SQL Project

Project Overview:

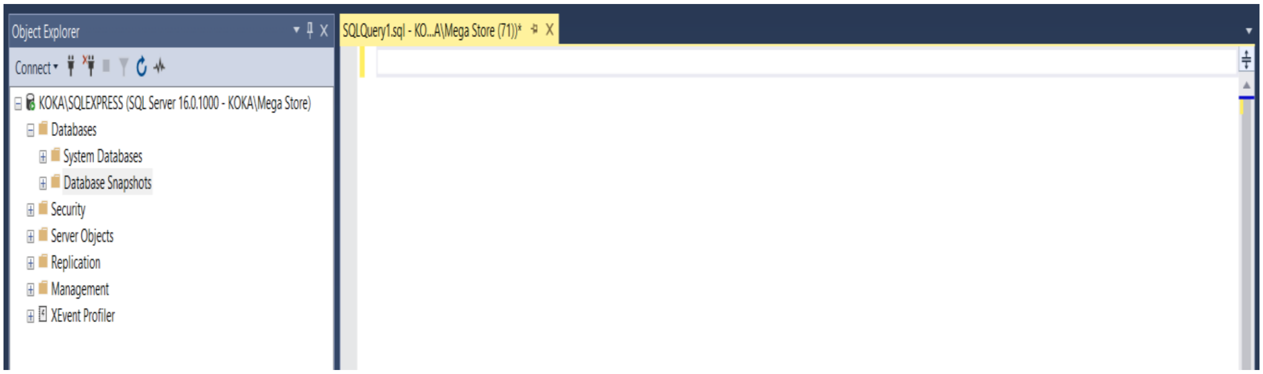
This project involves creating a database for managing student courses. The database will have tables for students, courses, enrollments, and instructors. Students will learn and practice SQL queries covering various topics such as selection, filtering, aggregation, joins, and subqueries.

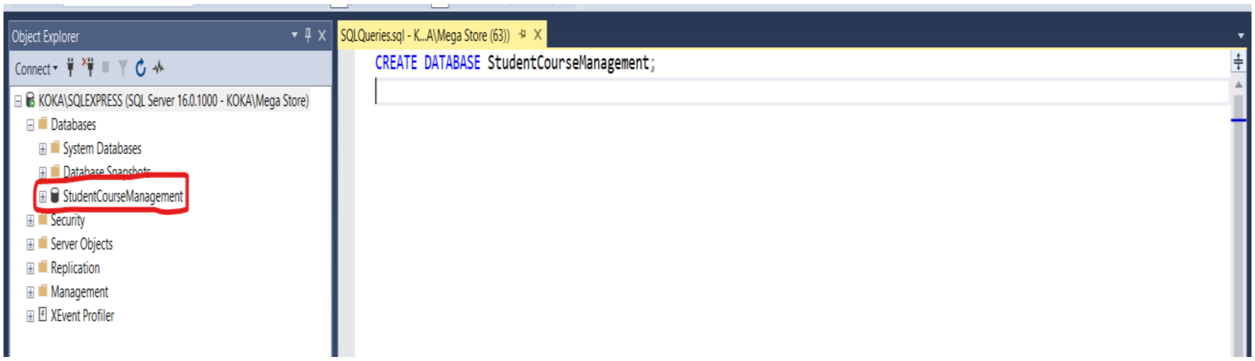
1.Database Setup:

Create a database named Student Course Management.

Query:

```
CREATE DATABASE StudentCourseManagement;
```





```sql
CREATE DATABASE StudentCourseManagement;
```

2.Table Creation:

.student table:

student_id (Primary Key, INT, AUTO_INCREMENT)

first_name (VARCHAR)

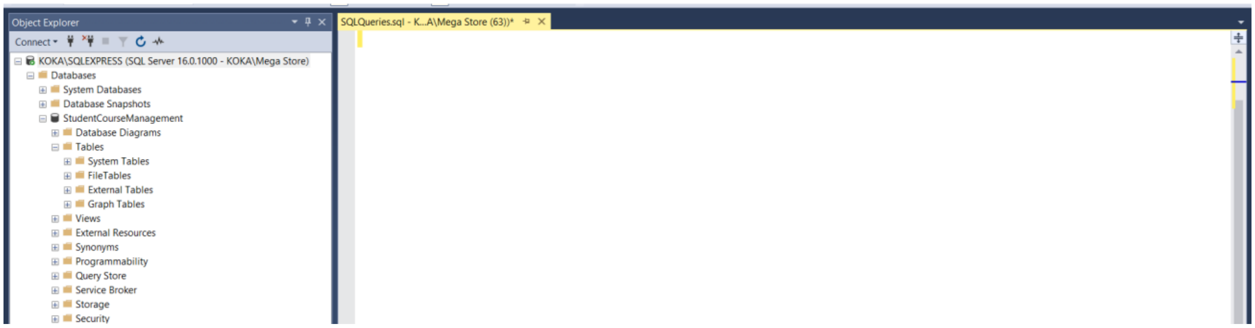
last_name (VARCHAR)

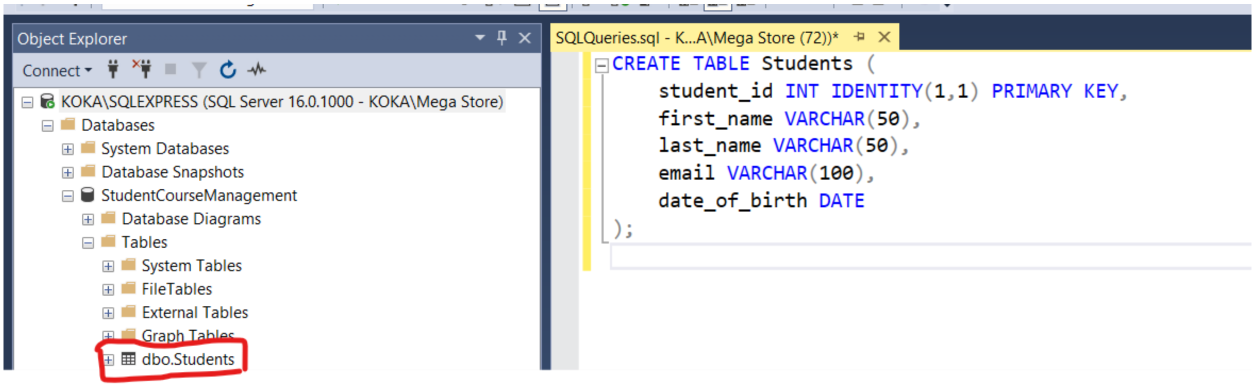
email (VARCHAR)

date_of_birth (DATE)

query:

```
CREATE TABLE Students (
    student_id INT IDENTITY(1,1) PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100),
    date_of_birth DATE
);
```





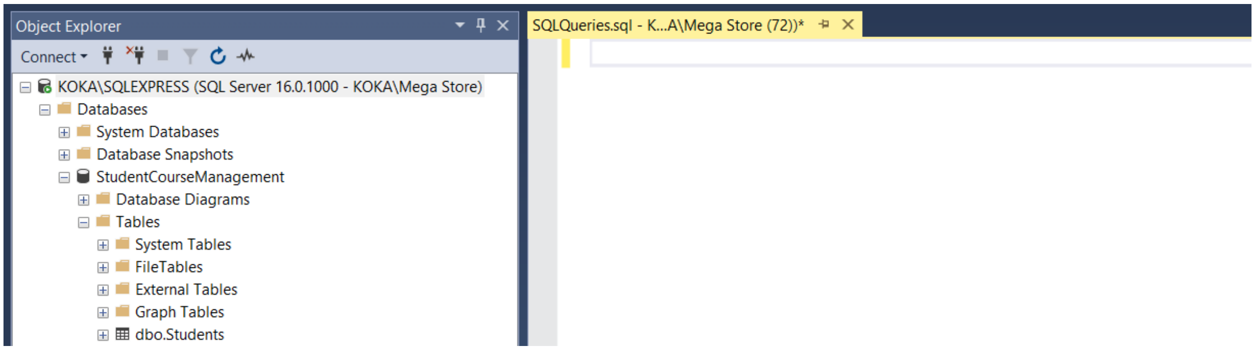
```
CREATE TABLE Students (
    student_id INT IDENTITY(1,1) PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100),
    date_of_birth DATE
);
```

.course table:

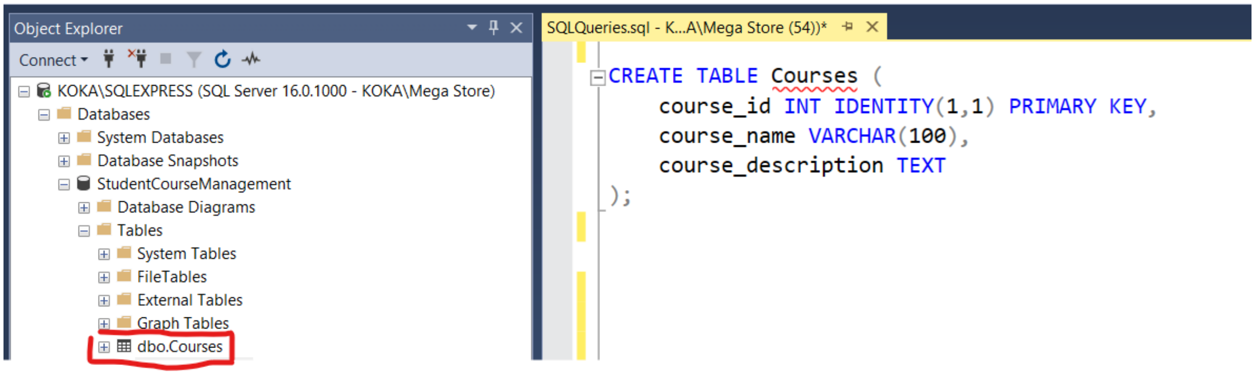
- course_id (Primary Key, INT, AUTO_INCREMENT)
- course_name (VARCHAR)
- course_description (TEXT)

query:

```
CREATE TABLE Courses (
    course_id INT IDENTITY(1,1) PRIMARY KEY,
    course_name VARCHAR(100),
    course_description TEXT
);
```



```
⊟ KOKA\SQLEXPRESS (SQL Server 16.0.1000 - KOKA\Mega Store)
  ⊟ Databases
    ⊞ System Databases
    ⊞ Database Snapshots
    ⊟ StudentCourseManagement
      ⊞ Database Diagrams
      ⊟ Tables
        ⊞ System Tables
        ⊞ FileTables
        ⊞ External Tables
        ⊞ Graph Tables
        ⊞ dbo.Students
```



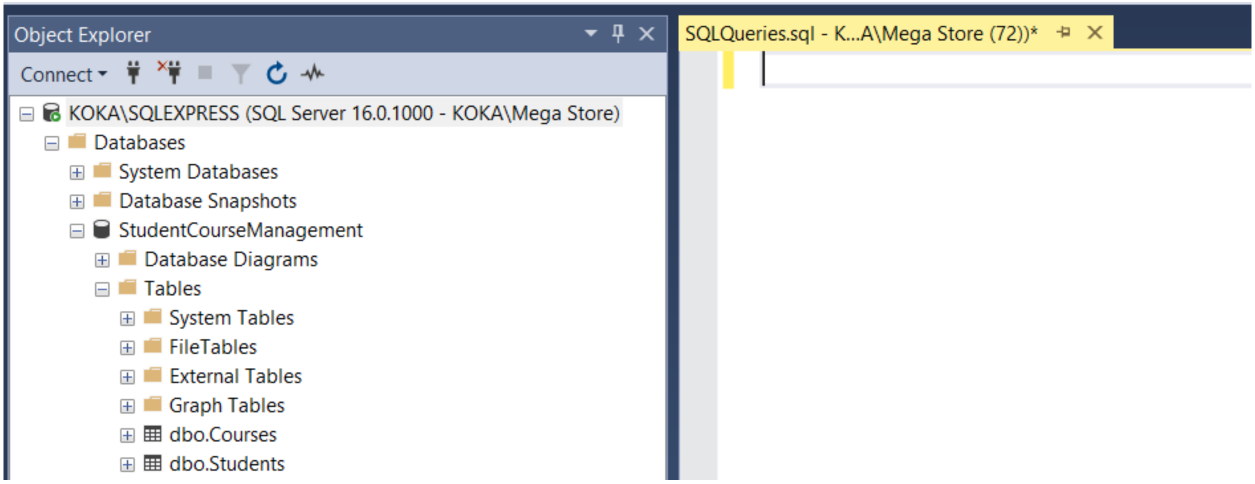
```sql
CREATE TABLE Courses (
    course_id INT IDENTITY(1,1) PRIMARY KEY,
    course_name VARCHAR(100),
    course_description TEXT
);
```

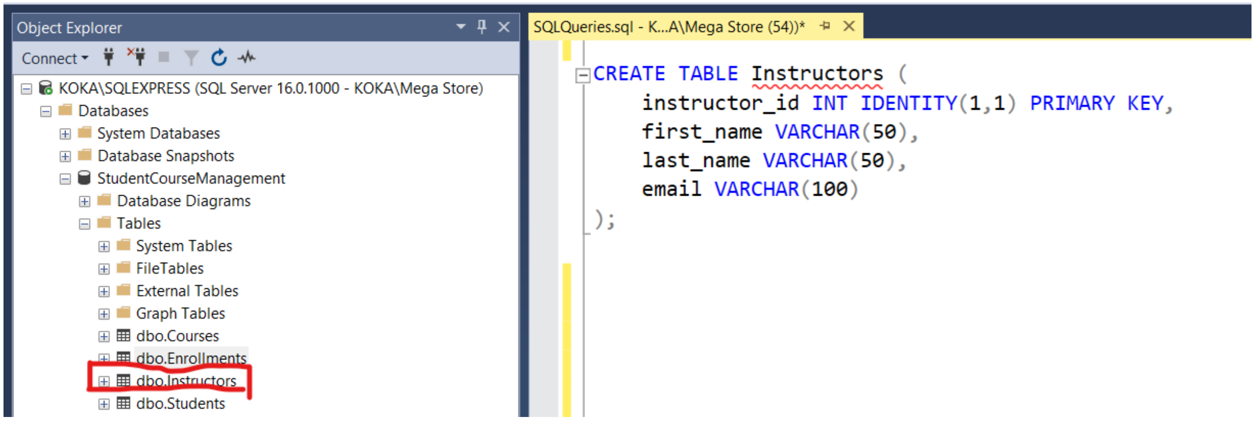
.instructor table:

- instructor_id (Primary Key, INT, AUTO_INCREMENT)
- first_name (VARCHAR)
- last_name (VARCHAR)
- email (VARCHAR)

query:

```sql
CREATE TABLE Instructors (
    instructor_id INT IDENTITY(1,1) PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100)
);
```





