Assignment 2(Student Information System)

💡 *Name : Pradum singh*

- ***Task 1: Database Design.***
  1. *Create the database named "SISDB"*
     a.
```
CREATE DATABASE SISDB;
USE 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

```
student_id INT PRIMARY KEY,
first_name VARCHAR(50),
last_name VARCHAR(50),
date_of_birth DATE,
email VARCHAR(100),
phone_number VARCHAR(15)
);
```

     b. Courses

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

     c. Enrollments

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

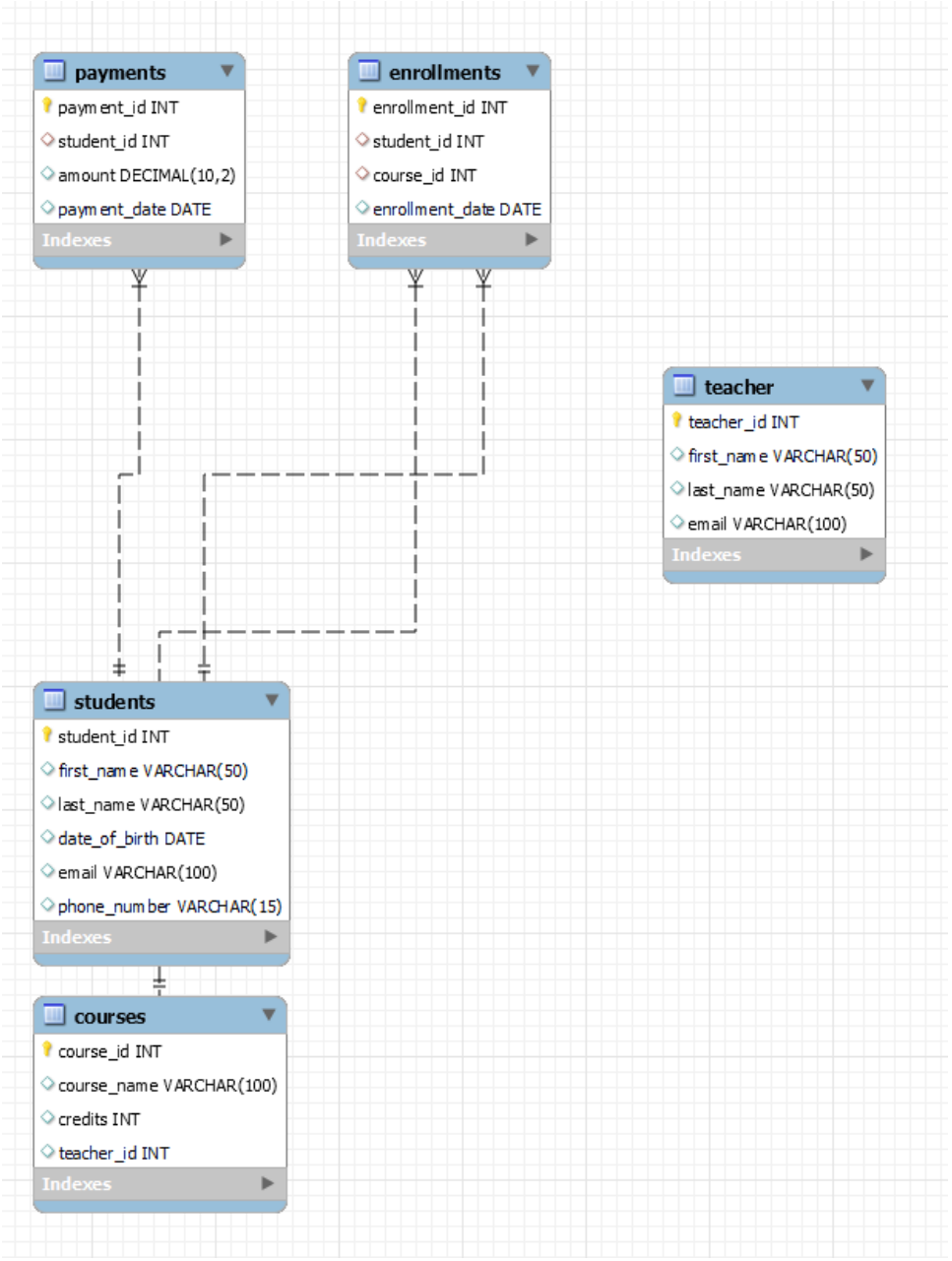
d. Teacher

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

e. Payments

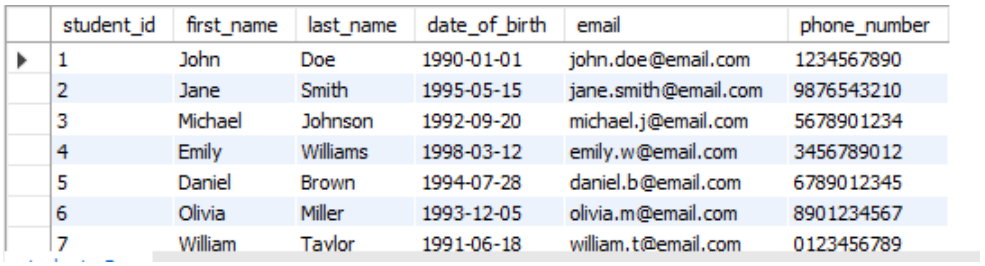
```
CREATE TABLE Payments (
payment_id INT PRIMARY KEY,
student_id INT,
amount DECIMAL(10, 2),
payment_date DATE,
FOREIGN KEY (student_id) REFERENCES Students(student_id)
);
```

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.*
    a. *students*

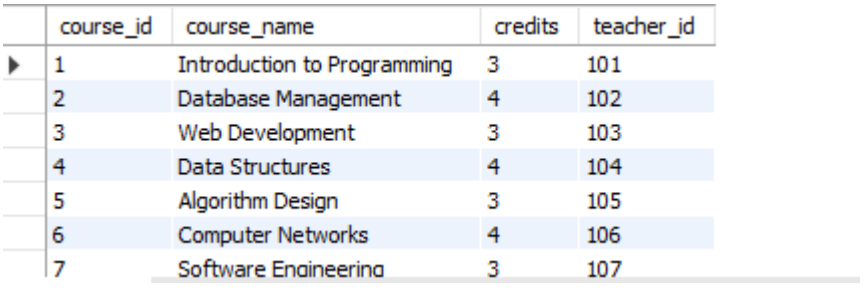
i.
```
INSERT INTO Students VALUES
(1, 'John', 'Doe', '1990-01-01', 'john.doe@email.com', '1234567890'),
(2, 'Jane', 'Smith', '1995-05-15', 'jane.smith@email.com', '9876543210'),
(3, 'Michael', 'Johnson', '1992-09-20', 'michael.j@email.com',
'5678901234'),
(4, 'Emily', 'Williams', '1998-03-12', 'emily.w@email.com',
'3456789012'),
(5, 'Daniel', 'Brown', '1994-07-28', 'daniel.b@email.com', '6789012345'),
(6, 'Olivia', 'Miller', '1993-12-05', 'olivia.m@email.com',
'8901234567'),
