HEXAWARE ASSIGNEMNT

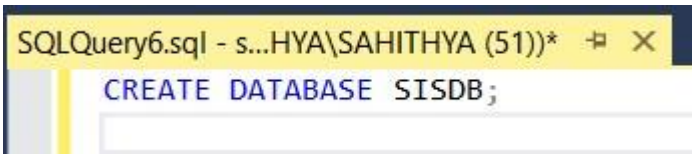
DONE BY – AMBATI SESHA SAI SAHITHYA

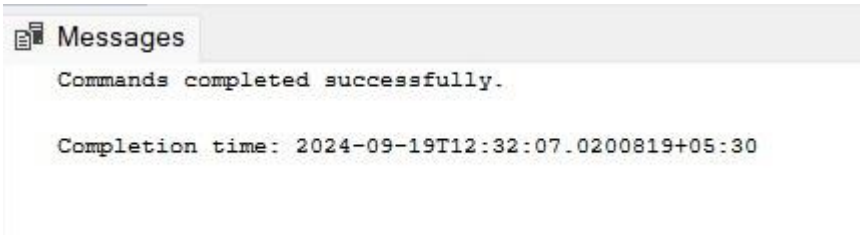
ASSIGNMENT – 2

Github sql file link :
https://github.com/sahithya007/HEXAWARE_ASSIGNMENT/blob/main/SQLQuery6_SISDB_ASSIGNMENT.sql

TASK- 1

1. Create the database named "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

```
CREATE TABLE Students (
    student_id INT PRIMARY KEY IDENTITY(1,1),
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    date_of_birth DATE,
    email VARCHAR(100) UNIQUE,
    phone_number VARCHAR(15)
);
```

b. Courses

```
CREATE TABLE Courses (
    course_id INT PRIMARY KEY IDENTITY(1,1),
    course_name VARCHAR(100),
    credits INT,
    teacher_id INT FOREIGN KEY REFERENCES Teacher(teacher_id)
);
```

c. Enrollments

```
CREATE TABLE Enrollments (
    enrollment_id INT PRIMARY KEY IDENTITY(1,1),
    student_id INT FOREIGN KEY REFERENCES Students(student_id),
    course_id INT FOREIGN KEY REFERENCES Courses(course_id),
    enrollment_date DATE
);
```

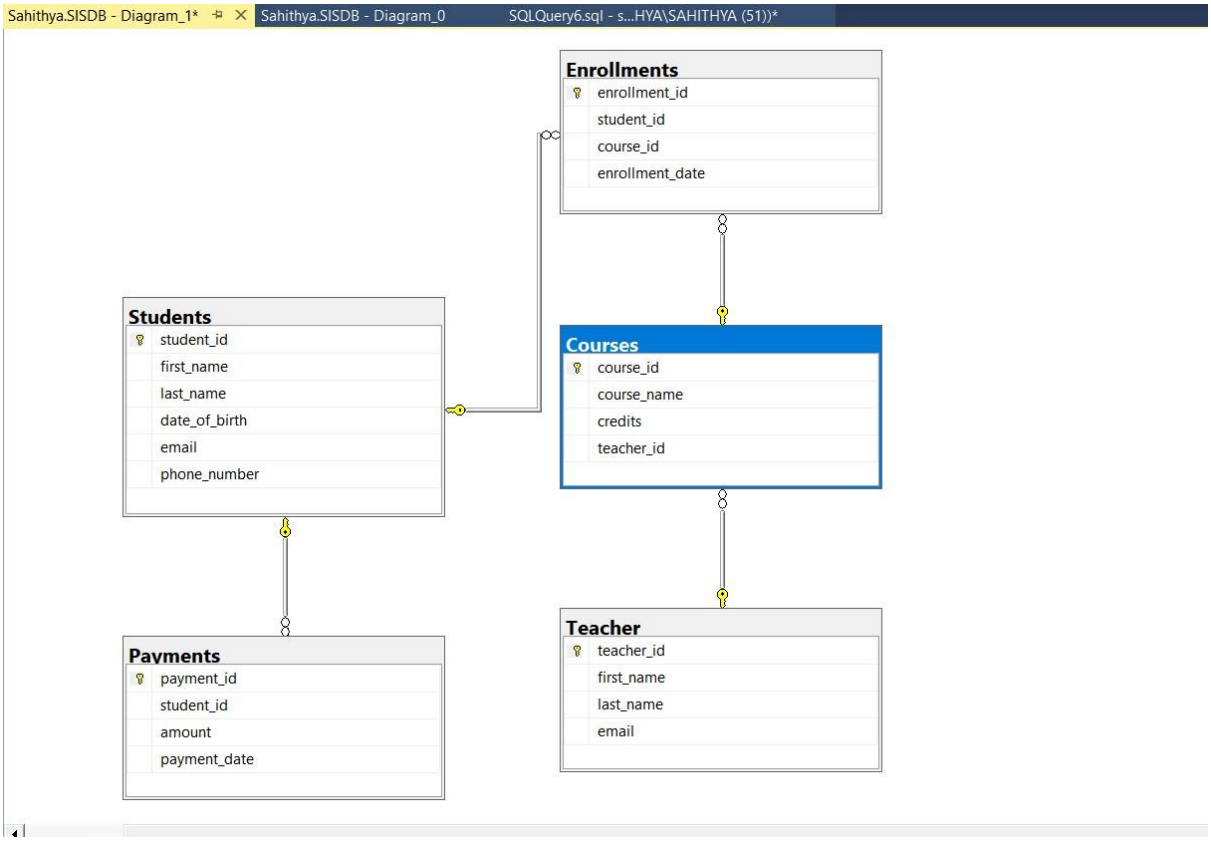
d. Teacher

```
CREATE TABLE Teacher (
    teacher_id INT PRIMARY KEY IDENTITY(1,1),
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    email VARCHAR(100) UNIQUE
);
```

