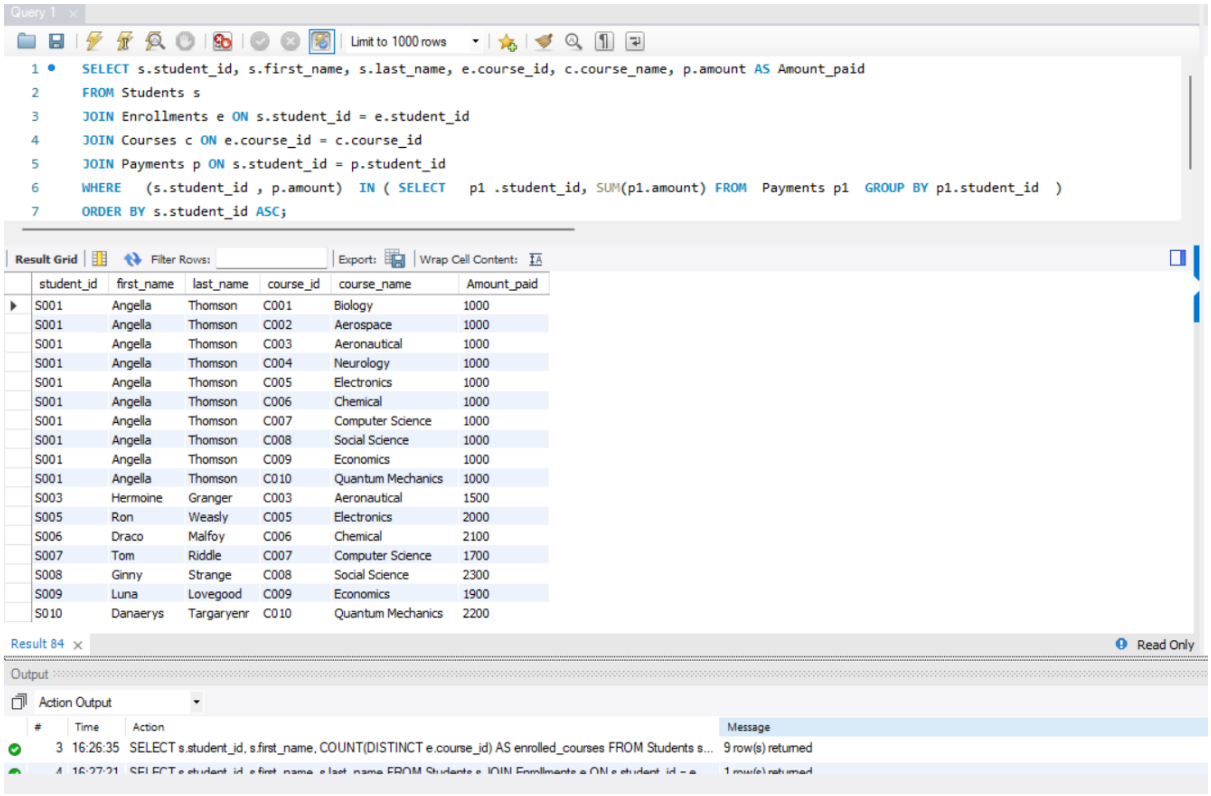
JOIN Enrollments e ON s.student_id = e.student_id

JOIN Courses c ON e.course_id = c.course_id

JOIN Payments p ON s.student_id = p.student_id

WHERE (s.student_id , p.amount) IN (SELECT p1 .student_id, SUM(p1.amount) FROM Payments p1 GROUP BY p1.student_id)

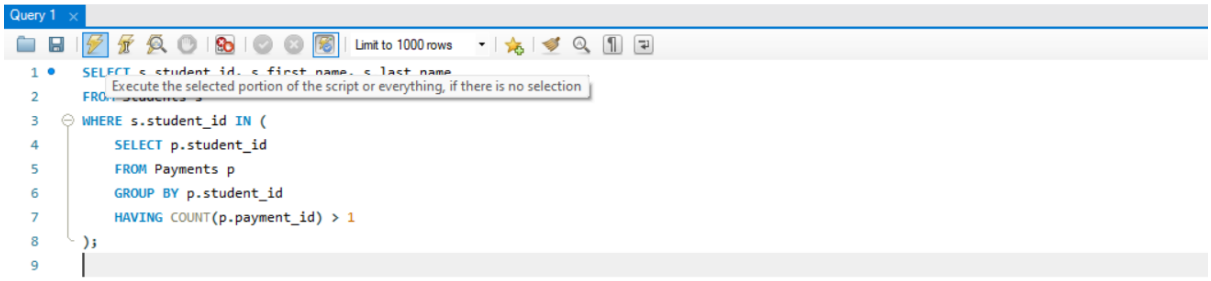
ORDER BY s.student_id ASC;