(7, 'William', 'Taylor', '1991-06-18', 'william.t@email.com',
'0123456789'),
(8, 'Sophia', 'Anderson', '1997-02-14', 'sophia.a@email.com',
'3210987654'),
(9, 'David', 'White', '1996-10-30', 'david.w@email.com', '2345678901'),
(10, 'Ava', 'Martin', '1999-04-23', 'ava.m@email.com', '4567890123');
```



student_id	first_name	last_name	date_of_birth	email	phone_number
1	John	Doe	1990-01-01	john.doe@email.com	1234567890
2	Jane	Smith	1995-05-15	jane.smith@email.com	9876543210
3	Michael	Johnson	1992-09-20	michael.j@email.com	5678901234
4	Emily	Williams	1998-03-12	emily.w@email.com	3456789012
5	Daniel	Brown	1994-07-28	daniel.b@email.com	6789012345
6	Olivia	Miller	1993-12-05	olivia.m@email.com	8901234567
7	William	Taylor	1991-06-18	william.t@email.com	0123456789

b. *Courses*

i.
```
INSERT INTO Courses VALUES
(1, 'Introduction to Programming', 3, 101),
(2, 'Database Management', 4, 102),
(3, 'Web Development', 3, 103),
(4, 'Data Structures', 4, 104),
(5, 'Algorithm Design', 3, 105),
(6, 'Computer Networks', 4, 106),
(7, 'Software Engineering', 3, 107),
(8, 'Database Design', 4, 108),
(9, 'Operating Systems', 3, 109),
(10, 'Artificial Intelligence', 4, 110);
```



course_id	course_name	credits	teacher_id
1	Introduction to Programming	3	101
2	Database Management	4	102
3	Web Development	3	103
4	Data Structures	4	104
5	Algorithm Design	3	105
6	Computer Networks	4	106
7	Software Engineering	3	107

c. *Teacher*

i.
```
INSERT INTO Teacher VALUES