e. Payments

```
CREATE TABLE Payments (
    payment_id INT PRIMARY KEY IDENTITY(1,1),
    student_id INT FOREIGN KEY REFERENCES Students(student_id),
    amount DECIMAL(10, 2),
    payment_date DATE
);
```

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



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

Ans - PRIMARY KEY and FOREIGN KEY constraints has been created already while creating the table

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

```
INSERT INTO Students (first_name, last_name, date_of_birth, email, phone_number)
VALUES
('John', 'Doe', '1995-08-15', 'john.doe@example.com', '1234567890'),
('Jane', 'Smith', '1993-05-23', 'jane.smith@example.com', '0987654321'),
('Michael', 'Johnson', '1992-11-02', 'michael.johnson@example.com', '1112223333'),
('Emily', 'Davis', '1994-04-11', 'emily.davis@example.com', '4445556666'),
('David', 'Brown', '1990-12-17', 'david.brown@example.com', '7778889999'),
('Sophia', 'Wilson', '1991-06-25', 'sophia.wilson@example.com', '1231231234'),
('Chris', 'Miller', '1994-09-08', 'chris.miller@example.com', '9876543210'),
('Amanda', 'Taylor', '1992-03-19', 'amanda.taylor@example.com', '5556667777'),
('Oliver', 'Anderson', '1995-01-30', 'oliver.anderson@example.com', '8889990000'),
('Ethan', 'Thomas', '1996-07-14', 'ethan.thomas@example.com', '1112224444');
```

ii. Courses

```
INSERT INTO Courses (course_name, credits, teacher_id)
VALUES
('Mathematics 101', 3, 1),
('Physics 101', 4, 2),
('Chemistry 101', 3, 3),
('Biology 101', 3, 4),
('Computer Science 101', 4, 5),
('History 101', 2, 6),
('Psychology 101', 3, 7),
('Sociology 101', 3, 8),
('Political Science 101', 2, 9),
('English 101', 3, 10);
```

iii. Enrollments

```
INSERT INTO Enrollments (student_id, course_id, enrollment_date)
VALUES
(1, 1, '2024-09-01'),
(2, 2, '2024-09-02'),
(3, 3, '2024-09-03'),
(4, 4, '2024-09-04'),
(5, 5, '2024-09-05'),
(6, 6, '2024-09-06'),
(7, 7, '2024-09-07'),
(8, 8, '2024-09-08'),
(9, 9, '2024-09-09'),
(10, 10, '2024-09-10');
```