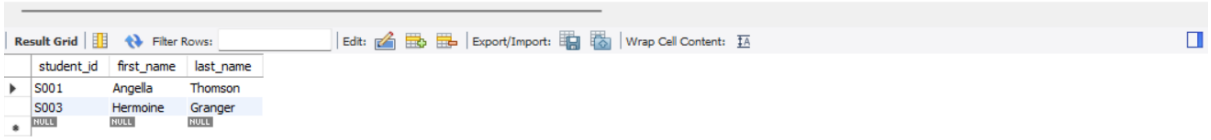


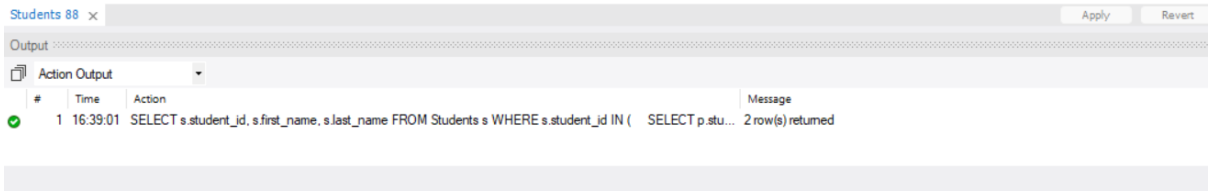
10. Identify students who have made more than one payment. Use subqueries and aggregate functions to count payments per student and filter for those with counts greater than one.

