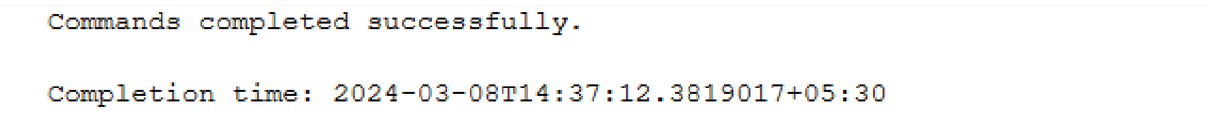
Assignment(Student Information System (SIS)) – MS SQL

- UDAYSANSTOSHKUMAR BURLU

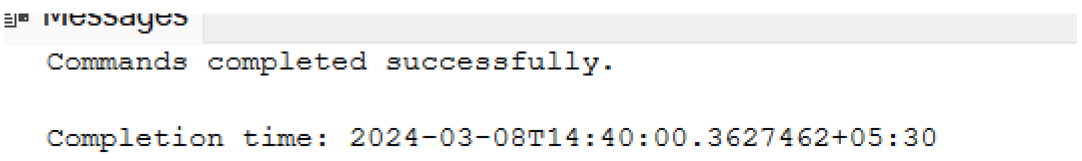
Task 1. Database Design

```
--1 Create the database named "SISDB
Create Database SISDB;
```



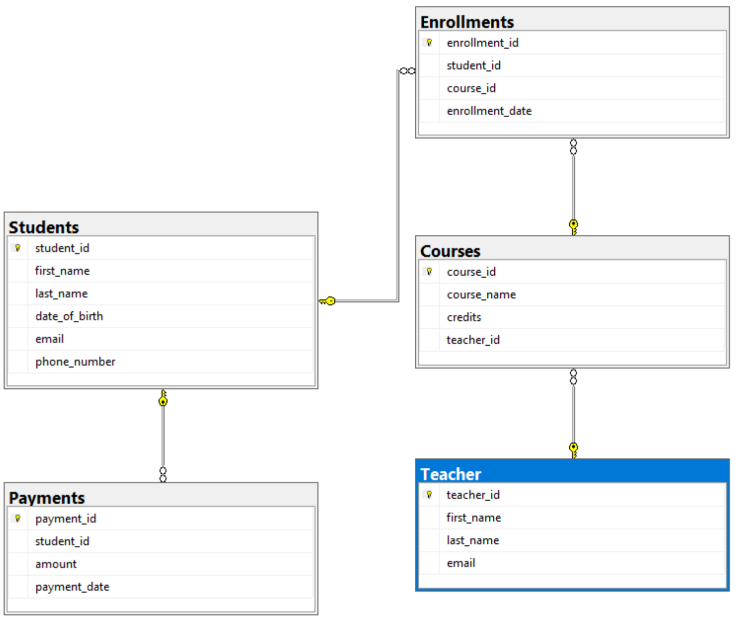
```
Commands completed successfully.

Completion time: 2024-03-08T14:37:12.3819017+05:30
```

```
--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
CREATE TABLE Students (
    student_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    date_of_birth DATE,
    email VARCHAR(100),
    phone_number VARCHAR(20)
);

CREATE TABLE Teacher (
    teacher_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100)
);


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

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


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



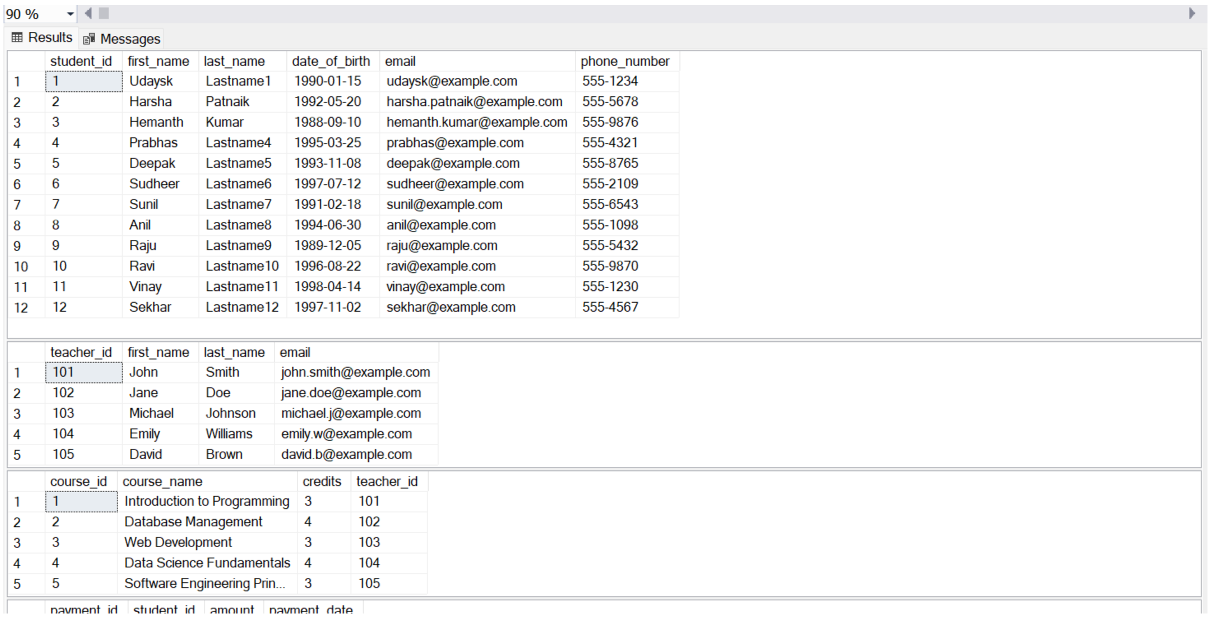
```
Messages
Commands completed successfully.