iv. Teacher

```
INSERT INTO Teacher (first_name, last_name, email)
VALUES
('Sarah', 'Smith', 'sarah.smith@example.com'),
('James', 'Williams', 'james.williams@example.com'),
('Linda', 'Brown', 'linda.brown@example.com'),
('Robert', 'Jones', 'robert.jones@example.com'),
('Michael', 'Garcia', 'michael.garcia@example.com'),
('William', 'Rodriguez', 'william.rodriguez@example.com'),
('David', 'Martinez', 'david.martinez@example.com'),
('Richard', 'Hernandez', 'richard.hernandez@example.com'),
('Charles', 'Lopez', 'charles.lopez@example.com'),
('Thomas', 'Gonzalez', 'thomas.gonzalez@example.com');
```

v. Payments

```
INSERT INTO Payments (student_id, amount, payment_date)
VALUES
(1, 500.00, '2024-09-01'),
(2, 300.00, '2024-09-02'),
(3, 200.00, '2024-09-03'),
(4, 400.00, '2024-09-04'),
(5, 350.00, '2024-09-05'),
(6, 250.00, '2024-09-06'),
(7, 450.00, '2024-09-07'),
(8, 600.00, '2024-09-08'),
(9, 700.00, '2024-09-09'),
(10, 100.00, '2024-09-10');
```

Tasks 2: Select, Where, Between, AND, LIKE:

1. Write an SQL query to insert a new student into the "Students" table with the following details:

a. First Name: John

b. Last Name: Doe

c. Date of Birth: 1995-08-15

d. Email: john.doe@example.com

e. Phone Number: 1234567890

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

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

```
INSERT INTO Enrollments (student_id, course_id, enrollment_date)
VALUES (2, 2, '2024-09-15');
```

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 = 'new.email@example.com'
WHERE teacher_id = 1;
```

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

```
DELETE FROM Enrollments
WHERE student_id = 1 AND course_id = 3;
```

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 = 3
WHERE course_id = 2;
```

6. Delete a specific student from the "Students" table and remove all their enrollment records from the "Enrollments" table. Be sure to maintain referential integrity.

```
DELETE FROM Enrollments WHERE student_id = 1;
DELETE FROM Students WHERE student_id = 1;
```

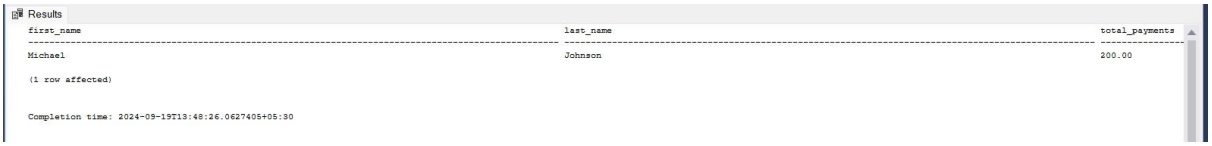
7. Update the payment amount for a specific payment record in the "Payments" table. Choose any payment record and modify the payment amount

```
UPDATE Payments
SET amount = 550.00
WHERE payment_id = 2;
```

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 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 = 3  -- Replace with the specific student ID
GROUP BY s.first_name, s.last_name;
```


Results

first_name	last_name	total_payments
Michael	Johnson	200.00

(1 row affected)

Completion time: 2024-09-19T13:48:26.0627405+05:30

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

Results

course_name	total_students
Biology 101	0
Chemistry 101	1
Computer Science 101	1
English 101	1
History 101	1
Mathematics 101	0
Physics 101	0
Political Science 101	1
Psychology 101	1
Sociology 101	1

Warning: Null value is eliminated by an aggregate or other SET operation.

(10 rows affected)

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.

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

Results

first_name	last_name
Jane	Smith
Emily	Davis
John	Doe

(3 rows affected)

Completion time: 2024-09-19T14:16:17.1560672+05:30

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.