Query:

```
SELECT s.student_id, s.first_name, s.last_name
FROM Students s
WHERE s.student_id IN (
    SELECT p.student_id
    FROM Payments p
    GROUP BY p.student_id
    HAVING COUNT(p.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.

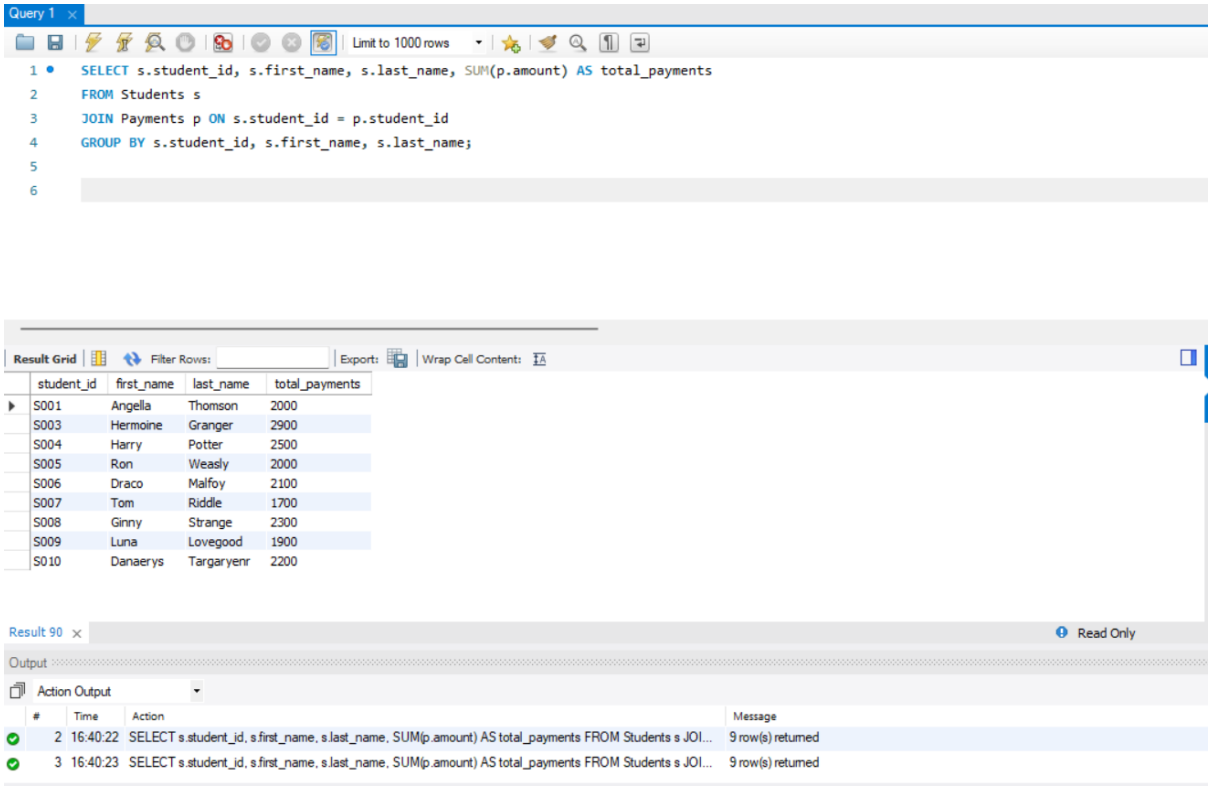
Query:

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, 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.

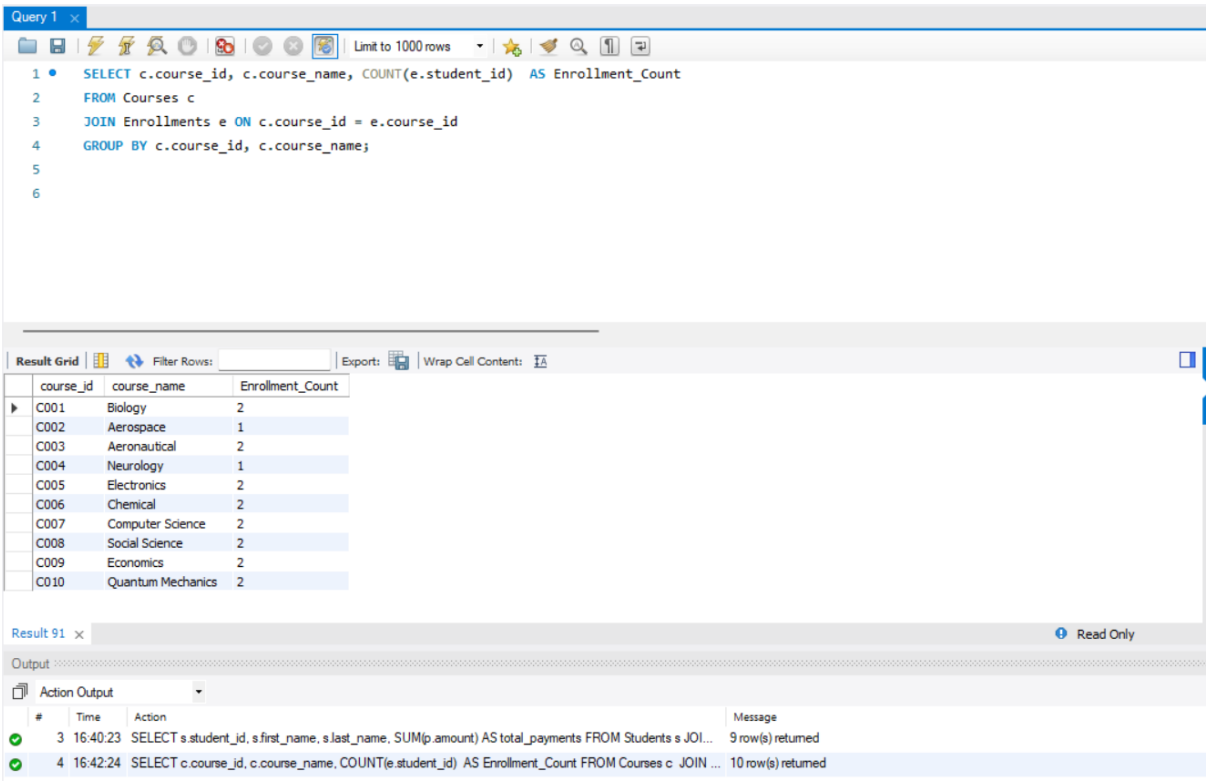
Query:

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;



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.

Query:

SELECT AVG(Total_amount) AS Average_Payment_Amount

FROM(SELECT s.student_id, Sum(p.amount) AS Total_amount

FROM Students s

JOIN Payments p ON s.student_id = p.student_id

GROUP BY s.student_id) AS Subquery;

Query 1

Limit to 1000 rows

```
1  SELECT AVG(Total_amount) AS Average_Payment_Amount
2  FROM(SELECT s.student_id, Sum(p.amount) AS Total_amount
3  FROM Students s
4  JOIN Payments p ON s.student_id = p.student_id
5  GROUP BY s.student_id) AS Subquery;
6
7
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Average_Payment_Amount
2177.7778

Result 92 | Read Only

Output

Action Output

#	Time	Action	Message
4	16:42:24	SELECT c.course_id, c.course_name, COUNT(e.student_id) AS Enrollment_Count FROM Courses c JOIN ...	10 row(s) returned
5	16:43:33	SELECT AVG(Total_amount) AS Average_Payment_Amount FROM(SELECT s.student_id, Sum(p.amount) ...	1 row(s) returned