Completion time: 2024-03-08T14:40:00.3627462+05:30
```

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


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



```
--5 Insert at least 10 sample records into each of the following tables.
INSERT INTO Students (student_id, first_name, last_name, date_of_birth, email,
phone_number)
VALUES
(1, 'Udaysk', 'Lastname1', '1990-01-15', 'udaysk@example.com', '555-1234'),
(2, 'Harsha', 'Patnaik', '1992-05-20', 'harsha.patnaik@example.com', '555-5678'),
(3, 'Hemanth', 'Kumar', '1988-09-10', 'hemanth.kumar@example.com', '555-9876'),
(4, 'Prabhas', 'Lastname4', '1995-03-25', 'prabhas@example.com', '555-4321'),
(5, 'Deepak', 'Lastname5', '1993-11-08', 'deepak@example.com', '555-8765'),
(6, 'Sudheer', 'Lastname6', '1997-07-12', 'sudheer@example.com', '555-2109'),
(7, 'Sunil', 'Lastname7', '1991-02-18', 'sunil@example.com', '555-6543'),
(8, 'Anil', 'Lastname8', '1994-06-30', 'anil@example.com', '555-1098'),
(9, 'Raju', 'Lastname9', '1989-12-05', 'raju@example.com', '555-5432'),
(10, 'Ravi', 'Lastname10', '1996-08-22', 'ravi@example.com', '555-9870'),
(11, 'Vinay', 'Lastname11', '1998-04-14', 'vinay@example.com', '555-1230'),
(12, 'Sekhar', 'Lastname12', '1997-11-02', 'sekhar@example.com', '555-4567');

INSERT INTO Courses (course_id, course_name, credits, teacher_id)
VALUES
(1, 'Introduction to Programming', 3, 101),
```

```sql
(2, 'Database Management', 4, 102),
(3, 'Web Development', 3, 103),
(4, 'Data Science Fundamentals', 4, 104),
(5, 'Software Engineering Principles', 3, 105);

INSERT INTO Enrollments (enrollment_id, student_id, course_id, enrollment_date)
VALUES
(1, 1, 1, '2024-02-27'),
(2, 2, 3, '2024-02-28'),
(3, 3, 2, '2024-02-29'),
(4, 4, 4, '2024-03-01');

INSERT INTO Teacher (teacher_id, first_name, last_name, email)
VALUES
(101, 'John', 'Smith', 'john.smith@example.com'),
(102, 'Jane', 'Doe', 'jane.doe@example.com'),
(103, 'Michael', 'Johnson', 'michael.j@example.com'),
(104, 'Emily', 'Williams', 'emily.w@example.com'),
(105, 'David', 'Brown', 'david.b@example.com');

INSERT INTO Payments (payment_id, student_id, amount, payment_date)
VALUES
(1, 1, 100.00, '2024-03-03'),
(2, 2, 75.50, '2024-03-04'),
(3, 3, 120.00, '2024-03-05'),
(4, 4, 90.25, '2024-03-06'),
(5, 5, 150.50, '2024-03-07');