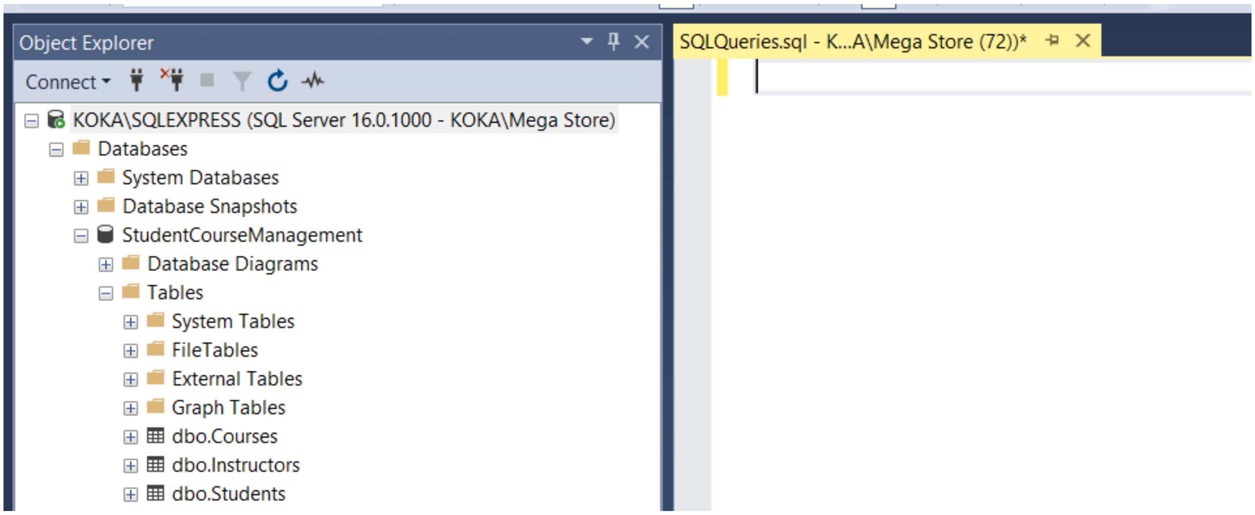
```sql
CREATE TABLE Instructors (
    instructor_id INT IDENTITY(1,1) PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100)
);
```

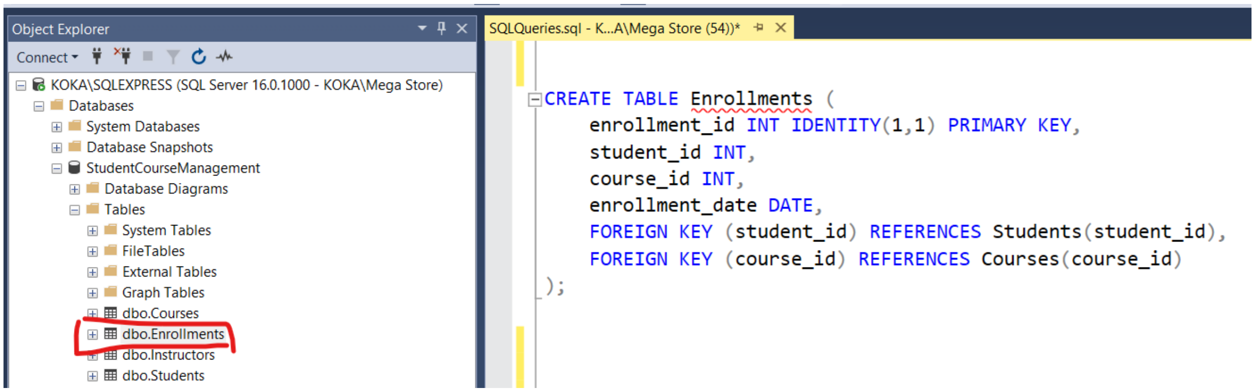
.enrollment table:

- enrollment_id (Primary Key, INT, AUTO_INCREMENT)
- student_id (Foreign Key, INT)
- course_id (Foreign Key, INT)
- enrollment_date (DATE)

query:

```
CREATE TABLE Enrollments (
    enrollment_id INT IDENTITY(1,1) 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 Enrollments (
    enrollment_id INT IDENTITY(1,1) 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)
);
```

3.Insert Sample Data:

- Insert at least 10 students, 5 courses, 3 instructors, and 15 enrollments.

1.Insert Students Data:

query:

INSERT INTO Students (first_name, last_name, email, date_of_birth) VALUES

('kirolos', 'magdy', 'kirolosmagdy@gmail.com', '2003-02-28'),

('ahmed', 'nabil', 'ahmednabil@gmail.com', '2000-11-22'),

('omar', 'nouh', 'omarnouh@gmail.com', '2002-06-15'),

```
('ahmed', 'hassan', 'ahmedhassan@gmail.com', '2003-03-08'),
('mazen', 'ahmed', 'mazenahmed@gmail.com', '1999-09-19'),
('madonna', 'daniel', 'madonnadaniel@gmail.com', '2004-01-10'),
('farah', 'mohamed', 'farahmohamed@gmail.com', '2000-07-25'),
('mario', 'nabil', 'marionabil@gmail.com', '2001-05-30'),
('arsany', 'noshy', 'arsanynoshy@gmail.com', '2002-12-20'),
('mina', 'medhat', 'minamedhat@gmail.com', '1998-10-05');
```

2. **Insert Courses Data:**

query:

```
INSERT INTO Courses (course_name, course_description) VALUES
('python', 'Introduction to python'),
('SQL', 'Introduction to SQL'),
('machine learning', 'Introduction to machine learning'),
('AI', 'Advanced AI'),
('data engineering', 'Introduction to data engineering');
```

3. Insert Instructors Data:

query:

```
INSERT INTO Instructors (first_name, last_name, email) VALUES
('ahmed', 'azab', 'ahmedazab@gmail.com'),
('fady', 'maged', 'fadymaged@gmail.com'),
('passant', 'mohamed', 'passantmohamed@gmail.com');
```

4. Insert Enrollments Data:

query:

```
INSERT INTO Enrollments (student_id, course_id, enrollment_date) VALUES
(1, 1, '2023-08-01'),
(2, 1, '2023-08-02'),
```

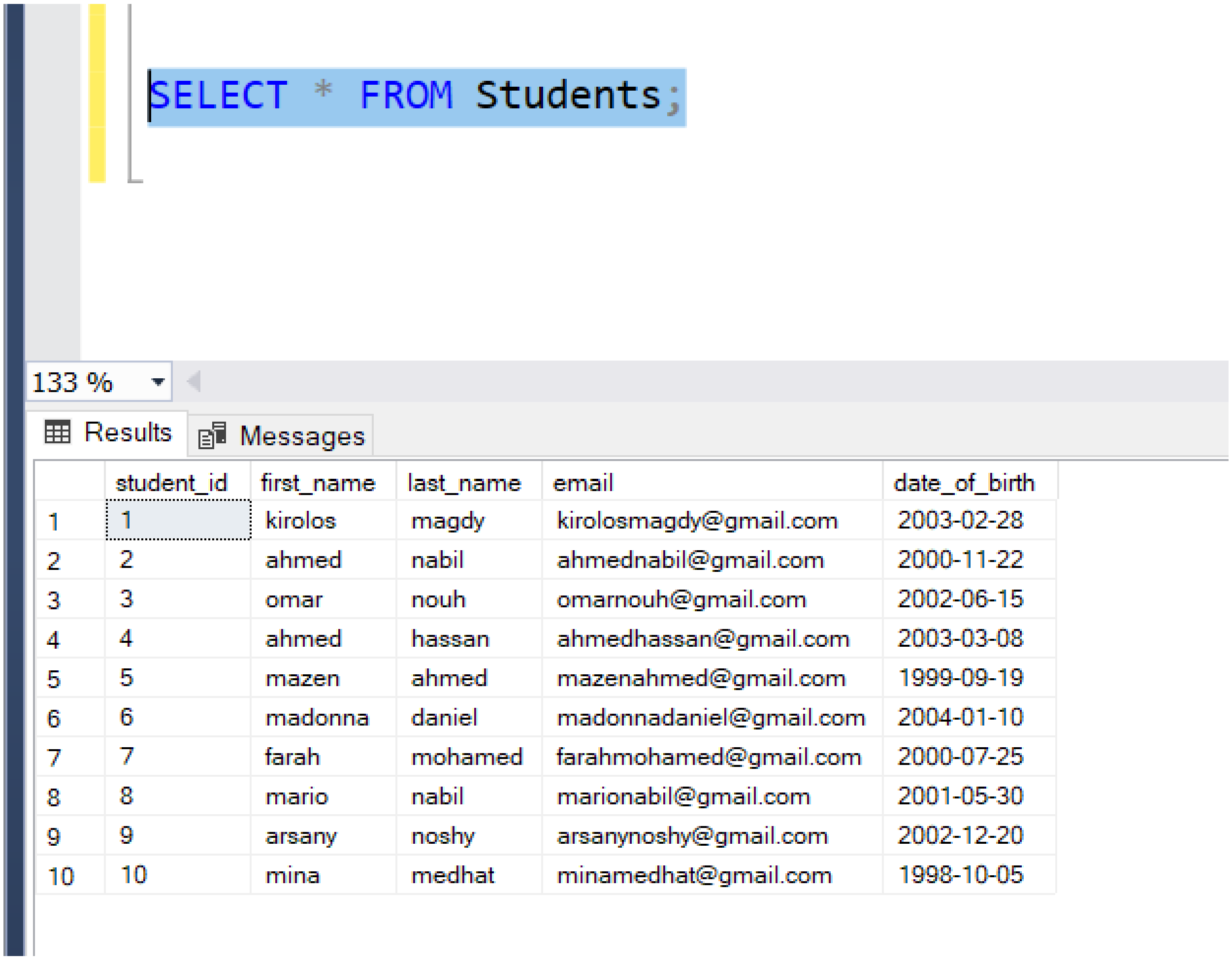
```sql
(3, 2, '2023-08-03'),
(4, 3, '2023-08-04'),
(5, 4, '2023-08-05'),
(6, 5, '2023-08-06'),
(7, 1, '2023-08-07'),
(8, 2, '2023-08-08'),
(9, 3, '2023-08-09'),
(10, 4, '2023-08-10'),
(1, 5, '2023-08-11'),
(2, 2, '2023-08-12'),
(3, 4, '2023-08-13'),
(4, 5, '2023-08-14'),
(5, 1, '2023-08-15');
```

4. Basic Queries:

1.Select all students.

query:

SELECT * FROM Students;



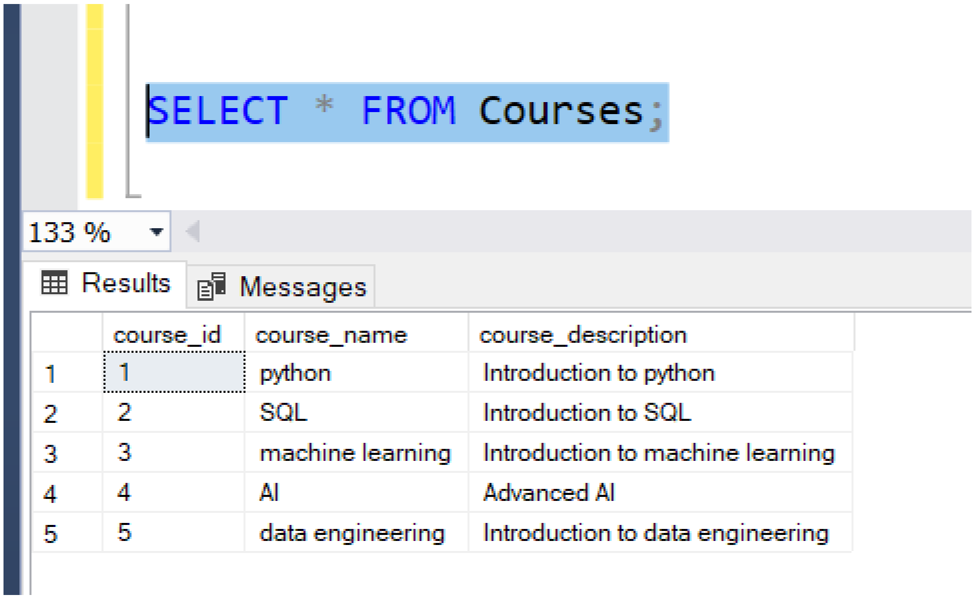
```
SELECT * FROM Students;
```

	student_id	first_name	last_name	email	date_of_birth
1	1	kirolos	magdy	kirolosmagdy@gmail.com	2003-02-28
2	2	ahmed	nabil	ahmednabil@gmail.com	2000-11-22
3	3	omar	nouh	omarnouh@gmail.com	2002-06-15
4	4	ahmed	hassan	ahmedhassan@gmail.com	2003-03-08
5	5	mazen	ahmed	mazenahmed@gmail.com	1999-09-19
6	6	madonna	daniel	madonnadaniel@gmail.com	2004-01-10
7	7	farah	mohamed	farahmohamed@gmail.com	2000-07-25
8	8	mario	nabil	marionabil@gmail.com	2001-05-30
9	9	arsany	noshy	arsanynoshy@gmail.com	2002-12-20
10	10	mina	medhat	minamedhat@gmail.com	1998-10-05

2.Select all courses.

query:

SELECT * FROM Courses;



```
SELECT * FROM Courses;
```

	course_id	course_name	course_description
1	1	python	Introduction to python
2	2	SQL	Introduction to SQL
3	3	machine learning	Introduction to machine learning
4	4	AI	Advanced AI
5	5	data engineering	Introduction to data engineering

3.Select all enrollments with student names and course names.

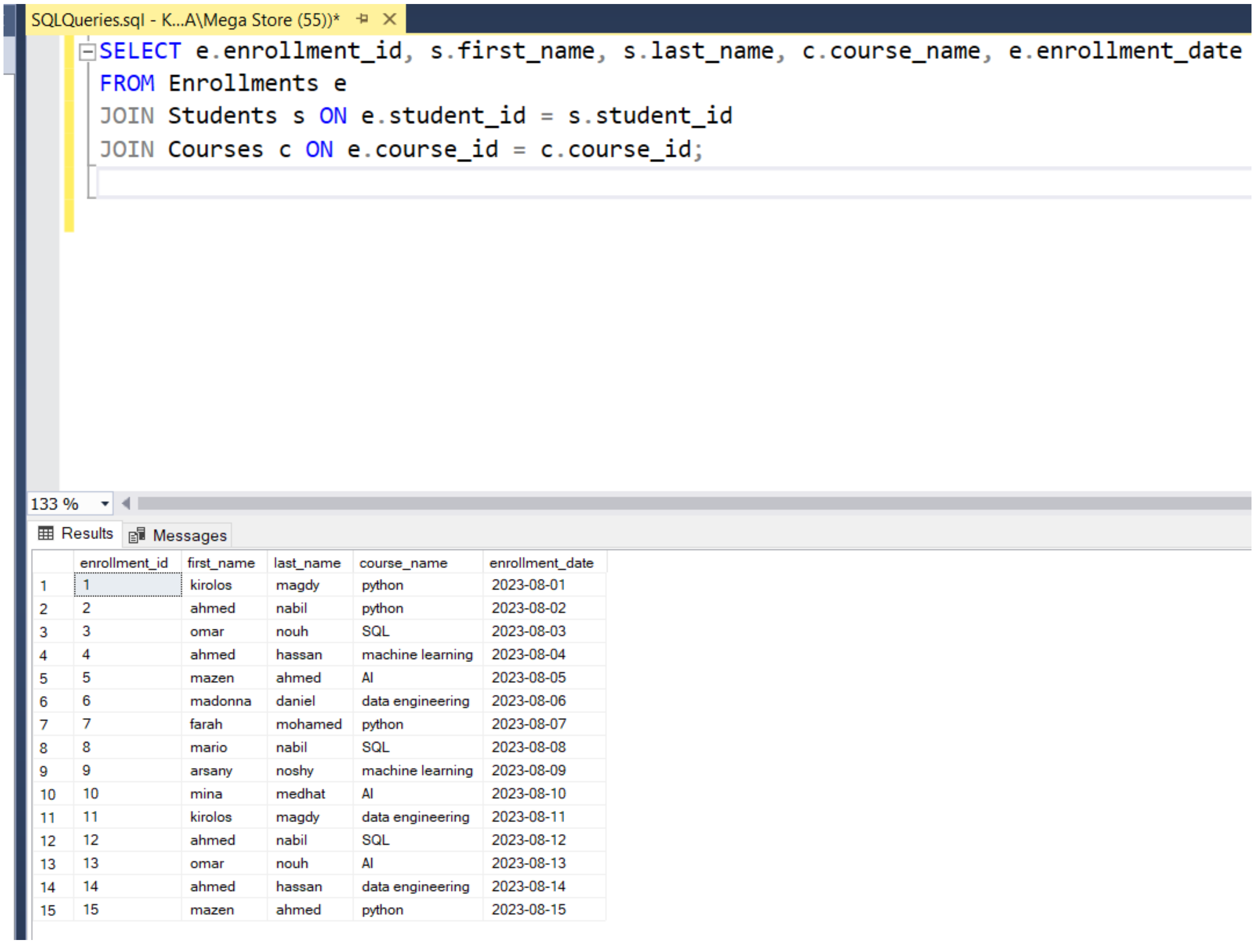
query:

SELECT e.enrollment_id, s.first_name, s.last_name, c.course_name, e.enrollment_date

FROM Enrollments e

JOIN Students s ON e.student_id = s.student_id

JOIN Courses c ON e.course_id = c.course_id;



```sql
SELECT e.enrollment_id, s.first_name, s.last_name, c.course_name, e.enrollment_date
FROM Enrollments e
JOIN Students s ON e.student_id = s.student_id
JOIN Courses c ON e.course_id = c.course_id;
```

	enrollment_id	first_name	last_name	course_name	enrollment_date
1	1	kirolos	magdy	python	2023-08-01
2	2	ahmed	nabil	python	2023-08-02
3	3	omar	nouh	SQL	2023-08-03
4	4	ahmed	hassan	machine learning	2023-08-04
5	5	mazen	ahmed	AI	2023-08-05
6	6	madonna	daniel	data engineering	2023-08-06
7	7	farah	mohamed	python	2023-08-07
8	8	mario	nabil	SQL	2023-08-08
9	9	arsany	noshy	machine learning	2023-08-09
10	10	mina	medhat	AI	2023-08-10
11	11	kirolos	magdy	data engineering	2023-08-11
12	12	ahmed	nabil	SQL	2023-08-12
13	13	omar	nouh	AI	2023-08-13
14	14	ahmed	hassan	data engineering	2023-08-14
15	15	mazen	ahmed	python	2023-08-15

5. Advanced Queries:

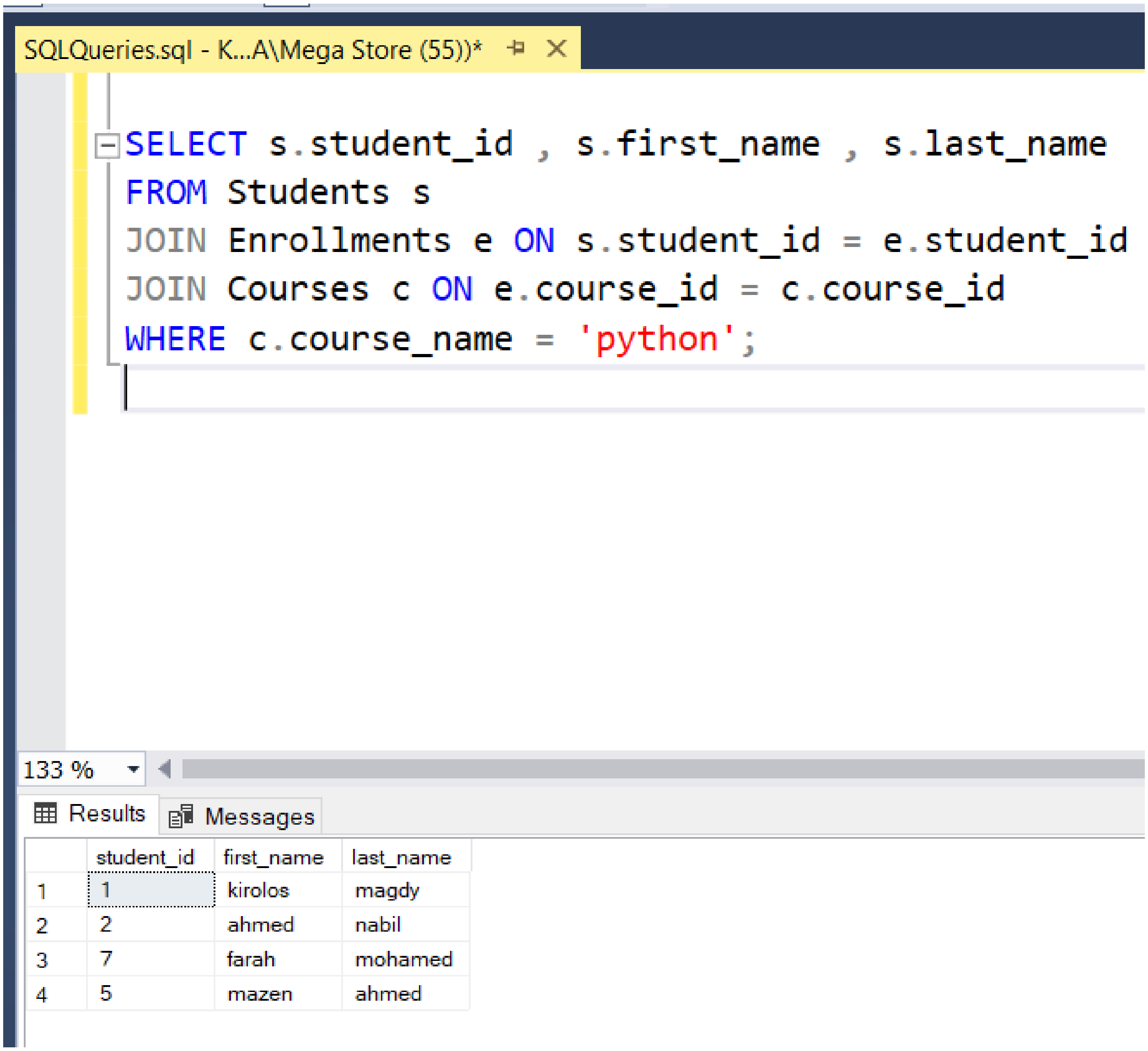
1.Select students who enrolled in a specific course.

query:

```sql
SELECT s.student_id , s.first_name , s.last_name
FROM Students s
JOIN Enrollments e ON s.student_id = e.student_id
```

JOIN Courses c ON e.course_id = c.course_id

WHERE c.course_name = 'python';



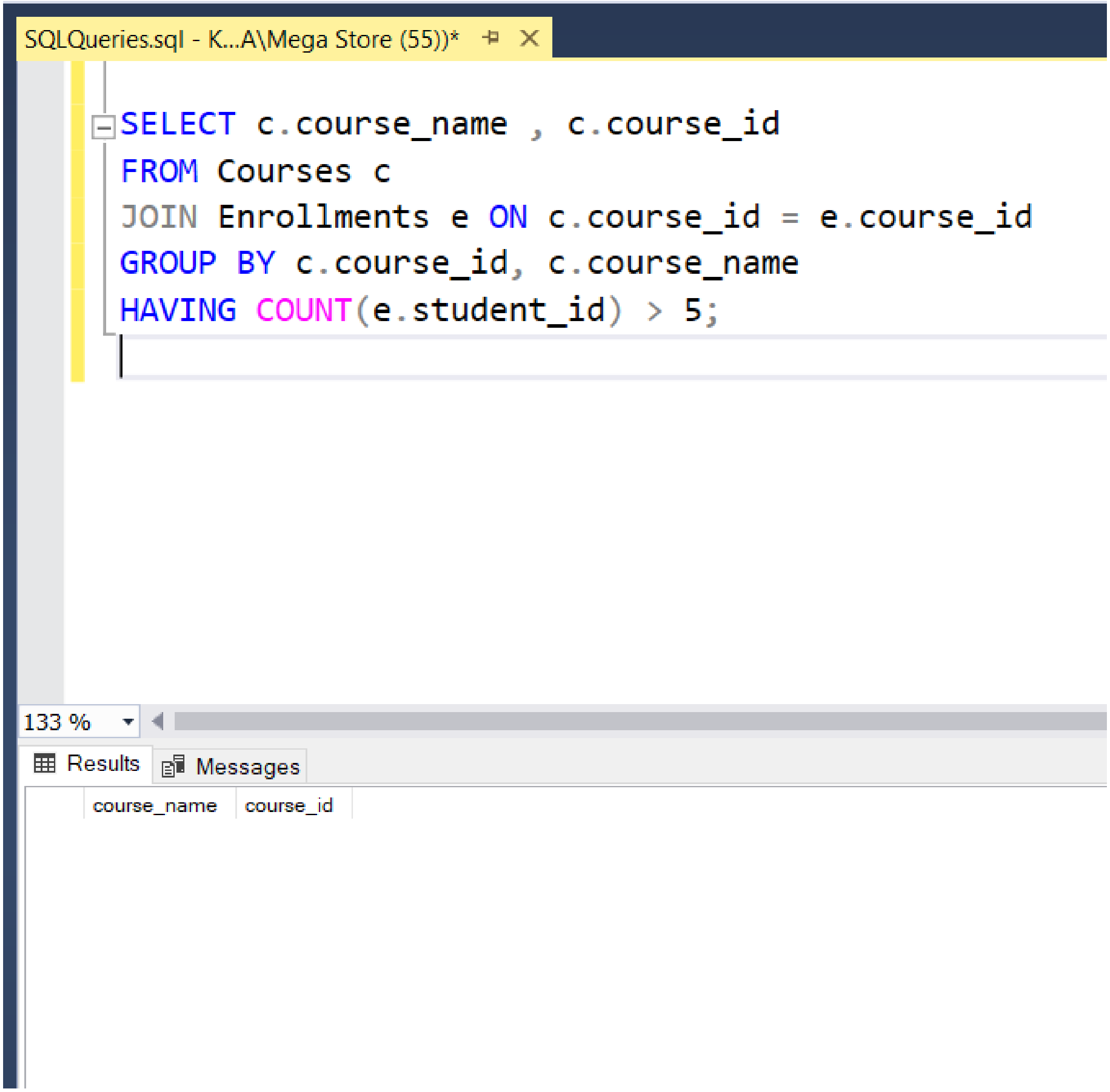
```
SELECT s.student_id , s.first_name , s.last_name
FROM Students s
JOIN Enrollments e ON s.student_id = e.student_id
JOIN Courses c ON e.course_id = c.course_id
WHERE c.course_name = 'python';
```

	student_id	first_name	last_name
1	1	kirolos	magdy
2	2	ahmed	nabil
3	7	farah	mohamed
4	5	mazen	ahmed

2.Select courses with more than 5 students.

query:

```
SELECT c.course_name , c.course_id
FROM Courses c
JOIN Enrollments e ON c.course_id = e.course_id
GROUP BY c.course_id, c.course_name
HAVING COUNT(e.student_id) > 5;
```



```
SELECT c.course_name , c.course_id
FROM Courses c
JOIN Enrollments e ON c.course_id = e.course_id
GROUP BY c.course_id, c.course_name
HAVING COUNT(e.student_id) > 5;
```

```
course_name   course_id
```

3.Update a student's email.

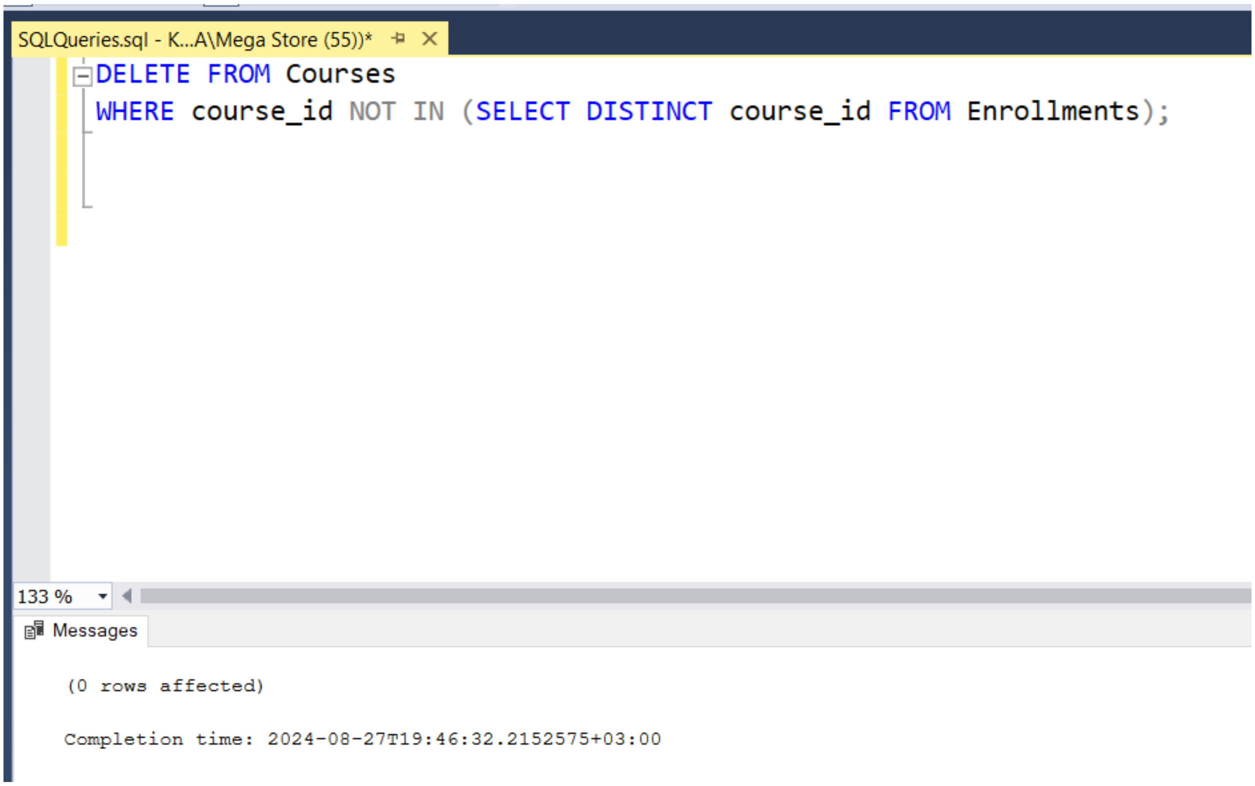
query:

```
UPDATE Students
SET email = 'kirolosmagdy10@gmail.com.com'
WHERE student_id = 1;
```

4.Delete a course that no students are enrolled in.

query:

```
DELETE FROM Courses
WHERE course_id NOT IN (SELECT DISTINCT course_id FROM Enrollments);
```