```sql
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;
```

Results

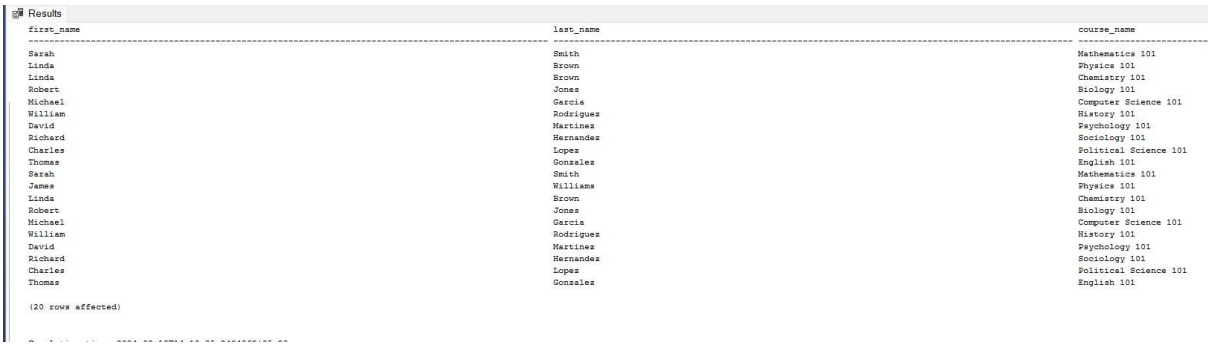
first_name	last_name	course_name
Michael	Johnson	Chemistry 101
David	Brown	Computer Science 101
Sophia	Wilson	History 101
Chris	Miller	Psychology 101
Amanda	Taylor	Sociology 101
Oliver	Anderson	Political Science 101
Ethan	Thomas	English 101

(7 rows affected)

Completion time: 2024-09-19T14:17:01.7563609+05:30

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;
```



Results

first_name	last_name	course_name
Sarah	Smith	Mathematics 101
Linda	Brown	Physics 101
Linda	Brown	Chemistry 101
Robert	Jones	Biology 101
Michael	Garcia	Computer Science 101
William	Rodriguez	History 101
David	Martinez	Psychology 101
Richard	Hernandez	Sociology 101
Charles	Lopez	Political Science 101
Thomas	Gonzalez	English 101
Sarah	Smith	Mathematics 101
James	Williams	Physics 101
Linda	Brown	Chemistry 101
Robert	Jones	Biology 101
Michael	Garcia	Computer Science 101
William	Rodriguez	History 101
David	Martinez	Psychology 101
Richard	Hernandez	Sociology 101
Charles	Lopez	Political Science 101
Thomas	Gonzalez	English 101

(20 rows affected)

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
INNER JOIN Enrollments e ON s.student_id = e.student_id
INNER JOIN Courses c ON e.course_id = c.course_id
WHERE c.course_name = 'Mathematics 101';  -- Replace with the specific course name
```

Results

first_name	last_name

(0 rows affected)

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;
```

Results

first_name	last_name
John	Doe

(1 row affected)

Completion time: 2024-09-19T14:21:54.9315805+05:30

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.

```sql
SELECT c.course_name
FROM Courses c
LEFT JOIN Enrollments e ON c.course_id = e.course_id
WHERE e.course_id IS NULL;
```

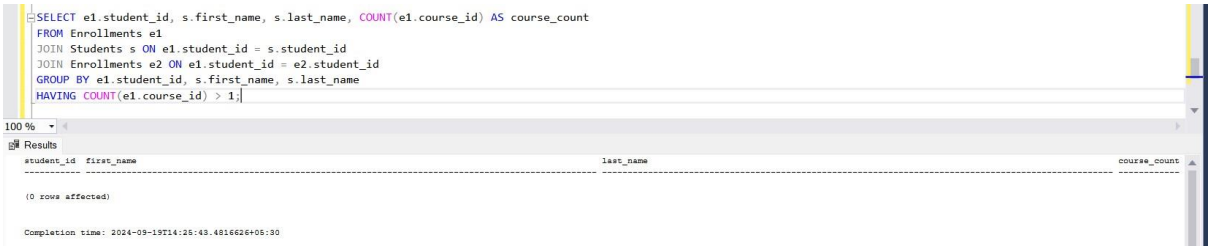
Results

```
course_name
----------------------------------------------------------------
Mathematics 101
Physics 101
Biology 101
Mathematics 101
Physics 101
Chemistry 101
Biology 101
Computer Science 101
History 101
Psychology 101
Sociology 101
Political Science 101
English 101

(13 rows affected)


Completion time: 2024-09-19T14:22:50.8262084+05:30
```

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.