SELECT * FROM STUDENTS;
SELECT * FROM Teacher;
SELECT * FROM Courses;
SELECT * FROM Payments;
SELECT * FROM Enrollments;
```


90 %

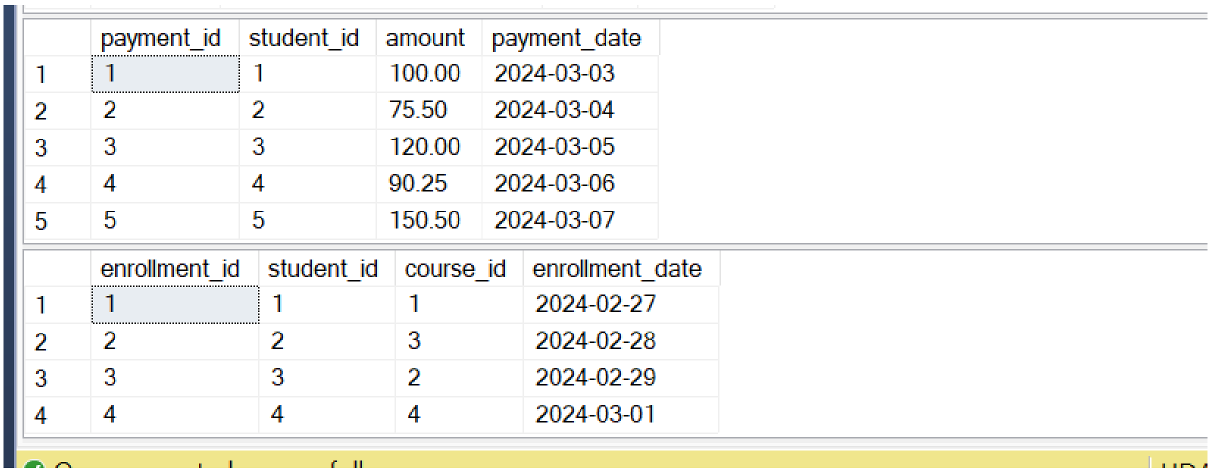
Results | Messages

	student_id	first_name	last_name	date_of_birth	email	phone_number
1	1	Udaysk	Lastname1	1990-01-15	udaysk@example.com	555-1234
2	2	Harsha	Patnaik	1992-05-20	harsha.patnaik@example.com	555-5678
3	3	Hemanth	Kumar	1988-09-10	hemanth.kumar@example.com	555-9876
4	4	Prabhas	Lastname4	1995-03-25	prabhas@example.com	555-4321
5	5	Deepak	Lastname5	1993-11-08	deepak@example.com	555-8765
6	6	Sudheer	Lastname6	1997-07-12	sudheer@example.com	555-2109
7	7	Sunil	Lastname7	1991-02-18	sunil@example.com	555-6543
8	8	Anil	Lastname8	1994-06-30	anil@example.com	555-1098
9	9	Raju	Lastname9	1989-12-05	raju@example.com	555-5432
10	10	Ravi	Lastname10	1996-08-22	ravi@example.com	555-9870
11	11	Vinay	Lastname11	1998-04-14	vinay@example.com	555-1230
12	12	Sekhar	Lastname12	1997-11-02	sekhar@example.com	555-4567

	teacher_id	first_name	last_name	email
1	101	John	Smith	john.smith@example.com
2	102	Jane	Doe	jane.doe@example.com
3	103	Michael	Johnson	michael.j@example.com
4	104	Emily	Williams	emily.w@example.com
5	105	David	Brown	david.b@example.com

	course_id	course_name	credits	teacher_id
1	1	Introduction to Programming	3	101
2	2	Database Management	4	102
3	3	Web Development	3	103
4	4	Data Science Fundamentals	4	104
5	5	Software Engineering Prin...	3	105

	payment_id	student_id	amount	payment_date



	payment_id	student_id	amount	payment_date
1	1	1	100.00	2024-03-03
2	2	2	75.50	2024-03-04
3	3	3	120.00	2024-03-05
4	4	4	90.25	2024-03-06
5	5	5	150.50	2024-03-07

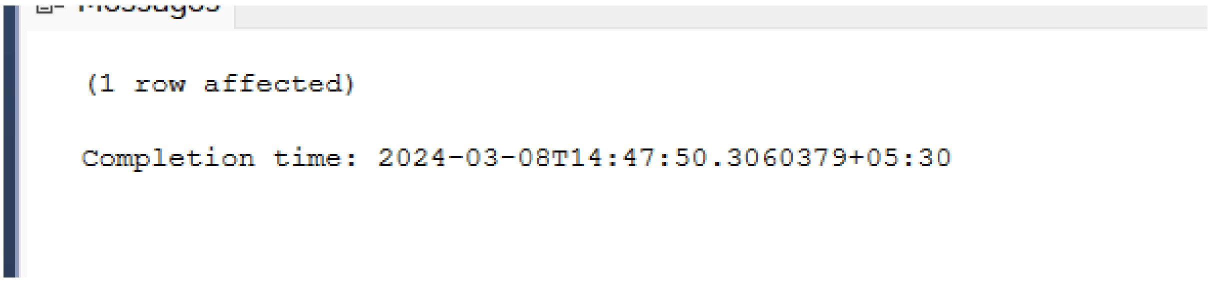
	enrollment_id	student_id	course_id	enrollment_date
1	1	1	1	2024-02-27
2	2	2	3	2024-02-28
3	3	3	2	2024-02-29
4	4	4	4	2024-03-01

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

```
--Task 2

--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 (student_id, first_name, last_name, date_of_birth, email,
phone_number)
VALUES (13, 'John', 'Doe', '1995-08-15', 'john.doe@example.com', '1234567890');
```

```
(1 row affected)

Completion time: 2024-03-08T14:47:17.1435126+05:30
```

```
--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.
INSERT INTO Enrollments (enrollment_id, student_id, course_id, enrollment_date)
VALUES (7, 1, 2, '2024-03-01');
```



```
(1 row affected)

Completion time: 2024-03-08T14:47:50.3060379+05:30
```

```
--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 = 'TEACHER@GMAIL.COM'
WHERE teacher_id = 105
```

```
(1 row affected)

Completion time: 2024-03-08T14:48:15.6026963+05:30
```

```sql
--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 = 1
```

```
(1 row affected)

Completion time: 2024-03-08T14:48:42.7151127+05:30
```

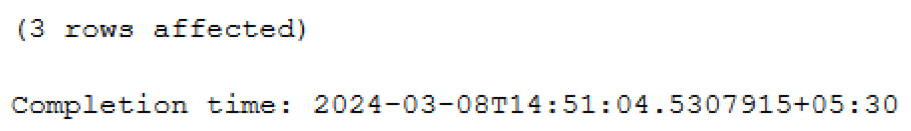
```sql
--5 Update the "Courses" table to assign a specific teacher to a course. Choose any course and teacher from the respective tables.
UPDATE COURSES
SET teacher_id = 104
WHERE course_id = 3
```

```
(1 row affected)

Completion time: 2024-03-08T14:50:03.9537241+05:30
```

```sql
--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.
BEGIN TRANSACTION;

DELETE FROM Enrollments
WHERE student_id = 5;

DELETE FROM Payments
WHERE student_id = 5;

DELETE FROM Students
WHERE student_id = 5;

COMMIT;
```

```
(1 row affected)

(0 rows affected)

(1 row affected)

(1 row affected)

Completion time: 2024-03-08T14:49:22.6326837+05:30
```

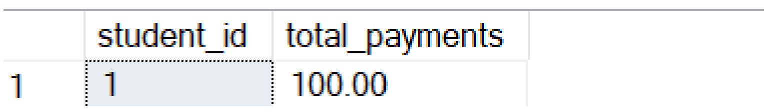
```
--7 Update the payment amount for a specific payment record in the "Payments" table.
Choose any  payment record and modify the payment amount.
UPDATE Payments
SET AMOUNT = 300
WHERE payment_date BETWEEN '2024-03-04' AND '2024-03-06'
```