(101, 'Professor', 'Smith', 'prof.smith@email.com'),
(102, 'Dr.', 'Johnson', 'dr.johnson@email.com'),
(103, 'Mrs.', 'Brown', 'mrs.brown@email.com'),
(104, 'Mr.', 'Taylor', 'mr.taylor@email.com'),
(105, 'Dr.', 'Anderson', 'dr.anderson@email.com'),
(106, 'Professor', 'Martin', 'prof.martin@email.com'),
(107, 'Mrs.', 'White', 'mrs.white@email.com'),
(108, 'Mr.', 'Miller', 'mr.miller@email.com'),
(109, 'Dr.', 'Davis', 'dr.davis@email.com'),
(110, 'Professor', 'Wilson', 'prof.wilson@email.com');
```

teacher_id	first_name	last_name	email
101	Professor	Smith	new.email@example.com
102	Dr.	Johnson	dr.johnson@email.com
103	Mrs.	Brown	mrs.brown@email.com
104	Mr.	Taylor	mr.taylor@email.com
105	Dr.	Anderson	dr.anderson@email.com
106	Professor	Martin	prof.martin@email.com
107	Mrs.	White	mrs.white@email.com

d. *payments*

i.
```
INSERT INTO Payments VALUES
(1, 1, 100.00, '2024-01-15'),
(2, 2, 150.00, '2024-01-16'),
(3, 3, 120.00, '2024-01-17'),
(4, 4, 200.00, '2024-01-18'),
(5, 5, 180.00, '2024-01-19'),
(6, 6, 130.00, '2024-01-20'),
(7, 7, 110.00, '2024-01-21'),
(8, 8, 160.00, '2024-01-22'),
(9, 9, 140.00, '2024-01-23'),
(10, 10, 190.00, '2024-01-24');
```

payment_id	student_id	amount	payment_date
1	1	120.00	2024-01-15
2	2	150.00	2024-01-16
3	3	120.00	2024-01-17
4	4	200.00	2024-01-18
5	5	180.00	2024-01-19
6	6	130.00	2024-01-20

- ***Task 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

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 (123, 1, 1, '2024-01-25');
```