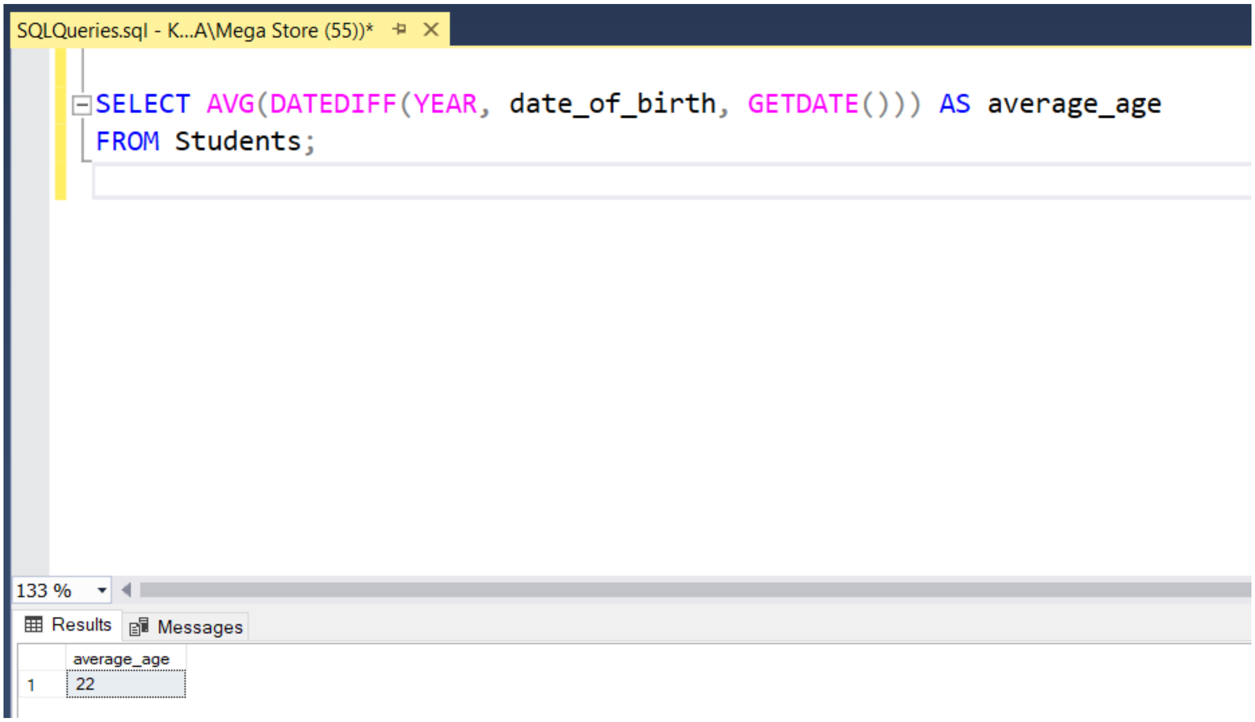
```sql
DELETE FROM Courses
WHERE course_id NOT IN (SELECT DISTINCT course_id FROM Enrollments);
```

```
(0 rows affected)

Completion time: 2024-08-27T19:46:32.2152575+03:00
```

5.Calculate the average age of students.

query:

```sql
SELECT AVG(DATEDIFF(YEAR, date_of_birth, GETDATE())) AS average_age
FROM Students;
```



```
SELECT AVG(DATEDIFF(YEAR, date_of_birth, GETDATE())) AS average_age
FROM Students;
```

```
   average_age
1  22
```

6.Find the course with the maximum enrollments.

query:

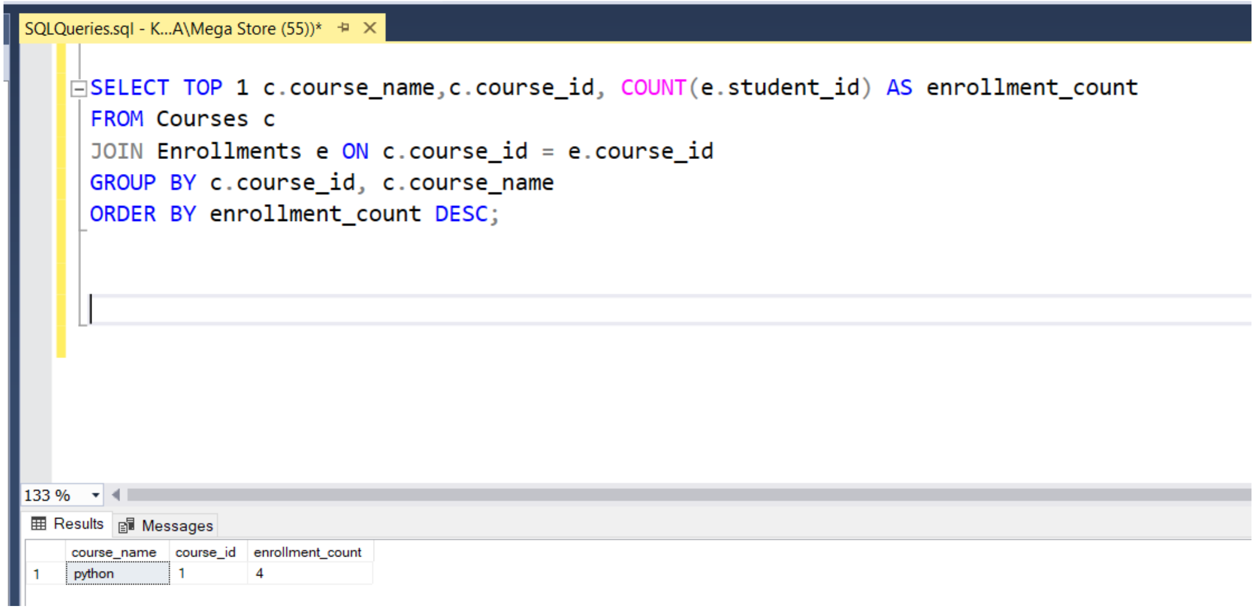
SELECT TOP 1 c.course_name,c.course_id, COUNT(e.student_id) AS enrollment_count

FROM Courses c

JOIN Enrollments e ON c.course_id = e.course_id

GROUP BY c.course_id, c.course_name

ORDER BY enrollment_count DESC;



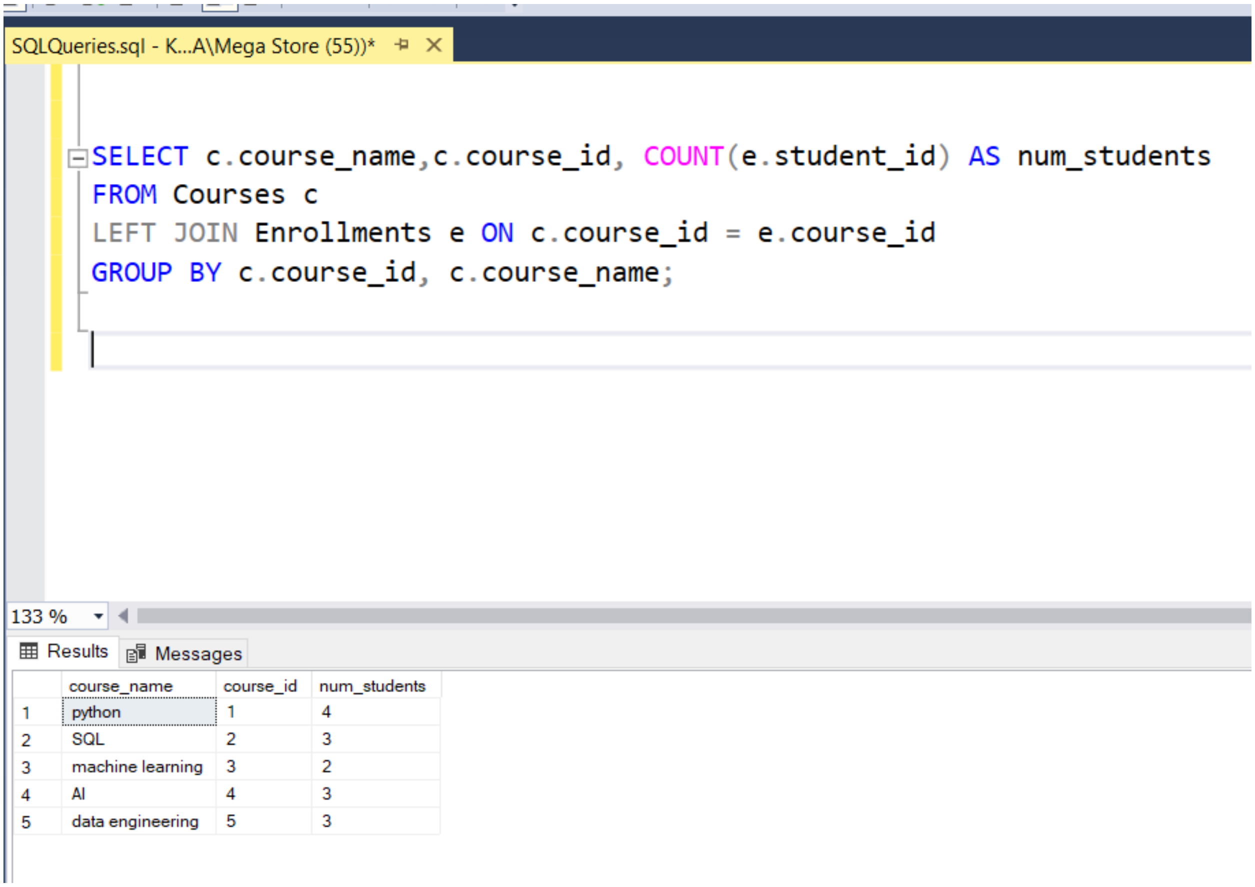
```
SELECT TOP 1 c.course_name,c.course_id, COUNT(e.student_id) AS enrollment_count
FROM Courses c
JOIN Enrollments e ON c.course_id = e.course_id
GROUP BY c.course_id, c.course_name
ORDER BY enrollment_count DESC;
```

	course_name	course_id	enrollment_count
1	python	1	4

7.List courses along with the number of students enrolled (use GROUP BY).

query:

```
SELECT c.course_name,c.course_id,
COUNT(e.student_id) AS num_students

FROM Courses c

LEFT JOIN Enrollments e ON c.course_id =
e.course_id

GROUP BY c.course_id, c.course_name;
```



```sql
SELECT c.course_name,c.course_id, COUNT(e.student_id) AS num_students
FROM Courses c
LEFT JOIN Enrollments e ON c.course_id = e.course_id
GROUP BY c.course_id, c.course_name;
```

	course_name	course_id	num_students
1	python	1	4
2	SQL	2	3
3	machine learning	3	2
4	AI	4	3
5	data engineering	5	3

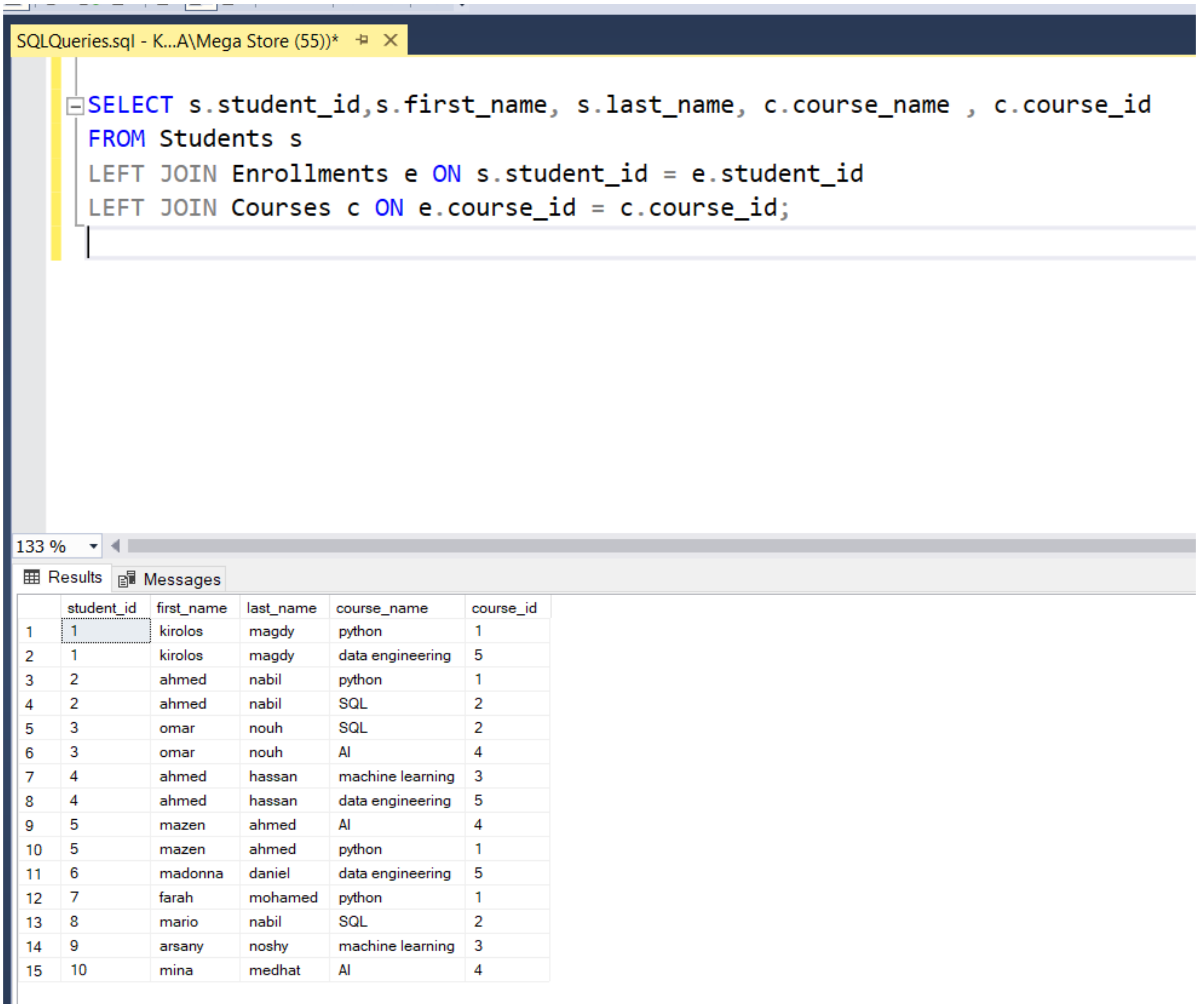
6. Join Queries:

1. Select all students with their enrolled courses (use JOIN).

query:

SELECT s.student_id,s.first_name, s.last_name, c.course_name , c.course_id

```
FROM Students s
LEFT JOIN Enrollments e ON s.student_id = e.student_id
LEFT JOIN Courses c ON e.course_id = c.course_id;
```



```
SELECT s.student_id,s.first_name, s.last_name, c.course_name , c.course_id
FROM Students s
LEFT JOIN Enrollments e ON s.student_id = e.student_id
LEFT JOIN Courses c ON e.course_id = c.course_id;
```

	student_id	first_name	last_name	course_name	course_id
1	1	kirolos	magdy	python	1
2	1	kirolos	magdy	data engineering	5
3	2	ahmed	nabil	python	1
4	2	ahmed	nabil	SQL	2
5	3	omar	nouh	SQL	2
6	3	omar	nouh	AI	4
7	4	ahmed	hassan	machine learning	3
8	4	ahmed	hassan	data engineering	5
9	5	mazen	ahmed	AI	4
10	5	mazen	ahmed	python	1
11	6	madonna	daniel	data engineering	5
12	7	farah	mohamed	python	1
13	8	mario	nabil	SQL	2
14	9	arsany	noshy	machine learning	3
15	10	mina	medhat	AI	4