```
(3 rows affected)

Completion time: 2024-03-08T14:51:04.5307915+05:30
```

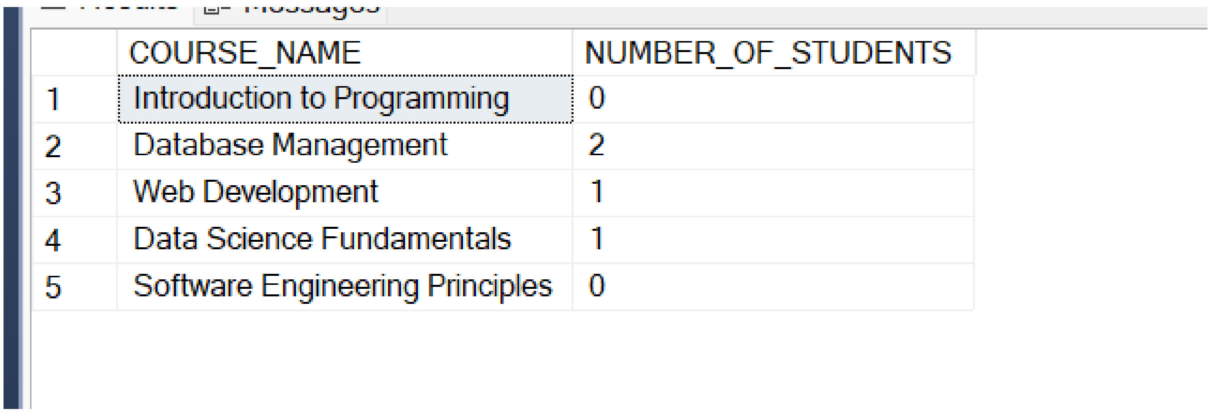
Task 3. Aggregate functions, Having, Order By, GroupBy and Joins:

```
--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
SELECT student_id, SUM(amount) AS total_payments
FROM Payments
WHERE student_id = 1
GROUP BY student_id;
```



	student_id	total_payments
1	1	100.00

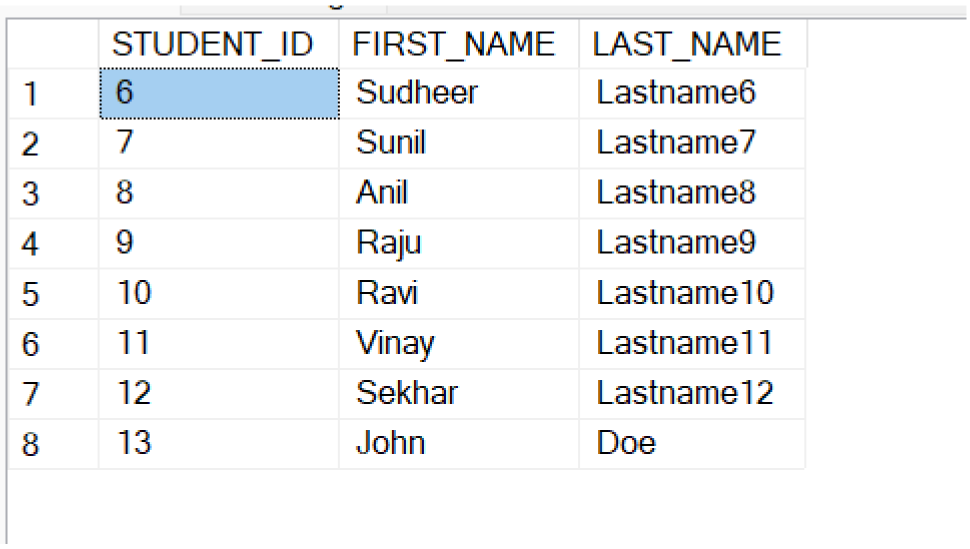
```
--2 Write an SQL query to retrieve a list of courses along with the count of students
enrolled in each  course. Use a JOIN operation between the "Courses" table and the
"Enrollments" table.

SELECT COURSES.COURSE_NAME, COUNT(ENROLLMENTS.STUDENT_ID) NUMBER_OF_STUDENTS
FROM COURSES LEFT JOIN ENROLLMENTS
ON COURSES.COURSE_ID = ENROLLMENTS.COURSE_ID
GROUP BY COURSES.COURSE_ID, COURSES.COURSE_NAME;
```



	COURSE_NAME	NUMBER_OF_STUDENTS
1	Introduction to Programming	0
2	Database Management	2
3	Web Development	1
4	Data Science Fundamentals	1
5	Software Engineering Principles	0

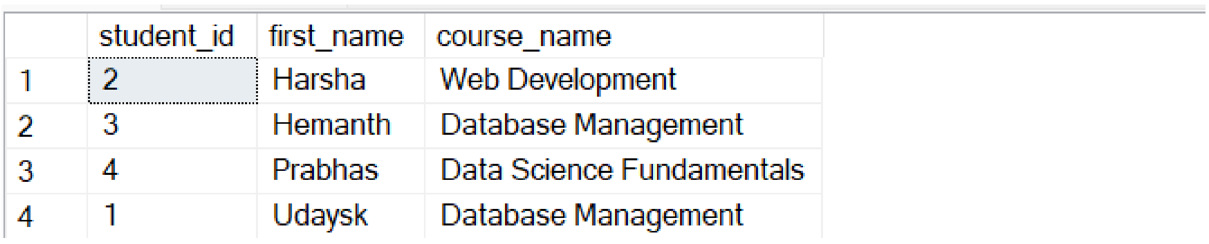
```
--3 Write an SQL query to find the names of students who have not enrolled in any
course. Use a  LEFT JOIN between the "Students" table and the "Enrollments" table to
identify students  without enrollments
SELECT S.STUDENT_ID, S.FIRST_NAME, S.LAST_NAME
FROM Students S
```