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

1.
```
UPDATE Teacher
SET email = 'new.email@example.com'
WHERE teacher_id = 101; -- Replace with the appropriate teacher_id
```

teacher_id	first_name	last_name	email
101	Professor	Smith	new.email@example.com
102	Dr.	Johnson	dr.johnson@email.com
103	Mrs.	Brown	mrs.brown@email.com
104	Mr.	Taylor	mr.taylor@email.com
105	Dr.	Anderson	dr.anderson@email.com

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.*

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

enrollment_id	student_id	course_id	enrollment_date
2	2	2	2024-01-02
3	3	3	2024-01-03
4	4	4	2024-01-04
5	5	5	2024-01-05
6	6	6	2024-01-06

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

1.
```
UPDATE Courses
SET teacher_id = 102
WHERE course_id = 2;
```

	course_id	course_name	credits	teacher_id
▶	1	Introduction to Programming	3	101
	2	Database Management	4	102
	3	Web Development	3	103
	4	Data Structures	4	104
	5	Algorithm Design	3	105

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.*

1. ```
   DELETE FROM Students
   WHERE student_id = 3; -- Replace with the appropriate student_id
   ```

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

1. ```
   UPDATE Payments
   SET amount = 120.00 -- Replace with the desired amount
   WHERE payment_id = 1; -- Replace with the appropriate payment_id
   ```

	payment_id	student_id	amount	payment_date
▶	1	1	120.00	2024-01-15
	2	2	150.00	2024-01-16
	3	3	120.00	2024-01-17
	4	4	200.00	2024-01-18
	5	5	180.00	2024-01-19

payments 18

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

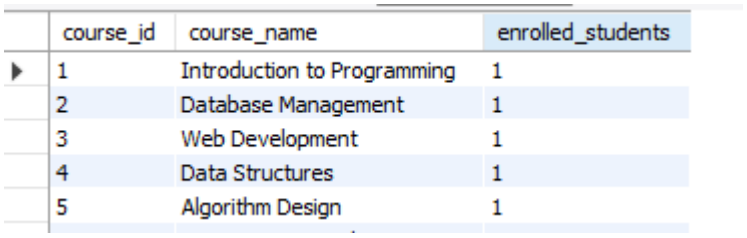
1. *Write an SQL query to calculate the total payments made by a specific student. You will need to join the "Payments" table with the "Students" table based on the student's ID.*

   a. ```
      SELECT s.first_name, s.last_name, SUM(p.amount) AS total_payments
      FROM Students s
      LEFT JOIN Payments p ON s.student_id = p.student_id
      WHERE s.student_id = 1; -- Replace with the appropriate student_id
      ```

| | student_id | total_payments |
|---|---|---|
| ▶ | 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.*
      ```

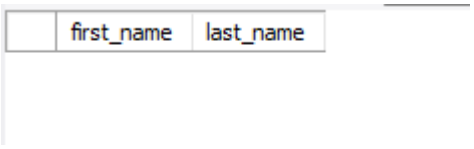
a.
```
SELECT c.course_id, course_name, COUNT(e.student_id) AS enrolled_students
FROM Courses c
LEFT JOIN Enrollments e ON c.course_id = e.course_id
GROUP BY c.course_id, course_name;
```



course_id	course_name	enrolled_students
1	Introduction to Programming	1
2	Database Management	1
3	Web Development	1
4	Data Structures	1
5	Algorithm Design	1

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.*

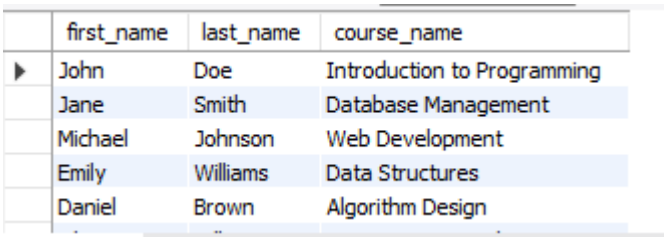
a.
```
SELECT s.first_name, s.last_name
FROM Students s
LEFT JOIN Enrollments e ON s.student_id = e.student_id
WHERE e.student_id IS NULL;
```



first_name	last_name

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.