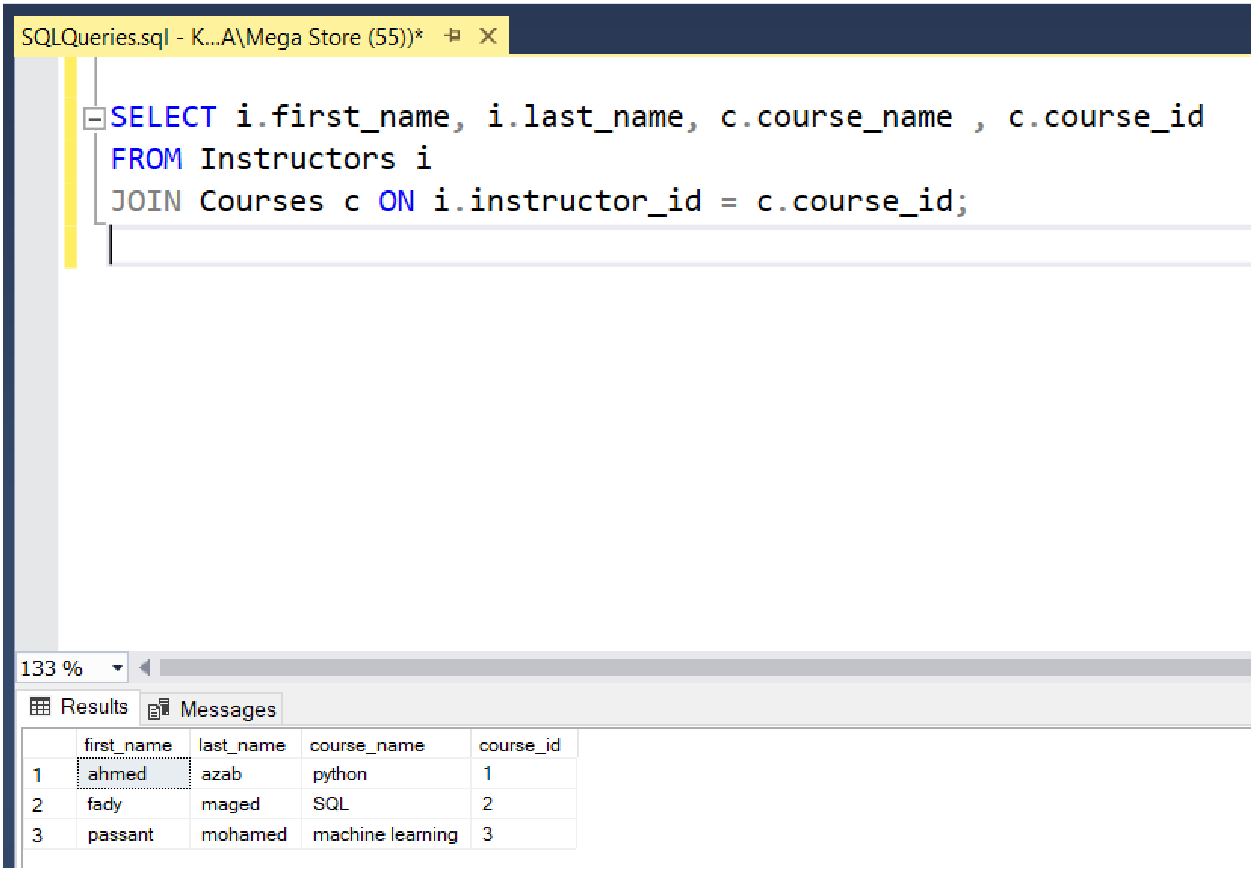
2.List all instructors and their courses.

query:

SELECT i.first_name, i.last_name, c.course_name , c.course_id

FROM Instructors i

JOIN Courses c ON i.instructor_id = c.course_id;



```
SELECT i.first_name, i.last_name, c.course_name , c.course_id
FROM Instructors i
JOIN Courses c ON i.instructor_id = c.course_id;
```

	first_name	last_name	course_name	course_id
1	ahmed	azab	python	1
2	fady	maged	SQL	2
3	passant	mohamed	machine learning	3

3.Find students who are not enrolled in any course.

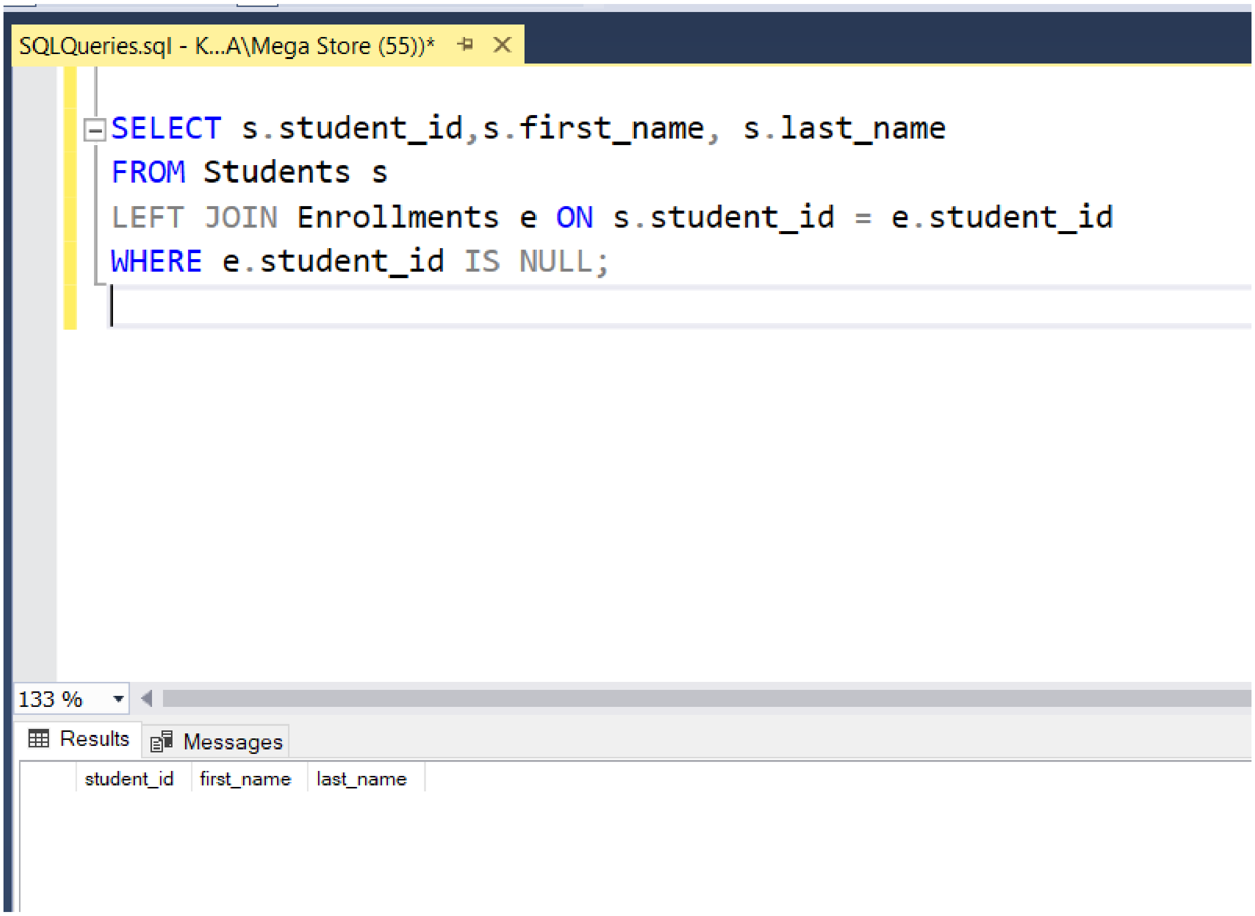
query:

SELECT s.student_id,s.first_name, s.last_name

FROM Students s

LEFT JOIN Enrollments e ON s.student_id = e.student_id

WHERE e.student_id IS NULL;



```
SELECT s.student_id,s.first_name, s.last_name
FROM Students s
LEFT JOIN Enrollments e ON s.student_id = e.student_id
WHERE e.student_id IS NULL;
```

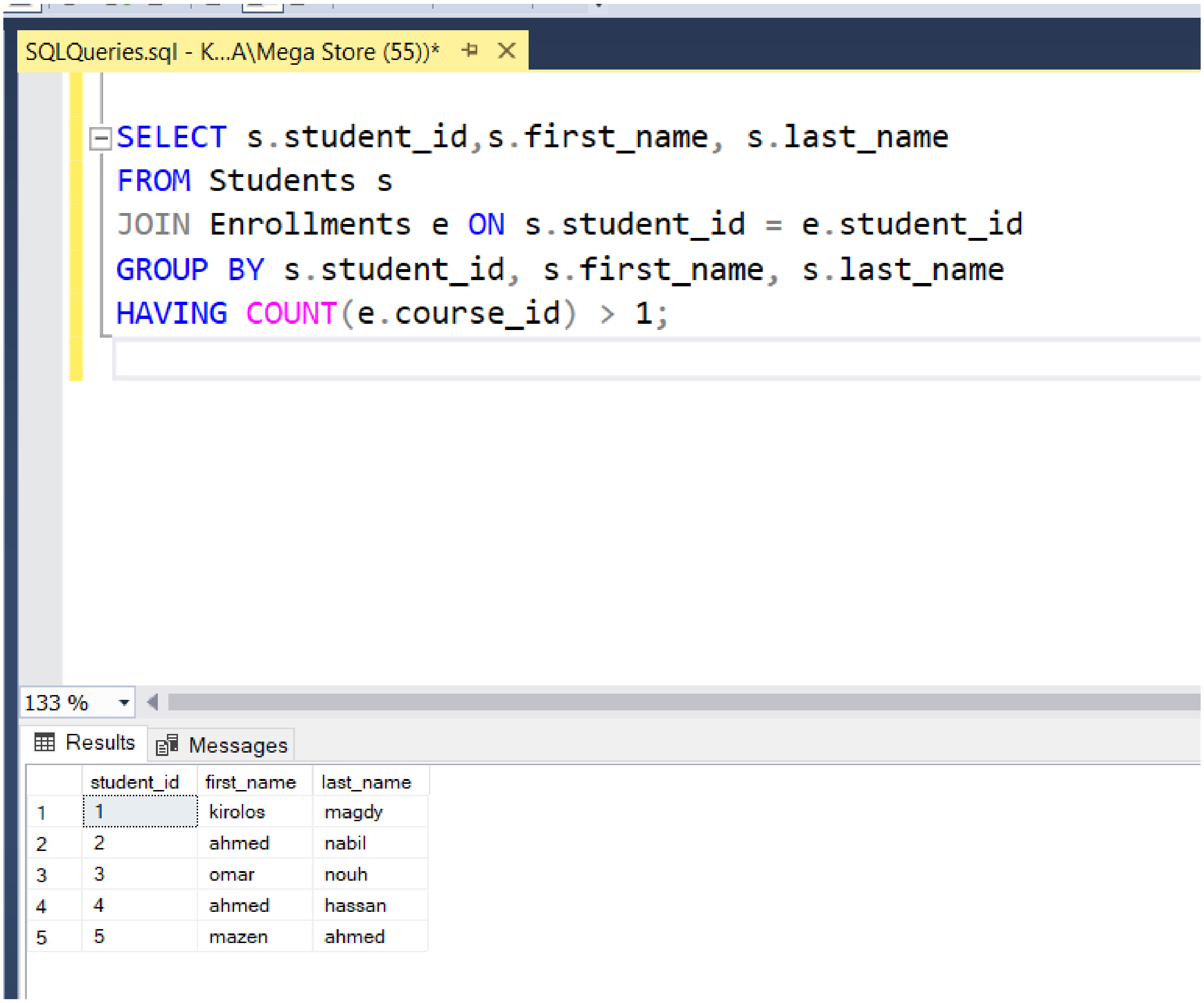
student_id	first_name	last_name

7. Subqueries and Set Operations:

1. Select students enrolled in more than one course.

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(e.course_id) > 1;
```



```
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(e.course_id) > 1;
```

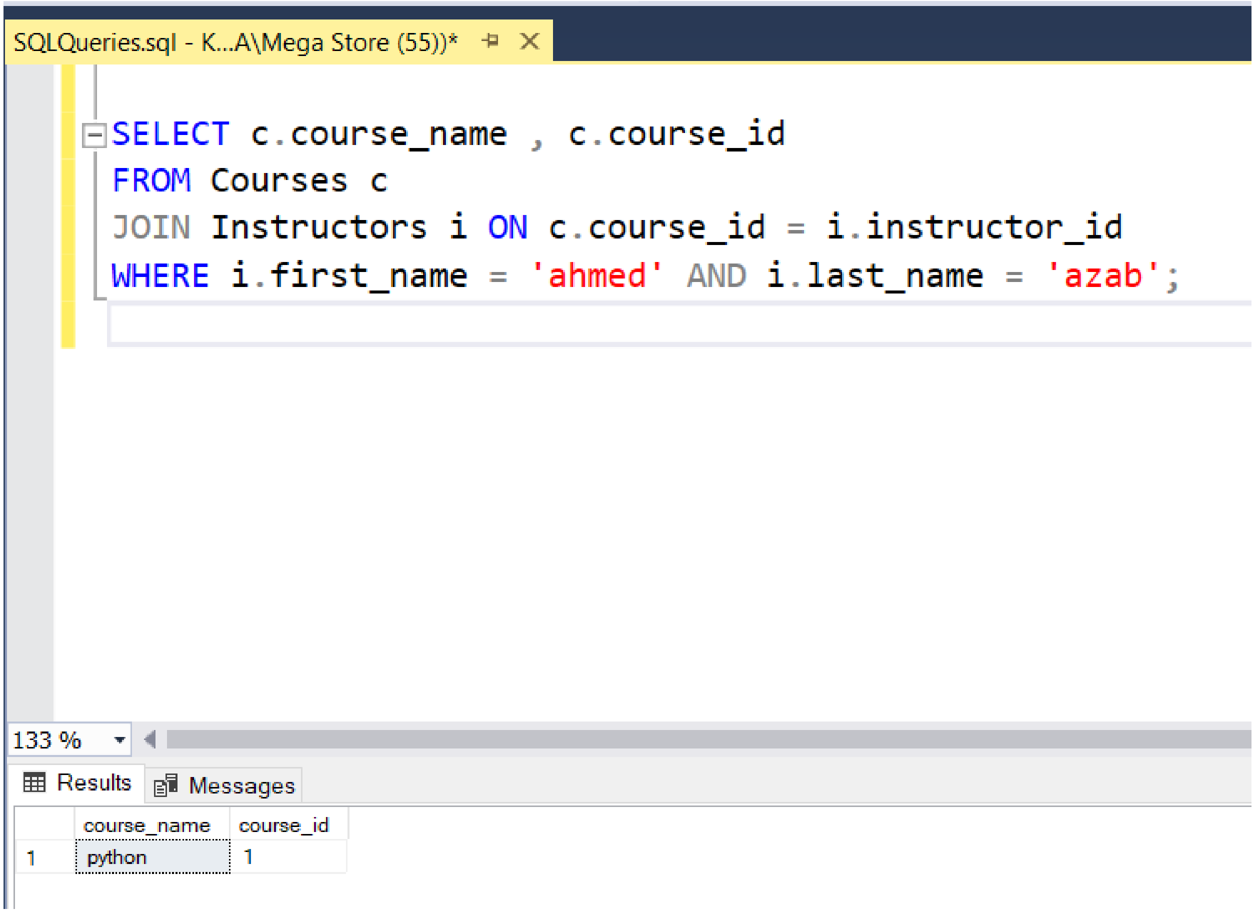
	student_id	first_name	last_name
1	1	kirolos	magdy
2	2	ahmed	nabil
3	3	omar	nouh
4	4	ahmed	hassan
5	5	mazen	ahmed

2.Find courses taught by a specific instructor.

query:

```
SELECT c.course_name , c.course_id
FROM Courses c
JOIN Instructors i ON c.course_id = i.instructor_id
```

WHERE i.first_name = 'ahmed' AND i.last_name = 'azab';



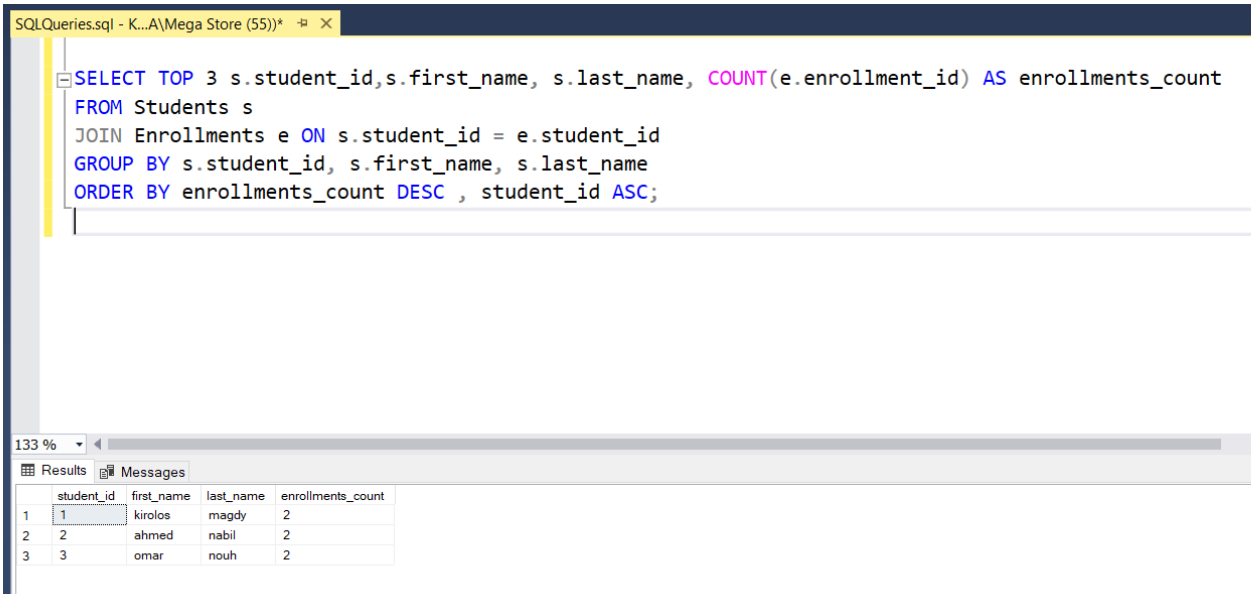
```
SELECT c.course_name , c.course_id
FROM Courses c
JOIN Instructors i ON c.course_id = i.instructor_id
WHERE i.first_name = 'ahmed' AND i.last_name = 'azab';
```

	course_name	course_id
1	python	1

3.Select the top 3 students with the most enrollments.

query:

```
SELECT TOP 3 s.student_id,s.first_name, s.last_name, COUNT(e.enrollment_id) AS enrollments_count
FROM Students s
JOIN Enrollments e ON s.student_id = e.student_id
GROUP BY s.student_id, s.first_name, s.last_name
ORDER BY enrollments_count DESC , student_id ASC;
```



```
SELECT TOP 3 s.student_id,s.first_name, s.last_name, COUNT(e.enrollment_id) AS enrollments_count
FROM Students s
JOIN Enrollments e ON s.student_id = e.student_id
GROUP BY s.student_id, s.first_name, s.last_name
ORDER BY enrollments_count DESC , student_id ASC;
```

	student_id	first_name	last_name	enrollments_count
1	1	kirolos	magdy	2
2	2	ahmed	nabil	2
3	3	omar	nouh	2

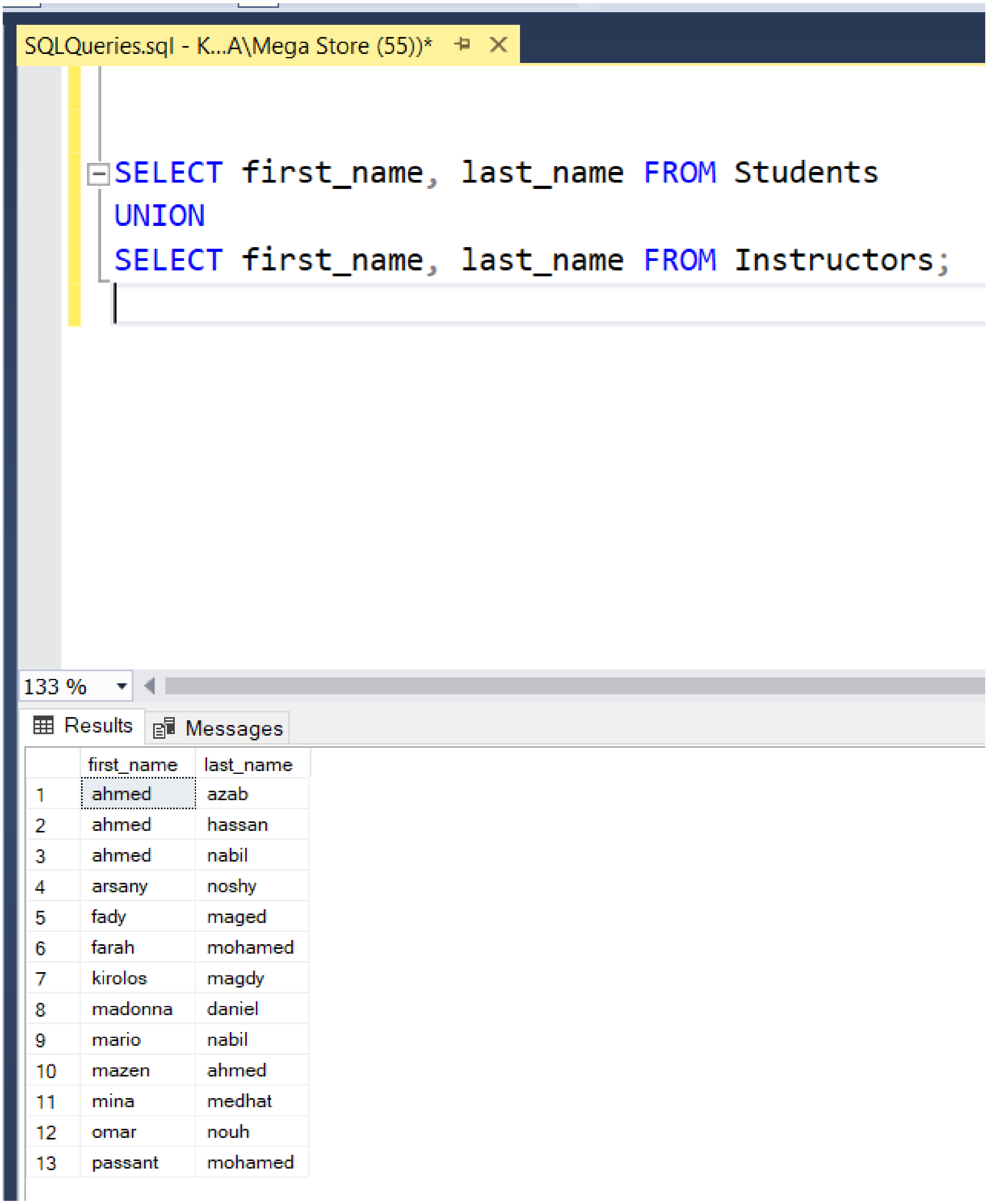
4.Use UNION to combine results of two different SELECT queries.

query:

```
SELECT first_name, last_name FROM Students
```

UNION

SELECT first_name, last_name FROM Instructors;



```sql
SELECT first_name, last_name FROM Students
UNION
SELECT first_name, last_name FROM Instructors;
```

	first_name	last_name
1	ahmed	azab
2	ahmed	hassan
3	ahmed	nabil
4	arsany	noshy
5	fady	maged
6	farah	mohamed
7	kirolos	magdy
8	madonna	daniel
9	mario	nabil
10	mazen	ahmed
11	mina	medhat
12	omar	nouh
13	passant	mohamed

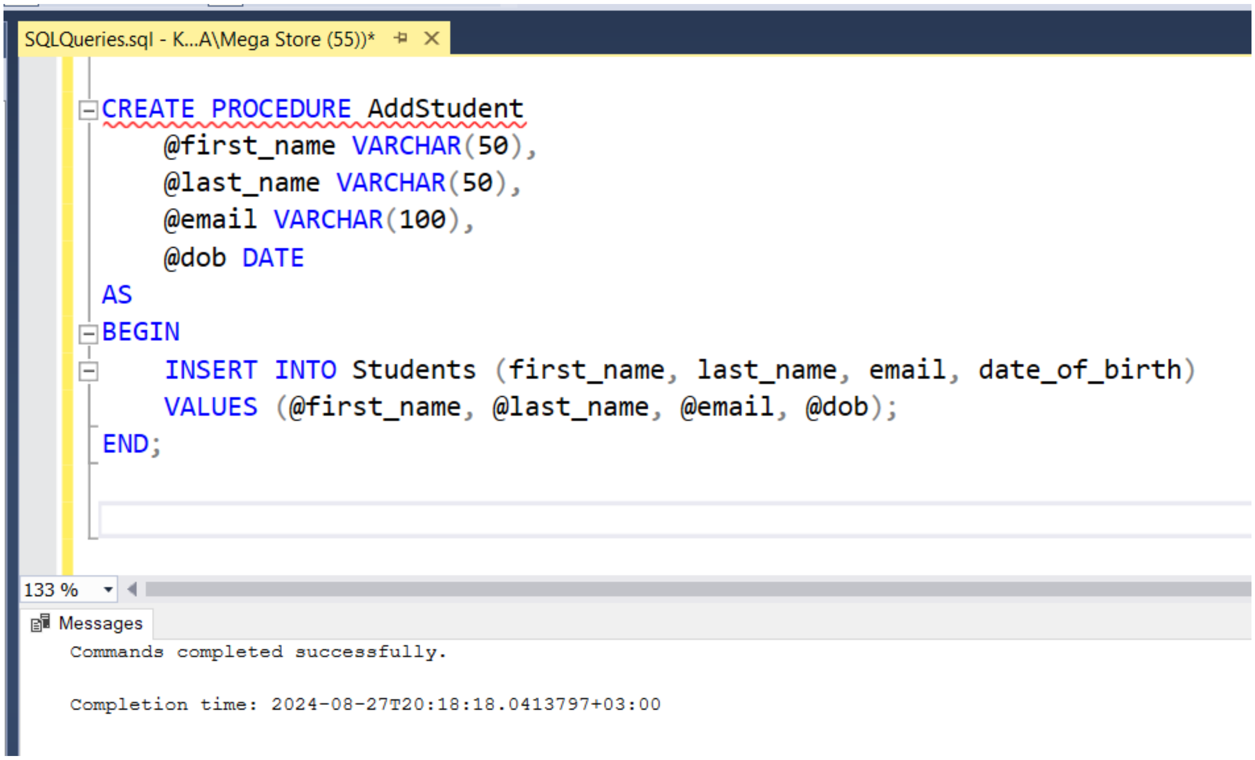
8.Functions and Stored Procedures:

1. Create a stored procedure to add a new student.

query:

```
CREATE PROCEDURE AddStudent
    @first_name VARCHAR(50),
    @last_name VARCHAR(50),
    @email VARCHAR(100),
    @dob DATE
AS
BEGIN
    INSERT INTO Students (first_name, last_name, email, date_of_birth)
    VALUES (@first_name, @last_name, @email, @dob);
```

END;



```
CREATE PROCEDURE AddStudent
    @first_name VARCHAR(50),
    @last_name VARCHAR(50),
    @email VARCHAR(100),
    @dob DATE
AS
BEGIN
    INSERT INTO Students (first_name, last_name, email, date_of_birth)
    VALUES (@first_name, @last_name, @email, @dob);
END;
```

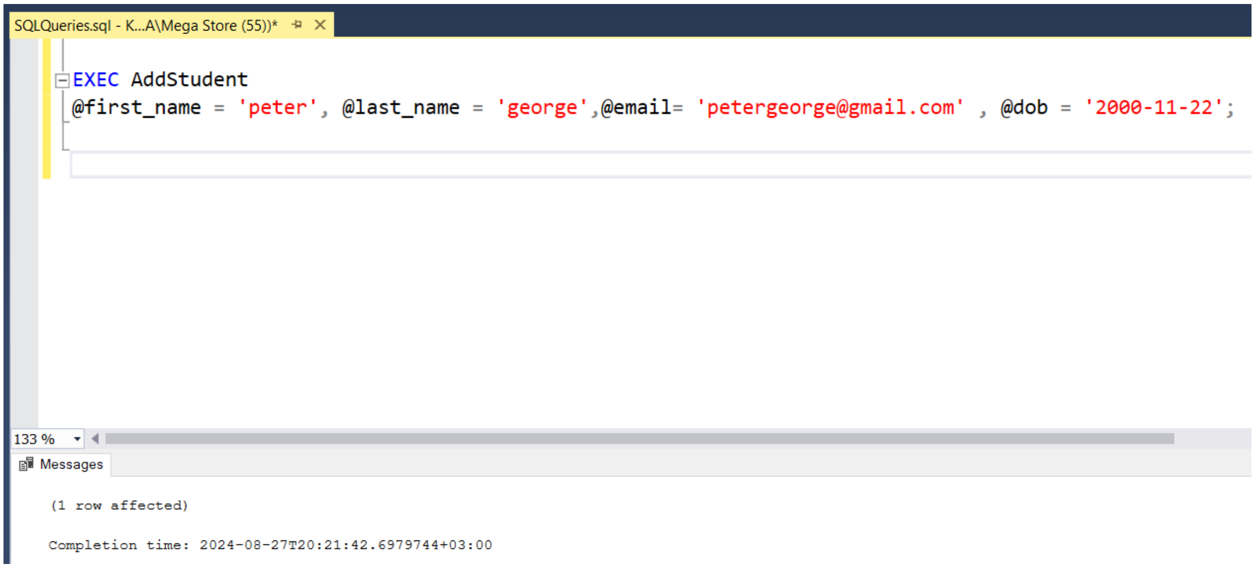
```
Commands completed successfully.