```sql
LEFT JOIN Enrollments E ON S.student_id=E.student_id
WHERE E.student_id IS NULL;
```



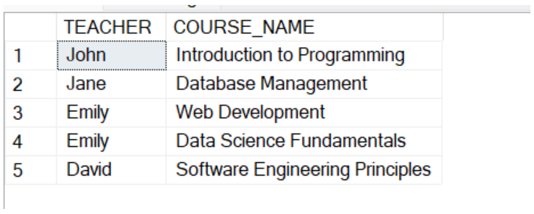
	STUDENT_ID	FIRST_NAME	LAST_NAME
1	6	Sudheer	Lastname6
2	7	Sunil	Lastname7
3	8	Anil	Lastname8
4	9	Raju	Lastname9
5	10	Ravi	Lastname10
6	11	Vinay	Lastname11
7	12	Sekhar	Lastname12
8	13	John	Doe

```sql
--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.student_id, s.first_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
```



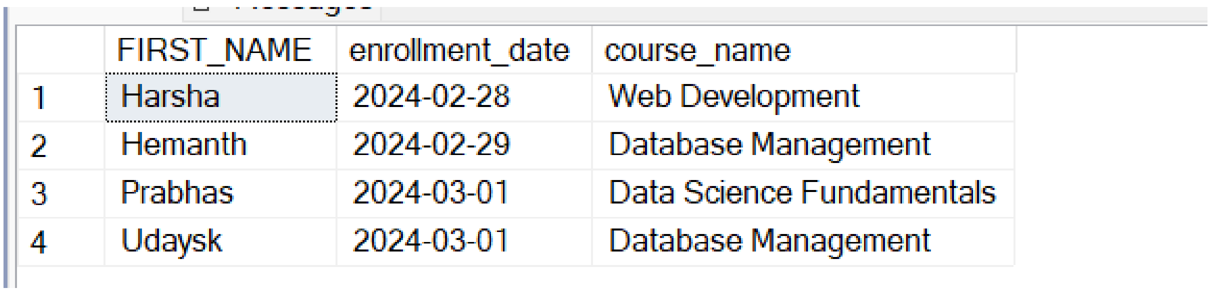
	student_id	first_name	course_name
1	2	Harsha	Web Development
2	3	Hemanth	Database Management
3	4	Prabhas	Data Science Fundamentals
4	1	Udaysk	Database Management

```sql
--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 TEACHER, C.COURSE_NAME
FROM Courses C
JOIN Teacher T
ON C.teacher_id = T.teacher_id

SELECT T.FIRST_NAME TEACHER, C.COURSE_NAME
FROM tEACHER T
JOIN COURSES C
ON C.teacher_id = T.teacher_id
```



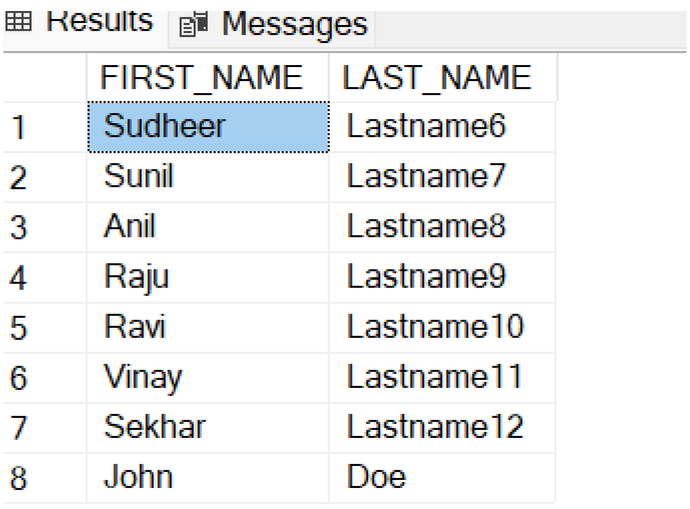
	TEACHER	COURSE_NAME
1	John	Introduction to Programming
2	Jane	Database Management
3	Emily	Web Development
4	Emily	Data Science Fundamentals
5	David	Software Engineering Principles

```
--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, E.enrollment_date, 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
```



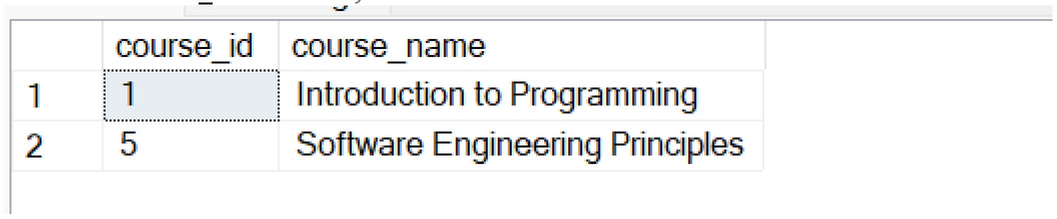
	FIRST_NAME	enrollment_date	course_name
1	Harsha	2024-02-28	Web Development
2	Hemanth	2024-02-29	Database Management
3	Prabhas	2024-03-01	Data Science Fundamentals
4	Udaysk	2024-03-01	Database Management