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;
```

```
100 %
Results
first_name                                                                                        last_name
------------------------------------------------------------------------------------------------- ---------------------------------

(0 rows affected)
```

Task 4.

Subquery and its type:

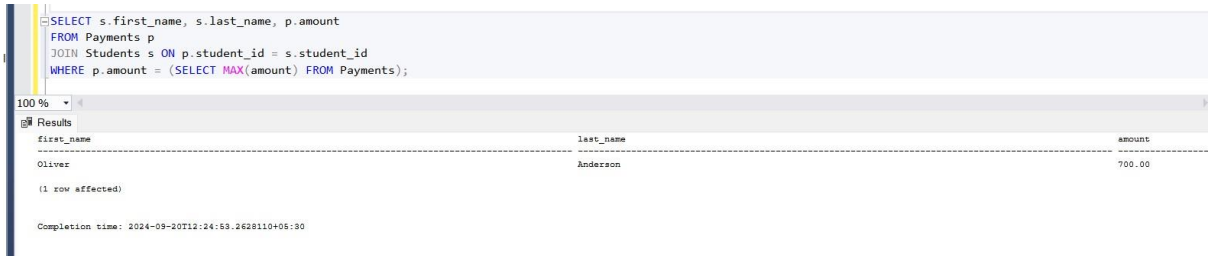
1. Write an SQL query to calculate the average number of students enrolled in each course. Use aggregate functions and subqueries to achieve this.

```
SELECT AVG(student_count) AS avg_students_per_course
FROM (SELECT COUNT(e.student_id) AS student_count
      FROM Enrollments e
      GROUP BY e.course_id) AS enrollment_counts;
```

```
100 %
Results
avg_students_per_course
-----------------------
1

(1 row affected)
```

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 s.first_name, s.last_name, p.amount
FROM Payments p
JOIN Students s ON p.student_id = s.student_id
WHERE p.amount = (SELECT MAX(amount) FROM Payments);
```

100 %

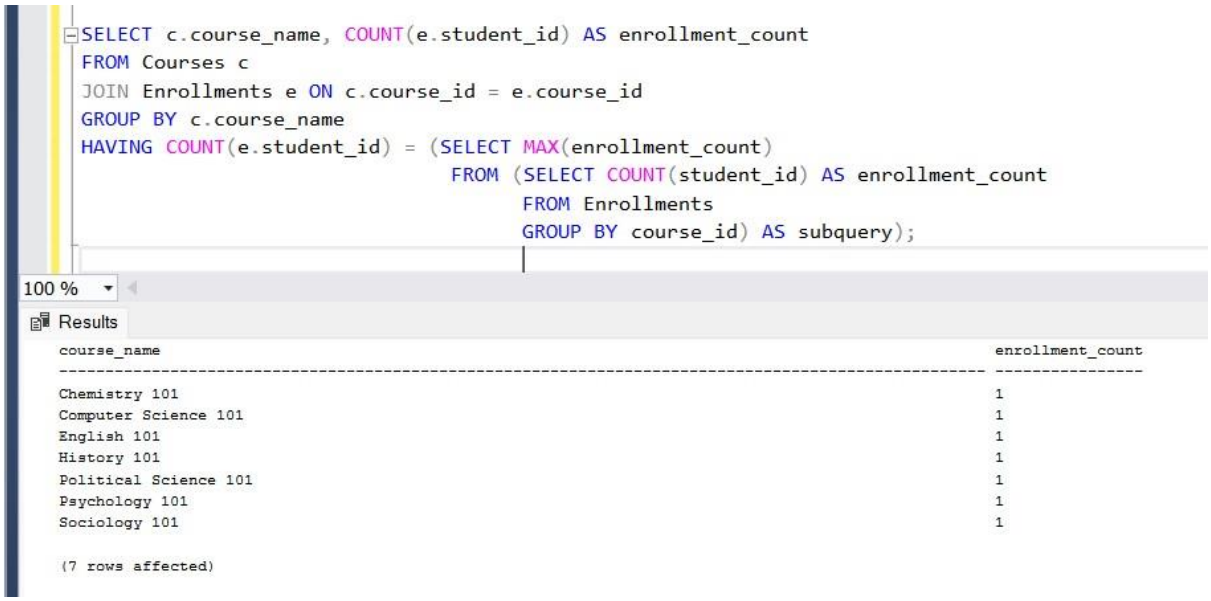
Results

first_name	last_name	amount
Oliver	Anderson	700.00

(1 row affected)