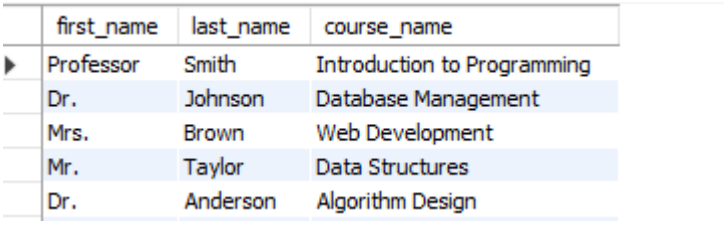
a.
```
SELECT s.first_name, s.last_name, c.course_name
FROM Students s
JOIN Enrollments e ON s.student_id = e.student_id
JOIN Courses c ON e.course_id = c.course_id;
```



first_name	last_name	course_name
John	Doe	Introduction to Programming
Jane	Smith	Database Management
Michael	Johnson	Web Development
Emily	Williams	Data Structures
Daniel	Brown	Algorithm Design

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.

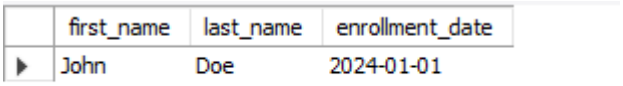
a.
```
SELECT t.first_name, t.last_name, c.course_name
FROM Teacher t
JOIN Courses c ON t.teacher_id = c.teacher_id;
```



first_name	last_name	course_name
Professor	Smith	Introduction to Programming
Dr.	Johnson	Database Management
Mrs.	Brown	Web Development
Mr.	Taylor	Data Structures
Dr.	Anderson	Algorithm Design

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.*

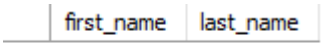
a.
```
SELECT s.first_name, s.last_name, e.enrollment_date
FROM Students s
JOIN Enrollments e ON s.student_id = e.student_id
JOIN Courses c ON e.course_id = c.course_id
WHERE c.course_id = 1; -- Replace with the appropriate course_id
```



first_name	last_name	enrollment_date
John	Doe	2024-01-01

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.

a.
```
SELECT s.first_name, s.last_name
FROM Students s
LEFT JOIN Payments p ON s.student_id = p.student_id
WHERE p.student_id IS NULL;
```



first_name	last_name

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.

a.
```
SELECT c.course_id, 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

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.*

   a.
```
SELECT s.first_name, s.last_name
FROM Students s
JOIN Enrollments e1 ON s.student_id = e1.student_id
JOIN Enrollments e2 ON s.student_id = e2.student_id AND e1.course_id <>
e2.course_id;
```

	first_name	last_name

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.

    a.
```
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;
```

	first_name	last_name

Task 4: Subquery and its type

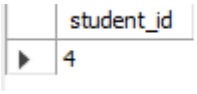
1.*Write an SQL query to calculate the average number of students enrolled in each course. Use*
aggregate functions and subqueries to achieve this.

1.
```
SELECT AVG(enrollment_count) AS average_students_per_course
FROM (
SELECT course_id, COUNT(student_id) AS enrollment_count
FROM Enrollments
GROUP BY course_id
) AS subquery;
```

	average_students_per_course
▶	1.0000

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.

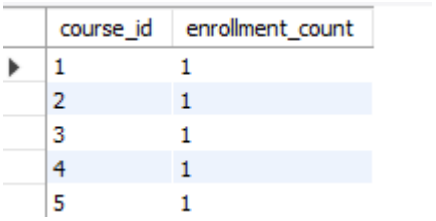
1. ```
   SELECT student_id, first_name, last_name
   FROM Students
   WHERE student_id = (SELECT student_id FROM Payments ORDER BY amount DESC LIMIT 1);
   ```



	student_id
▶	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.