```
--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;
```



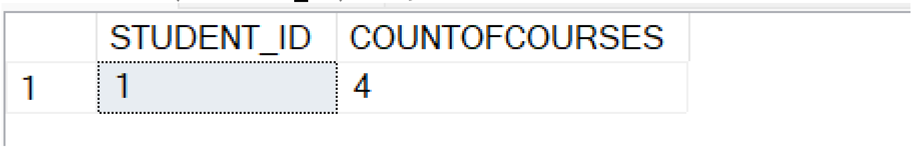
	FIRST_NAME	LAST_NAME
1	Sudheer	Lastname6
2	Sunil	Lastname7
3	Anil	Lastname8
4	Raju	Lastname9
5	Ravi	Lastname10
6	Vinay	Lastname11
7	Sekhar	Lastname12
8	John	Doe

```
--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.course_id IS NULL;
```



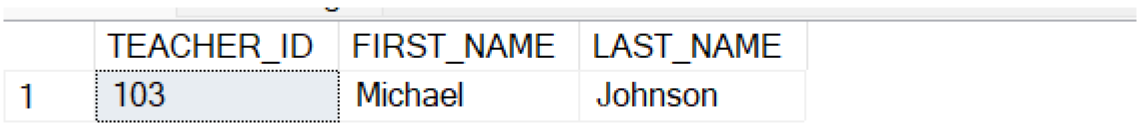
	course_id	course_name
1	1	Introduction to Programming
2	5	Software Engineering Principles

```sql
--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.
INSERT INTO Enrollments (enrollment_id, student_id, course_id, enrollment_date)
VALUES (8, 1, 2, '2024-03-01');


SELECT DISTINCT E.STUDENT_ID, COUNT(E.COURSE_ID) COUNTOFCOURSES
FROM ENROLLMENTS E
JOIN ENROLLMENTS C
ON E.STUDENT_ID = C.STUDENT_ID
GROUP BY E.STUDENT_ID, E.COURSE_ID
HAVING COUNT(E.COURSE_ID) > 1;
```



	STUDENT_ID	COUNTOFCOURSES
1	1	4

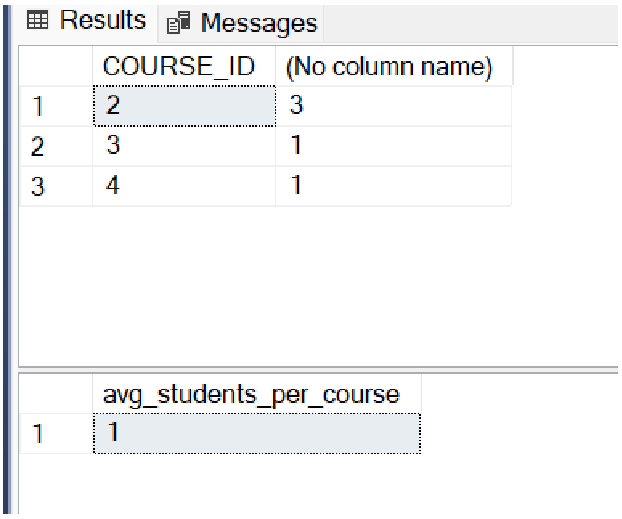
```sql
--10  Find teachers who are not assigned to any courses. Use a LEFT JOIN between the
"Teacher" table and the "Courses" table and filter for teachers with NULL course
assignment
SELECT T.TEACHER_ID, T.FIRST_NAME, T.LAST_NAME
FROM TEACHER T
LEFT JOIN COURSES C
ON T.TEACHER_ID = C.TEACHER_ID
WHERE C.TEACHER_ID IS NULL;
```



	TEACHER_ID	FIRST_NAME	LAST_NAME
1	103	Michael	Johnson

Task 4: Subquery and its type:

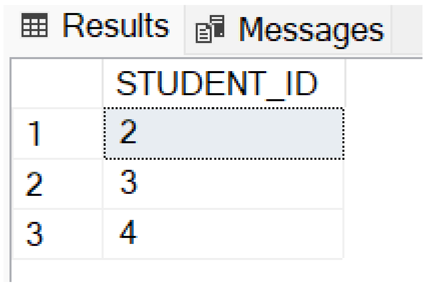
```sql
--1 Write an SQL query to calculate the average number of students enrolled in each
course. Use aggregate functions and subqueries to achieve this
SELECT E.COURSE_ID, COUNT(E.STUDENT_id)
FROM Enrollments E
GROUP BY E.course_id

SELECT AVG(enrollment_count) AS avg_students_per_course
FROM (SELECT course_id, COUNT(DISTINCT student_id) AS enrollment_count
      FROM Enrollments
      GROUP BY course_id) AS CourseEnrollmentCounts;
```



	COURSE_ID	(No column name)
1	2	3
2	3	1
3	4	1

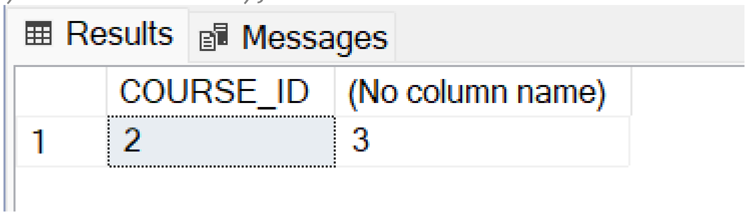
	avg_students_per_course
1	1