Completion time: 2024-09-20T12:24:53.2628110+05:30

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_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_name
HAVING COUNT(e.student_id) = (SELECT MAX(enrollment_count)
                              FROM (SELECT COUNT(student_id) AS enrollment_count
                                    FROM Enrollments
                                    GROUP BY course_id) AS subquery);
```

100 %

Results

course_name	enrollment_count
Chemistry 101	1
Computer Science 101	1
English 101	1
History 101	1
Political Science 101	1
Psychology 101	1
Sociology 101	1

(7 rows affected)

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

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

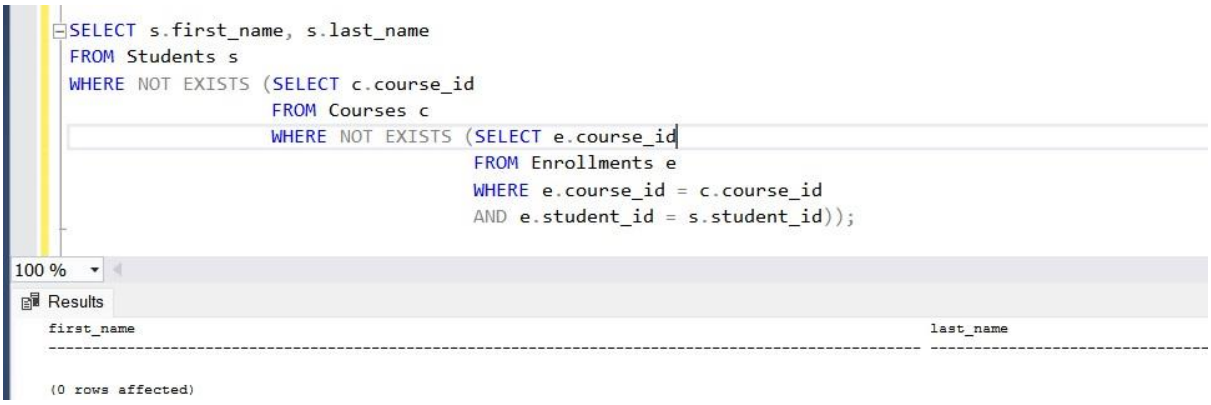
100 %

Results

first_name	last_name	total_payments
Charles	Lopez	700.00
David	Martinez	450.00
Linda	Brown	200.00
Michael	Garcia	350.00
Richard	Hernandez	600.00
Thomas	Gonzalez	100.00
William	Rodriguez	250.00

(7 rows affected)

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 s.first_name, s.last_name
FROM Students s
WHERE NOT EXISTS (SELECT c.course_id
                  FROM Courses c
                  WHERE NOT EXISTS (SELECT e.course_id
                                    FROM Enrollments e
                                    WHERE e.course_id = c.course_id
                                    AND e.student_id = s.student_id));
```

100 %

Results

first_name	last_name

(0 rows affected)

from my data no student has enrolled in all courses

6. Retrieve the names of teachers who have not been assigned to any courses. Use subqueries to find teachers with no course assignments.

```
                                    AND e.student_id = s.student_id));
SELECT t.first_name, t.last_name
FROM Teacher t
WHERE NOT EXISTS (SELECT c.course_id
                  FROM Courses c
                  WHERE c.teacher_id = t.teacher_id);
```

100 %

Results

```
first_name                                                                                                                       last_name
-------------------------------------------------------------------------------------------------------------------------------- --------------------------------------------------

(0 rows affected)

Completion time: 2024-09-20T12:29:04.4401423+05:30
```