1. ```
   SELECT course_id, course_name, enrollment_count
   FROM (
   SELECT course_id, course_name, COUNT(student_id) AS enrollment_count
   FROM Enrollments
   GROUP BY course_id, course_name
   ORDER BY enrollment_count DESC
   LIMIT 1
   ) AS subquery;
   ```



	course_id	enrollment_count
▶	1	1
	2	1
	3	1
	4	1
	5	1

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

1. ```
   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
   LEFT JOIN Enrollments e ON c.course_id = e.course_id
   LEFT 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
101	Professor	Smith	100.00
102	Dr.	Johnson	150.00
103	Mrs.	Brown	120.00
104	Mr.	Taylor	200.00
105	Dr.	Anderson	180.00

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

1. ```
   SELECT student_id, first_name, last_name
   FROM Students
   WHERE (SELECT COUNT(DISTINCT course_id) FROM Courses) = (SELECT COUNT(DISTINCT course_id) FROM Enrollments WHERE student_id = Students.student_id);
   ```

student_id	first_name	last_name
NULL	NULL	NULL

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

1. ```
   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
NULL	NULL	NULL

7.*Calculate the average age of all students. Use subqueries to calculate the age of each student based on their date of birth.*

1. ```
   SELECT AVG(TIMESTAMPDIFF(YEAR, date_of_birth, CURDATE())) AS average_age FROM Students;
   ```

average_age
28.6000

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

1. ```
   SELECT course_id, course_name
   FROM Courses
   WHERE course_id NOT IN (SELECT DISTINCT course_id FROM Enrollments);
   ```

	course_id	course_name
*	NULL	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.*

1.
```
SELECT s.student_id, 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
LEFT JOIN Payments p ON s.student_id = p.student_id
GROUP BY s.student_id, s.first_name, s.last_name, c.course_name;
```

	student_id	first_name	last_name	course_name	total_payments
▶	1	John	Doe	Introduction to Programming	100.00
	2	Jane	Smith	Database Management	150.00
	3	Michael	Johnson	Web Development	120.00
	4	Emily	Williams	Data Structures	200.00
	5	Daniel	Brown	Algorithm Design	180.00

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.*

1.
```
SELECT student_id, first_name, last_name
FROM Payments
WHERE student_id IN (SELECT student_id FROM Payments GROUP BY student_id HAVING
COUNT(payment_id) > 1);
```

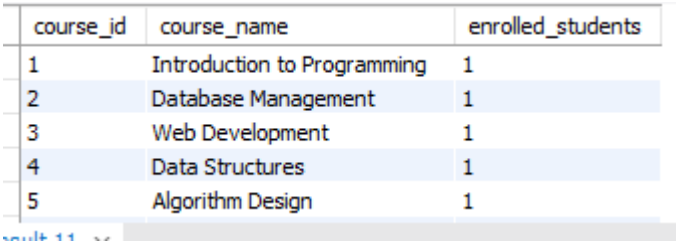
11.Write an SQL query to calculate the total payments made by each student. Join the "Students" table with the "Payments" table and use GROUP BY to calculate the sum of payments for each student.

1.
```
SELECT s.student_id, s.first_name, s.last_name, SUM(p.amount) AS
total_payments
FROM Students s
LEFT JOIN Payments p ON s.student_id = p.student_id
GROUP BY s.student_id, s.first_name, s.last_name;
```

	student_id	first_name	last_name	total_payments
▶	1	John	Doe	100.00
	2	Jane	Smith	150.00
	3	Michael	Johnson	120.00
	4	Emily	Williams	200.00
	5	Daniel	Brown	180.00

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.

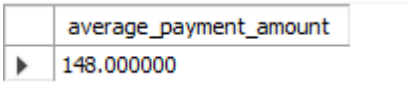
1.
```
SELECT c.course_id, course_name, COUNT(e.student_id) AS enrolled_students
FROM Courses c
LEFT JOIN Enrollments e ON c.course_id = e.course_id
GROUP BY c.course_id, course_name;
```



course_id	course_name	enrolled_students
1	Introduction to Programming	1
2	Database Management	1
3	Web Development	1
4	Data Structures	1
5	Algorithm Design	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 AVG(amount) AS average_payment_amount
FROM Payments;
```



average_payment_amount
148.000000