```
--2 Identify the student(s) who made the highest payment. Use a subquery to find the
maximum payment amount and then retrieve the student(s) associated with that amount
SELECT STUDENT_ID
FROM PAYMENTS
WHERE AMOUNT = (
SELECT MAX(AMOUNT) FROM Payments)
```



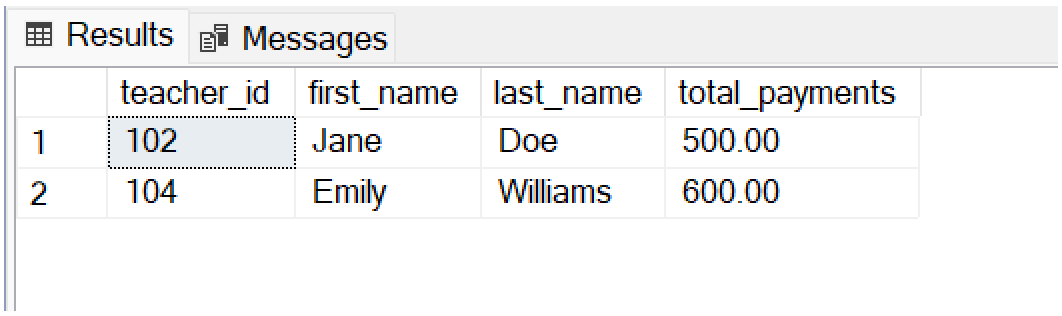
	STUDENT_ID
1	2
2	3
3	4

```
--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 COURSE_ID, COUNT(student_id) FROM Enrollments
group by course_id
having COUNT(student_id) = (
SELECT MAX(ENROLLMENTCOUNT) FROM (
SELECT COURSE_ID, COUNT(student_id) ENROLLMENTCOUNT
FROM Enrollments
group by course_id
) ENROLLMENTCOUNT);
```



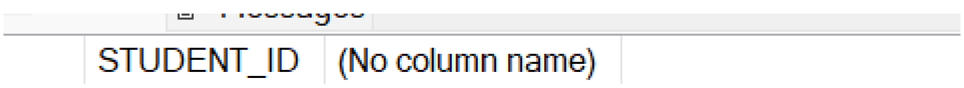
	COURSE_ID	(No column name)
1	2	3

```
--4 Calculate the total payments made to courses taught by each teacher. Use
subqueries to sum payments for each teacher's courses
SELECT t.teacher_id, t.first_name, t.last_name, SUM(p.amount) AS total_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.teacher_id, t.first_name, t.last_name;
```



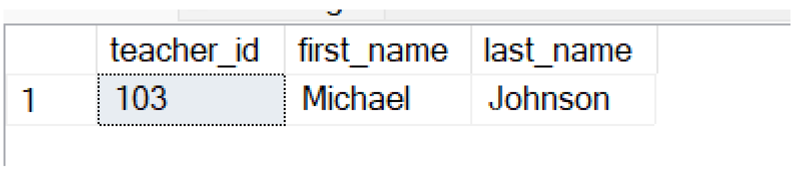
	teacher_id	first_name	last_name	total_payments
1	102	Jane	Doe	500.00
2	104	Emily	Williams	600.00

```
--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, COUNT(COURSE_ID)
FROM Enrollments
GROUP BY student_id
HAVING COUNT(COURSE_ID) = (SELECT COUNT(DISTINCT COURSE_ID) FROM Courses)
```



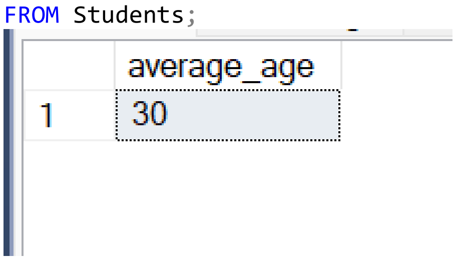
STUDENT_ID	(No column name)

```
--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);
```



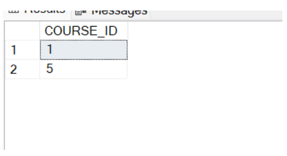
	teacher_id	first_name	last_name
1	103	Michael	Johnson

```
--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 average_age
FROM Students;
```



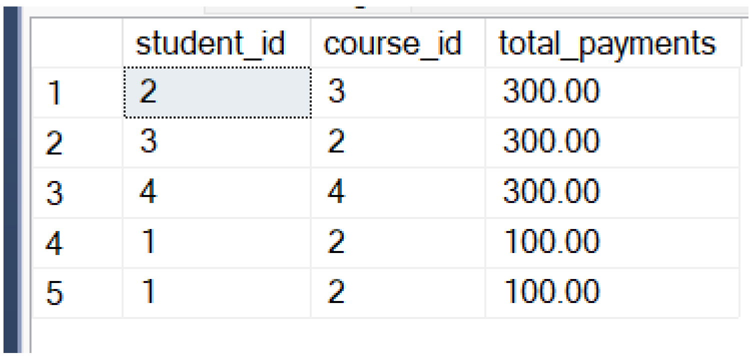
	average_age
1	30

```
--8 Identify courses with no enrollments. Use subqueries to find courses without
enrollment records
SELECT COURSE_ID
FROM COURSES
WHERE course_id NOT IN (SELECT course_id FROM Enrollments);
```



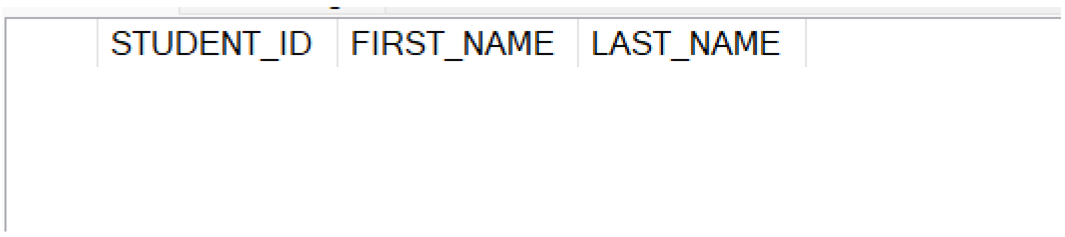
	COURSE_ID
1	1
2	5