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(DATEDIFF(YEAR, date_of_birth, GETDATE())) AS avg_age
FROM Students;
```

100 %

Results

```
avg_age
-----------
30

(1 row affected)
```

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

```
SELECT c.course_name
FROM Courses c
WHERE NOT EXISTS (SELECT e.course_id
                  FROM Enrollments e
                  WHERE e.course_id = c.course_id);
```

100 %

Results

```
course_name
--------------------------------------------------------------------------------------------------------------------------------
Mathematics 101
Physics 101
Biology 101
Mathematics 101
Physics 101
Chemistry 101
Biology 101
Computer Science 101
History 101
Psychology 101
Sociology 101
Political Science 101
English 101

(13 rows affected)
```

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 s.first_name, s.last_name, c.course_name, SUM(p.amount) AS total_payments
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
GROUP BY s.first_name, s.last_name, c.course_name;
```

100 %

Results

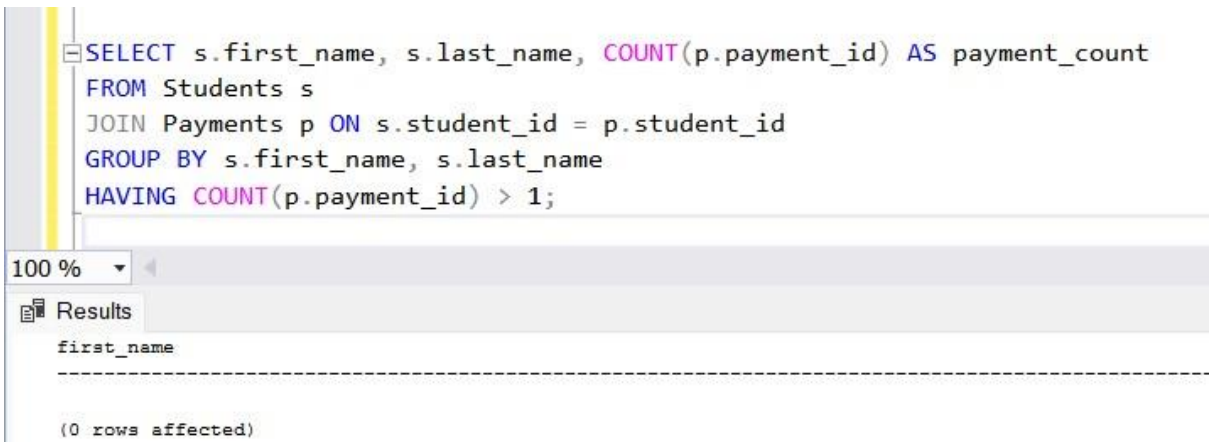
first_name	last_name	course_name
Amanda	Taylor	Sociology 101
Chris	Miller	Psychology 101
David	Brown	Computer Science 1
Ethan	Thomas	English 101
Michael	Johnson	Chemistry 101
Oliver	Anderson	Political Science
Sophia	Wilson	History 101

(7 rows affected)

Completion time: 2024-09-20T12:31:09.6727793+05:30

course_name	total_payments
Sociology 101	600.00
Psychology 101	450.00
Computer Science 101	350.00
English 101	100.00
Chemistry 101	200.00
Political Science 101	700.00
History 101	250.00