Completion time: 2024-08-27T20:18:18.0413797+03:00
```

Test the procedcure :

Query:

EXEC AddStudent

@first_name = 'peter', @last_name = 'george',@email= 'petergeorge@gmail.com' , @dob = '2000-11-22';



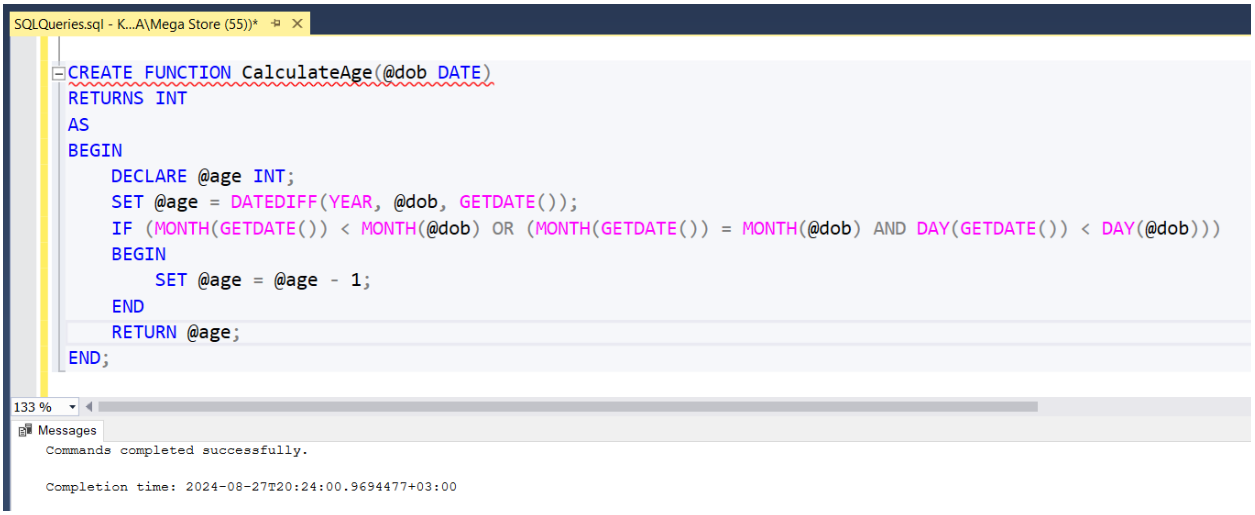
```
EXEC AddStudent
@first_name = 'peter', @last_name = 'george',@email= 'petergeorge@gmail.com' , @dob = '2000-11-22';
```

```
(1 row affected)

Completion time: 2024-08-27T20:21:42.6979744+03:00
```

2.Create a function to calculate the age of a student based on their date of birth.

query:

```
CREATE FUNCTION CalculateAge(@dob DATE)
RETURNS INT
AS
BEGIN
    DECLARE @age INT;
    SET @age = DATEDIFF(YEAR, @dob, GETDATE());
    IF (MONTH(GETDATE()) < MONTH(@dob) OR (MONTH(GETDATE()) = MONTH(@dob) AND DAY(GETDATE()) < DAY(@dob)))
    BEGIN
        SET @age = @age - 1;
    END
    RETURN @age;
END;
```



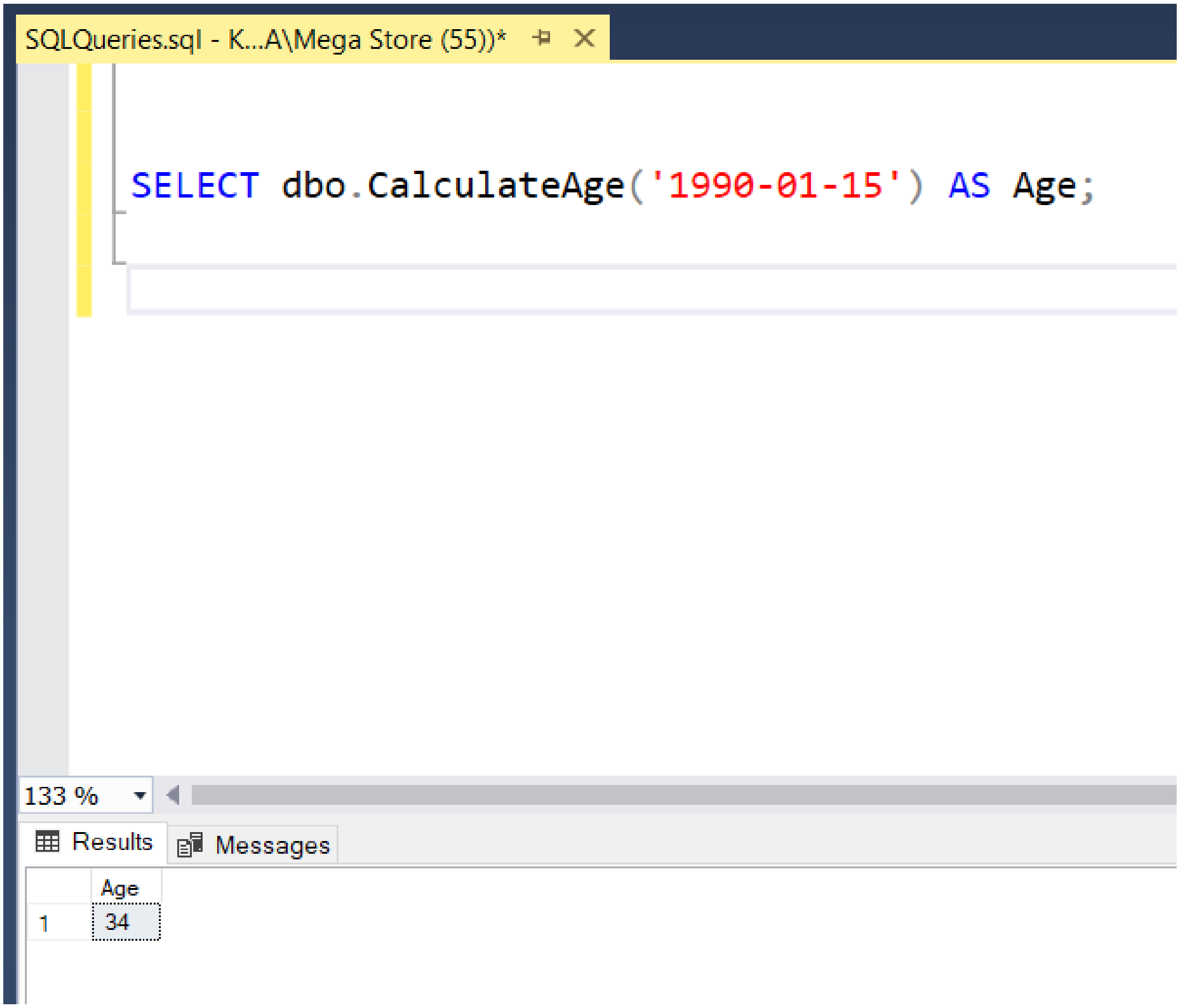
```
CREATE FUNCTION CalculateAge(@dob DATE)
RETURNS INT
AS
BEGIN
    DECLARE @age INT;
    SET @age = DATEDIFF(YEAR, @dob, GETDATE());
    IF (MONTH(GETDATE()) < MONTH(@dob) OR (MONTH(GETDATE()) = MONTH(@dob) AND DAY(GETDATE()) < DAY(@dob)))
    BEGIN
        SET @age = @age - 1;
    END
    RETURN @age;
END;
```

```
Commands completed successfully.

Completion time: 2024-08-27T20:24:00.9694477+03:00
```

Test the function :

Query :

SELECT dbo.CalculateAge('1990-01-15') AS Age;



```sql
SELECT dbo.CalculateAge('1990-01-15') AS Age;
```

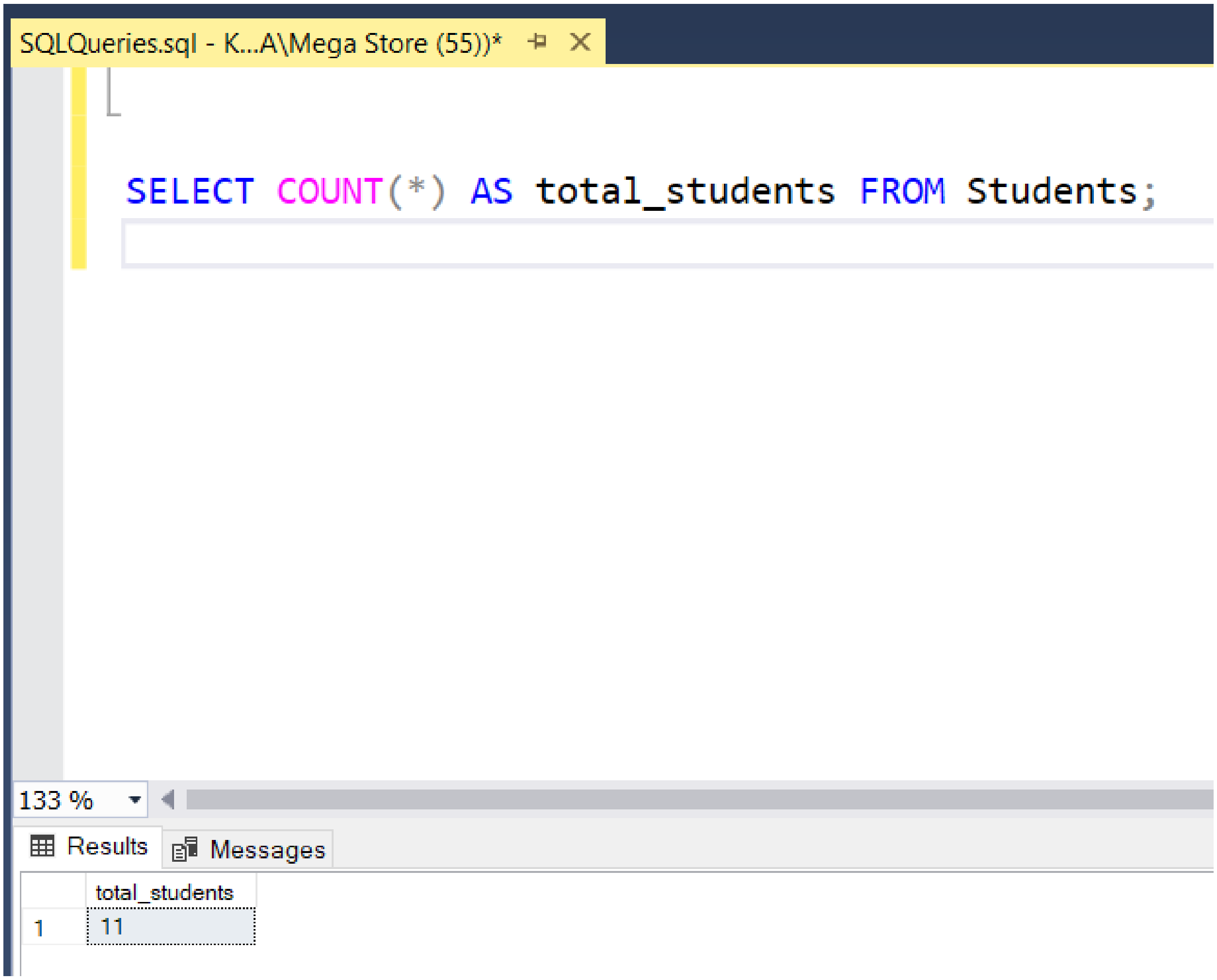
	Age
1	34

9. Aggregate Functions and Grouping:

1. Calculate the total number of students.

query:

```
SELECT COUNT(*) AS total_students FROM
Students;
```



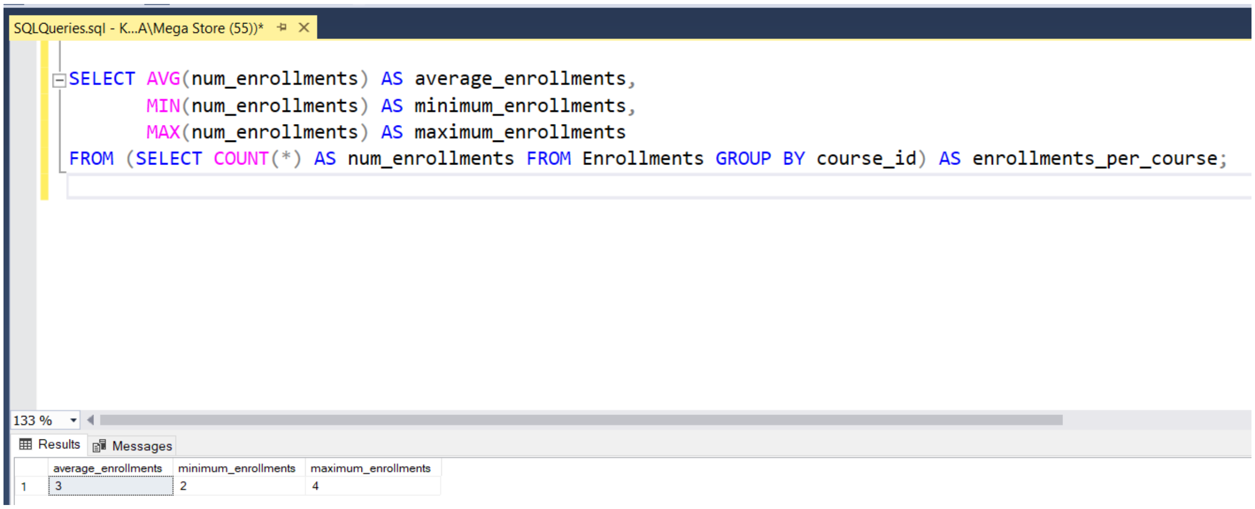
```
SELECT COUNT(*) AS total_students FROM Students;
```

```
   total_students
1  11
```

2.Calculate the average, minimum, and maximum number of enrollments per course.

query:

```
SELECT
        AVG(num_enrollments) AS
average_enrollments,
        MIN(num_enrollments) AS minimum_enrollments,
        MAX(num_enrollments) AS maximum_enrollments
FROM (SELECT COUNT(*) AS num_enrollments FROM Enrollments GROUP BY course_id) AS enrollments_per_course;
```