```sql
--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;
```



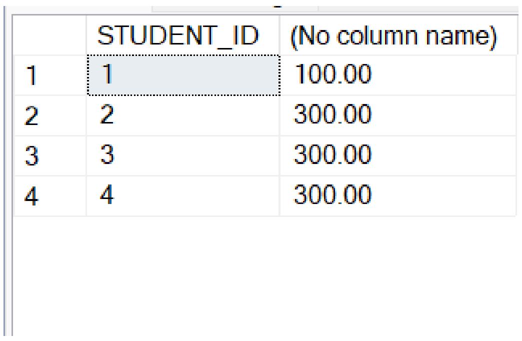
	student_id	course_id	total_payments
1	2	3	300.00
2	3	2	300.00
3	4	4	300.00
4	1	2	100.00
5	1	2	100.00

```sql
--10 Identify students who have made more than one payment. Use subqueries and
aggregate functions to count payments per student and filter for those with counts
greater than one
SELECT STUDENT_ID, FIRST_NAME, LAST_NAME
FROM STUDENTS
WHERE STUDENT_ID IN (
SELECT student_id
FROM Payments
GROUP BY student_id
HAVING COUNT(payment_id) > 1
);
```



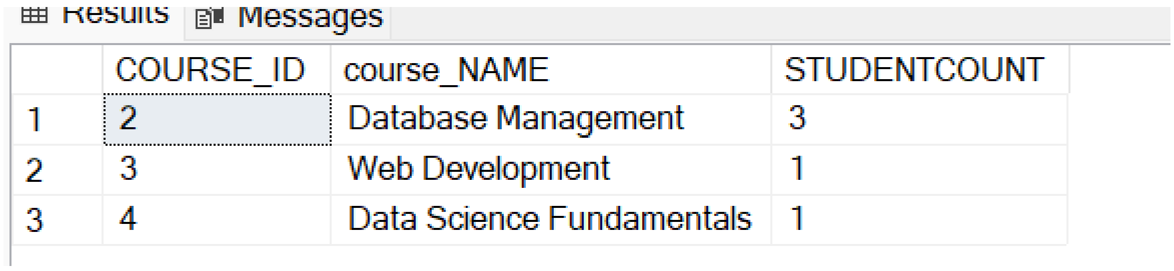
	STUDENT_ID	FIRST_NAME	LAST_NAME

```sql
--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, SUM(P.AMOUNT)
FROM Students S
JOIN Payments P
ON S.student_id = P.payment_id
GROUP BY S.student_id
```



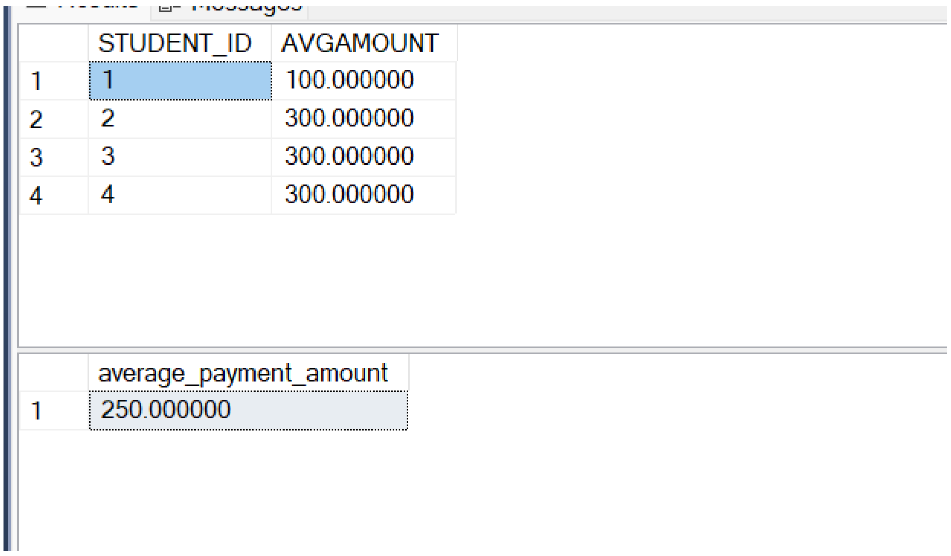
	STUDENT_ID	(No column name)
1	1	100.00
2	2	300.00
3	3	300.00
4	4	300.00

```
--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_ID, C.course_NAME, COUNT(E.STUDENT_ID) STUDENTCOUNT
FROM Courses C
JOIN ENROLLMENTS E
ON C.course_id = E.course_id
GROUP BY C.COURSE_ID, C.course_NAME
```



	COURSE_ID	course_NAME	STUDENTCOUNT
1	2	Database Management	3
2	3	Web Development	1
3	4	Data Science Fundamentals	1

```
--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, AVG(P.AMOUNT) AVGAMOUNT
FROM Students S
JOIN Payments P
ON S.student_id = P.payment_id
GROUP BY S.student_id

SELECT AVG(amount) AS average_payment_amount
FROM Payments;
```



	STUDENT_ID	AVGAMOUNT
1	1	100.000000
2	2	300.000000
3	3	300.000000
4	4	300.000000

	average_payment_amount
1	250.000000