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.first_name, s.last_name, COUNT(p.payment_id) AS payment_count
FROM Students s
JOIN Payments p ON s.student_id = p.student_id
GROUP BY s.first_name, s.last_name
HAVING COUNT(p.payment_id) > 1;
```

100 %

Results

first_name

(0 rows affected)

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.

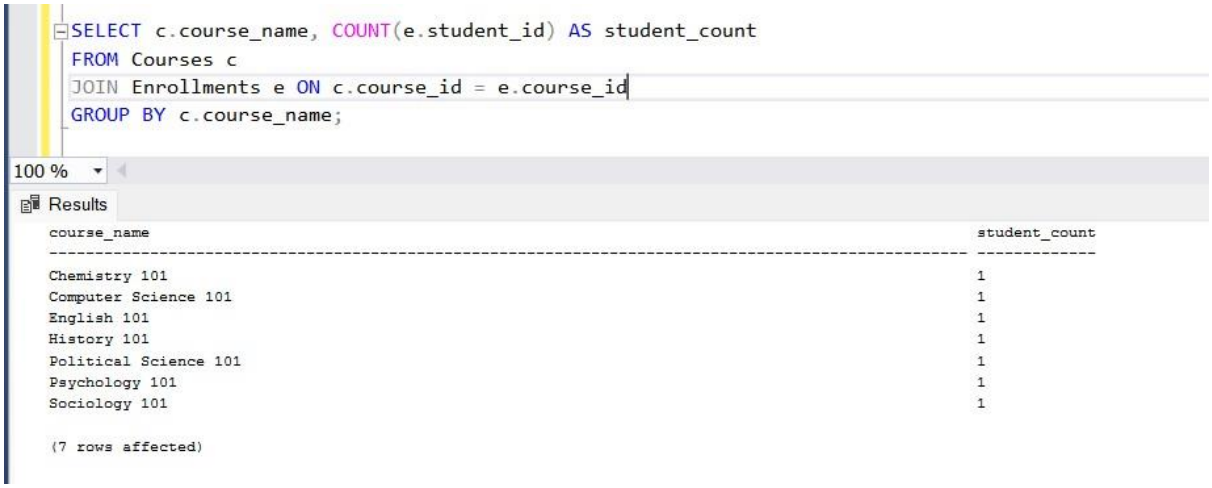
```
SELECT s.first_name, s.last_name, SUM(p.amount) AS total_payments
FROM Students s
JOIN Payments p ON s.student_id = p.student_id
GROUP BY s.first_name, s.last_name;
```

100 %

Results

first_name	last_name	total_payments
Oliver	Anderson	700.00
David	Brown	350.00
Emily	Davis	400.00
Michael	Johnson	200.00
Chris	Miller	450.00
Jane	Smith	550.00
Amanda	Taylor	600.00
Ethan	Thomas	100.00
Sophia	Wilson	250.00

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.student_id) AS student_count
FROM Courses c
JOIN Enrollments e ON c.course_id = e.course_id
GROUP BY c.course_name;
```

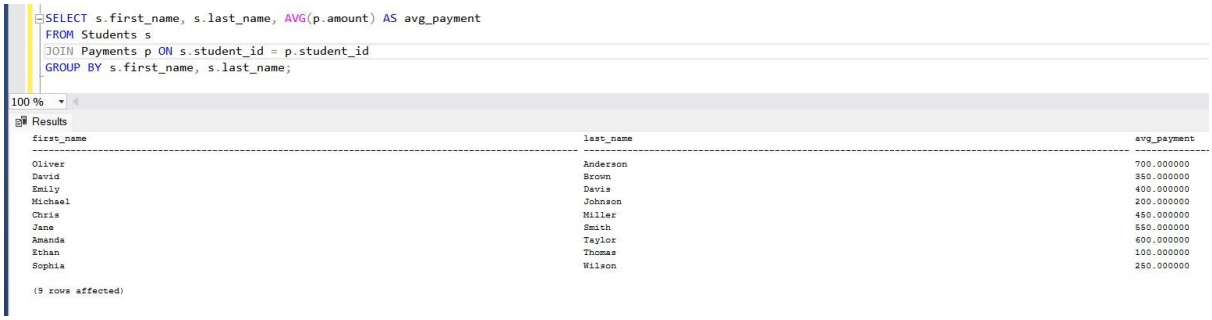
100 %

Results

course_name	student_count
Chemistry 101	1
Computer Science 101	1
English 101	1
History 101	1
Political Science 101	1
Psychology 101	1
Sociology 101	1

(7 rows affected)

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.first_name, s.last_name, AVG(p.amount) AS avg_payment
FROM Students s
JOIN Payments p ON s.student_id = p.student_id
GROUP BY s.first_name, s.last_name;
```

100 %

Results

first_name	last_name	avg_payment
Oliver	Anderson	700.000000
David	Brown	350.000000
Emily	Davis	400.000000
Michael	Johnson	200.000000
Chris	Miller	450.000000
Jane	Smith	550.000000
Amanda	Taylor	600.000000
Ethan	Thomas	100.000000
Sophia	Wilson	250.000000

(9 rows affected)