```
SELECT AVG(num_enrollments) AS average_enrollments,
       MIN(num_enrollments) AS minimum_enrollments,
       MAX(num_enrollments) AS maximum_enrollments
FROM (SELECT COUNT(*) AS num_enrollments FROM Enrollments GROUP BY course_id) AS enrollments_per_course;
```

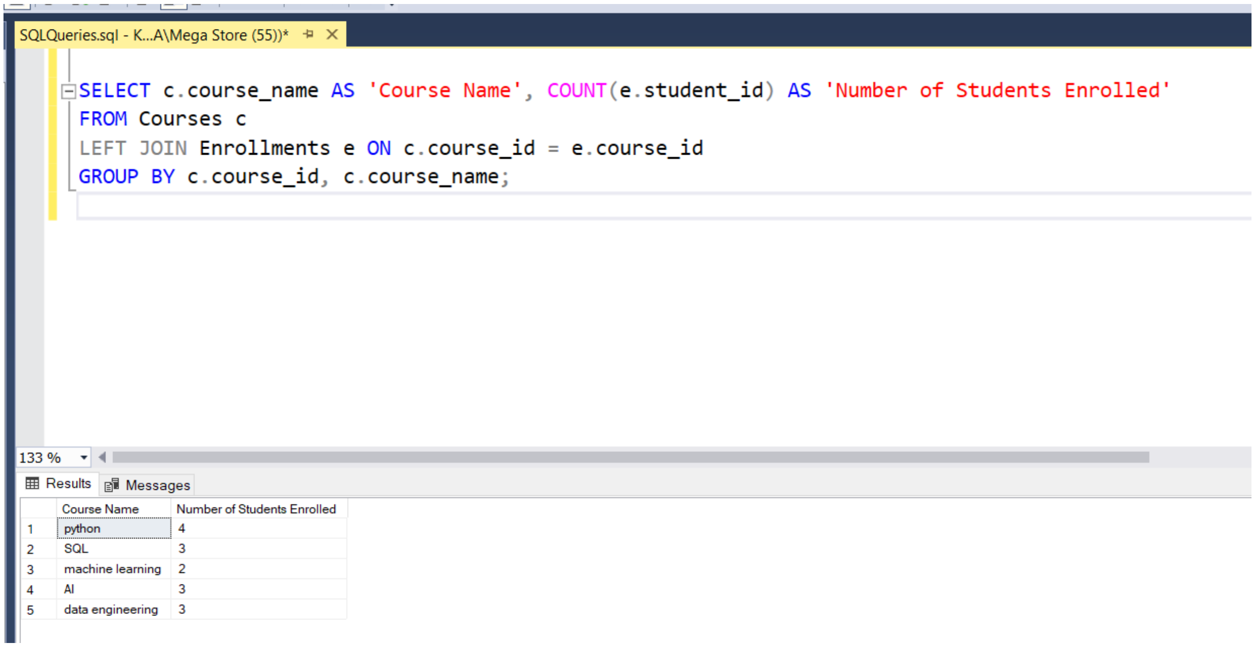
	average_enrollments	minimum_enrollments	maximum_enrollments
1	3	2	4

10. Additional Tasks:

1. Create aliases for complex column names.

query:

```
SELECT c.course_name AS 'Course Name', COUNT(e.student_id) AS 'Number of Students Enrolled'
FROM Courses c
LEFT JOIN Enrollments e ON c.course_id = e.course_id
GROUP BY c.course_id, c.course_name;
```



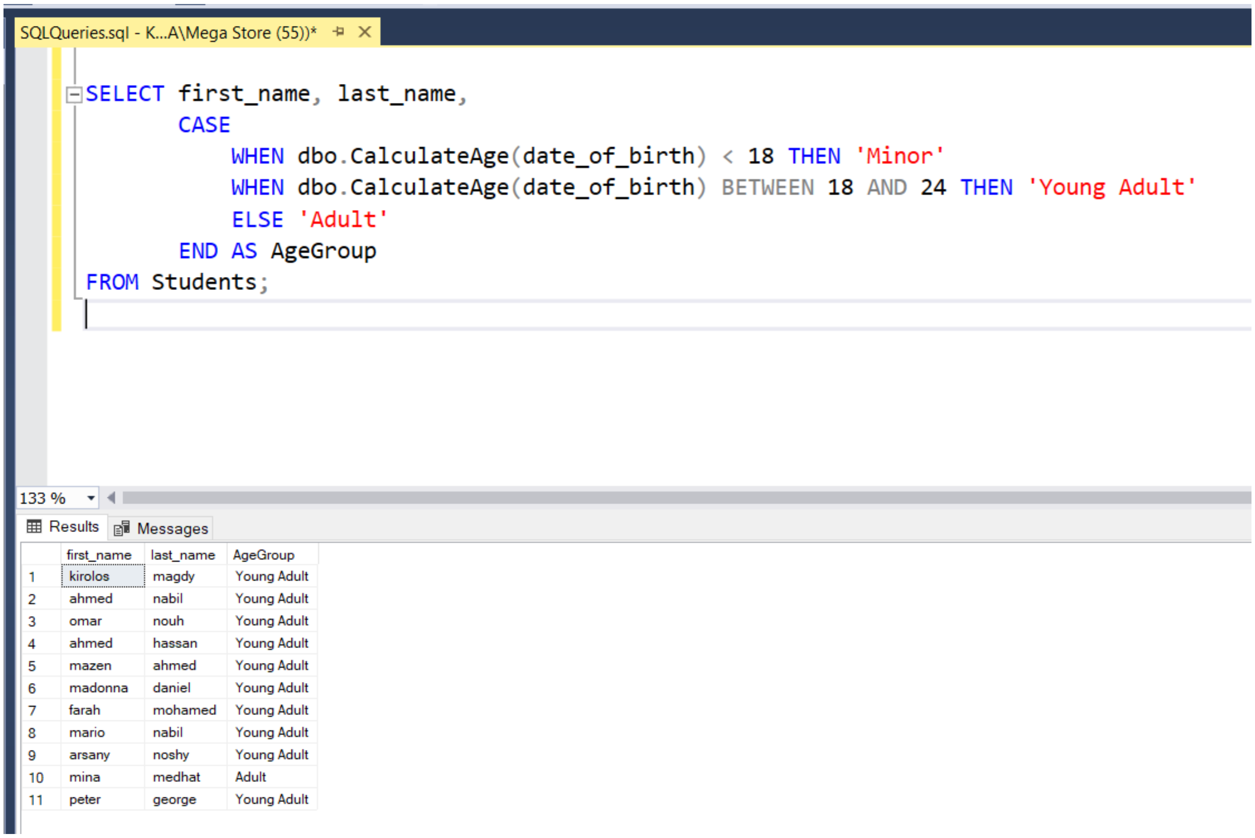
```
SELECT c.course_name AS 'Course Name', COUNT(e.student_id) AS 'Number of Students Enrolled'
FROM Courses c
LEFT JOIN Enrollments e ON c.course_id = e.course_id
GROUP BY c.course_id, c.course_name;
```

	Course Name	Number of Students Enrolled
1	python	4
2	SQL	3
3	machine learning	2
4	AI	3
5	data engineering	3

2.Use CASE to categorize students based on their age.

query:

```
SELECT first_name, last_name,
       CASE
           WHEN dbo.CalculateAge(date_of_birth) < 18 THEN 'Minor'
           WHEN dbo.CalculateAge(date_of_birth) BETWEEN 18 AND 24 THEN 'Young Adult'
           ELSE 'Adult'
       END AS AgeGroup
FROM Students;
```



```
SELECT first_name, last_name,
       CASE
           WHEN dbo.CalculateAge(date_of_birth) < 18 THEN 'Minor'
           WHEN dbo.CalculateAge(date_of_birth) BETWEEN 18 AND 24 THEN 'Young Adult'
           ELSE 'Adult'
       END AS AgeGroup
FROM Students;
```

	first_name	last_name	AgeGroup
1	kirolos	magdy	Young Adult
2	ahmed	nabil	Young Adult
3	omar	nouh	Young Adult
4	ahmed	hassan	Young Adult
5	mazen	ahmed	Young Adult
6	madonna	daniel	Young Adult
7	farah	mohamed	Young Adult
8	mario	nabil	Young Adult
9	arsany	noshy	Young Adult
10	mina	medhat	Adult
11	peter	george	Young Adult

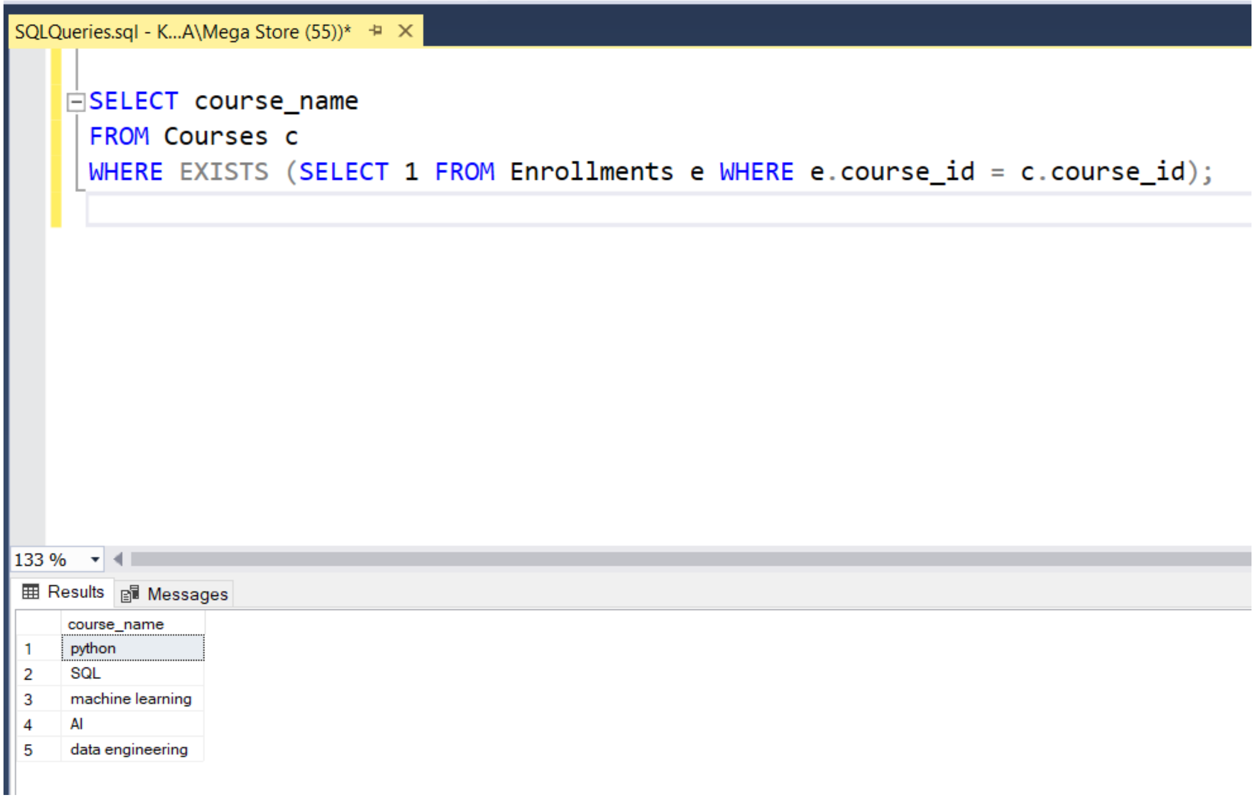
3.Use EXISTS to find courses with at least one enrolled student.

query:

SELECT course_name

FROM Courses c

WHERE EXISTS (SELECT 1 FROM Enrollments e WHERE e.course_id = c.course_id);



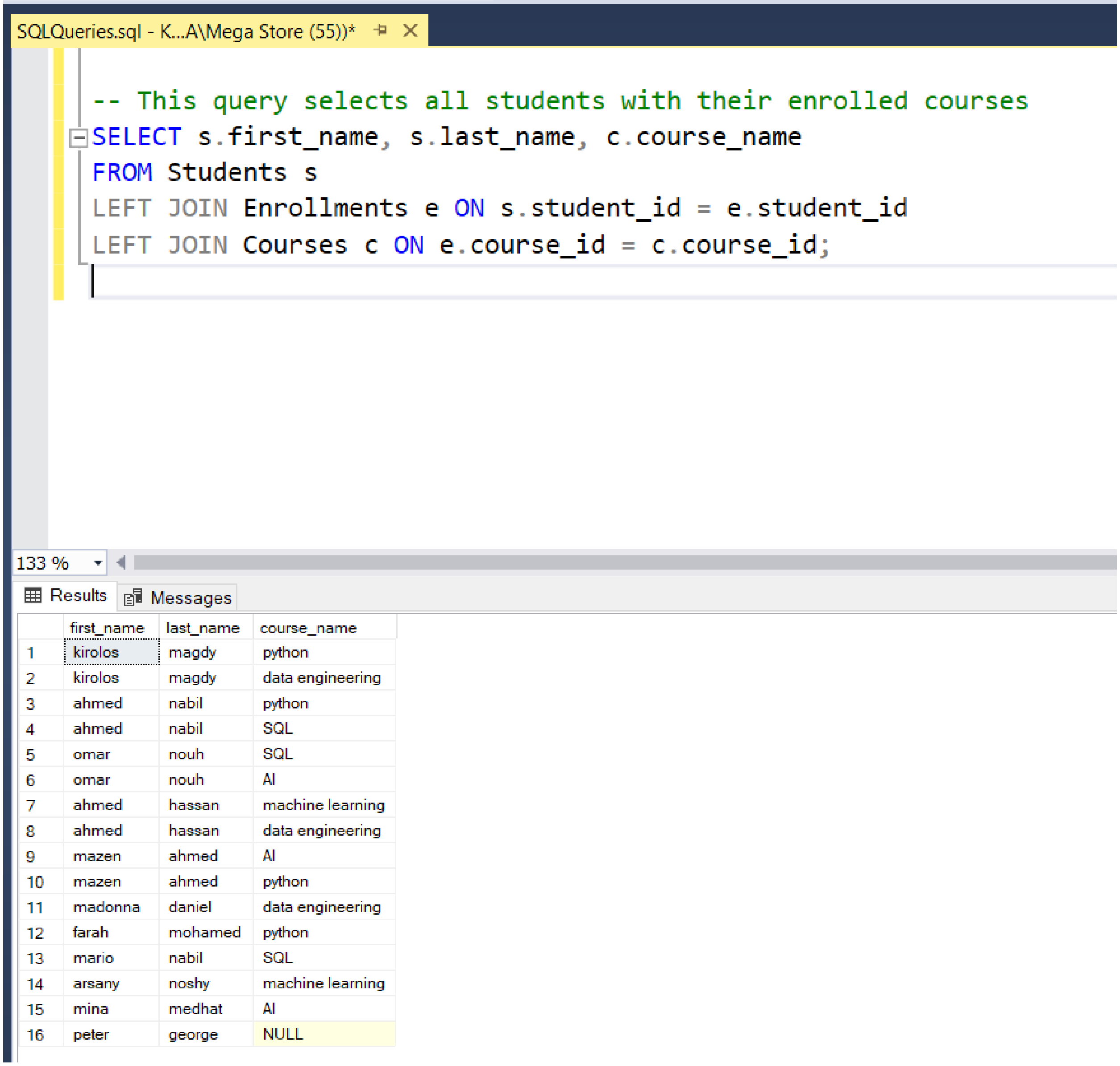
```
SELECT course_name
FROM Courses c
WHERE EXISTS (SELECT 1 FROM Enrollments e WHERE e.course_id = c.course_id);
```

	course_name
1	python
2	SQL
3	machine learning
4	AI
5	data engineering

4.Create comments in SQL for clarity.

query:

```
-- This query selects all students with their enrolled courses
SELECT s.first_name, s.last_name, c.course_name
FROM Students s
LEFT JOIN Enrollments e ON s.student_id = e.student_id
LEFT JOIN Courses c ON e.course_id = c.course_id;
```



```sql
-- This query selects all students with their enrolled courses
SELECT s.first_name, s.last_name, c.course_name
FROM Students s
LEFT JOIN Enrollments e ON s.student_id = e.student_id
LEFT JOIN Courses c ON e.course_id = c.course_id;
```

	first_name	last_name	course_name
1	kirolos	magdy	python
2	kirolos	magdy	data engineering
3	ahmed	nabil	python
4	ahmed	nabil	SQL
5	omar	nouh	SQL
6	omar	nouh	AI
7	ahmed	hassan	machine learning
8	ahmed	hassan	data engineering
9	mazen	ahmed	AI
10	mazen	ahmed	python
11	madonna	daniel	data engineering
12	farah	mohamed	python
13	mario	nabil	SQL
14	arsany	noshy	machine learning
15	mina	medhat	AI
16	peter	george	NULL