Assignment-3(SISDB)

```
use SISDB;
describe Students;
describe Courses;
describe Teacher;
describe Enrollments;
describe Payments;
-- Students
INSERT INTO Students (first_name, last_name, date_of_birth, email, phone_number)
VALUES
('Harry', 'Potter', '1995-08-15', 'harry.potter@gmail.com', '1234567890'),
('Shreya', 'Ghosal', '1998-03-20', 'shreya.ghosal@gmail.com', '9876543210'),
('Miley', 'Cyrus', '1997-11-10', 'miley.cyrus@gmail.com', '5551234567'),
('Udit', 'Narayanan', '1996-09-25', 'udit.narayanan@gmail.com', '4567890123'),
('Justin', 'Bieber', '1999-02-28', 'justin.bieber@gmail.com', '3216549870');

select*from Students;
-- course

select*from Courses;

INSERT INTO Courses (course_name, credits, teacher_id)
VALUES
('Mathematics', 3, 1),
('English Literature', 4, 2),
('Biology',3,4),
('Art', 2, 3),
('Music', 2, 3);

-- enrollments
```

```
select*from Enrollments;

INSERT INTO Enrollments (student_id, course_id, enrollment_date)
VALUES
(1, 1, '2024-01-15'),
(2, 2, '2024-02-07'),
(3, 3, '2024-03-05'),
(4, 4, '2024-01-27'),
(5, 5, '2024-02-02');

-- teacher

select*from Teacher;

INSERT INTO Teacher (first_name, last_name, email)
VALUES
('Arjit', 'Singh', 'arjit.singh@gmail.com'),
('hermione', 'Granger', 'hermione.granger@gmail.com'),
('Draco', 'Malfoy', 'draco.malfoy@gmail.com'),
('Ron', 'Weasley', 'ron.weasley@gmail.com'),
('Luna', 'Kalix', 'luna.kalix@gmail.com');

-- payments

select*from Payments;

INSERT INTO Payments (student_id, amount, payment_date)
VALUES
(1, 1500, '2024-01-20'),
(2, 3700, '2024-02-15'),
(3, 550, '2024-03-10'),
(4, 890, '2024-01-25'),
```

(5, 1200, '2024-03-01');

-- 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');

/*output:

1	Harry	Potter	1995-08-15	harry.potter@gmail.com	1234567890
2	Shreya	Ghosal	1998-03-20	shreya.ghosal@email.com	9876543210
3	Miley	Cyrus	1997-11-10	miley.cyrus@gmail.com	5551234567
4	Udit	Narayanan	1996-09-25	udit.narayanan@gmail.com.com	4567890123
5	Justin	Bieber	1999-02-28	justin.bieber@gmail.com	3216549870
6	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 enrollment date

INSERT INTO Enrollments (student_id, course_id, enrollment_date)

VALUES (3, 4,'2024-04-09');

-- 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='sunitha.abi@gmail.com'

where teacher_id=1;

/* output:

1	Arjit	Singh	sunitha.abi@gmail.com
2	hermione	Granger	hermione.granger@gmail.com
3	Draco	Malfoy	draco.malfoy@gmail.com
4	Ron	Weasley	ron.weasley@gmail.com
5	Luna	Kalix	luna.kalix@gmail.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.

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

/* output:

1	1	1	1	2024-01-15
	2	2	2	2024-02-07
	4	4	4	2024-01-27
	5	5	5	2024-02-02
	6	3	4	2024-04-09

*/

-- 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=2 where course_id=1;

```
/*      output:
   1    Mathematics          3   2
        2       English Literature      4       2
        3       Biology         3       4
        4       Art                 2       3
        5       Music           2       3
   */
```

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

```
DELETE from Students where student_id=2;
```

-- 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=1700 where payment_id=3;
```

```
 /*output:
   1    1       1500    2024-01-20
        2       2       3700    2024-02-15
        3       3       1700    2024-03-10
        4       4       890     2024-01-25
        5       5       1200    2024-03-01
   */
```

-- task 3

-- 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.student_id,sum(p.amount)
from Students s,Payments p
```

```
where s.student_id=p.student_id
and s.student_id=1
group by s.student_id;

/*        output:
 student_id       sum(p.amount)
          1                1500
   */

-- 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(*) as count_student
from Students s JOIN Enrollments e on s.student_id=e.student_id
             JOIN Courses c on e.course_id=c.course_id
             group by c.course_name;

/*        output:
 course_name   count_student
          Art                    2
          English Literature         1
          Mathematics            1
          Music              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.
select concat(first_name,last_name) as Name
from Students s
where s.student_id not in(select s.student_id
from Students s  JOIN Enrollments e on s.student_id=e.student_id);

/* ouput:
          Name
```

JohnDoe

*/

-- 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.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;
```

/* output:

first_name	last_name	course_name
Harry	Potter	Mathematics
Shreya	Ghosal	English Literature
Udit	Narayanan	Art
Justin	Bieber	Music
Miley	Cyrus	Art

*/

-- 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 concat(t.first_name,t.last_name) as Teacher_name,c.course_name
from Teacher t Join Courses c ON c.teacher_id=t.teacher_id
group by c.course_name;
```

/*output:

Teacher_name	course_name
DracoMalfoy	Art
RonWeasley	Biology
hermioneGranger	English Literature
hermioneGranger	Mathematics

```
        DracoMalfoy         Music
   */
```

-- 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 concat(s.first_name,s.last_name) as student_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_name='Mathematics';

/*output
        student_name    enrollment_date
        HarryPotter         2024-01-15
   */
```

-- 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 concat(s.first_name,s.last_name)as student_name,s.student_id
from Students s
where student_id NOT IN(select s.student_id
from Students s left JOIN Payments p on s.student_id=p.student_id);

/*ouput
student_name   student_id
*/
```

-- 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_name
```

```
from Courses c
where c.course_id NOT IN (
select e.enrollment_id
from Courses c JOIN Enrollments e ON e.course_id=c.course_id);

/*ouput
        course_name
        Biology
*/
```

-- 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.

```
select concat(s.first_name,last_name) as student_name,s.student_id,c.course_id
from students s join enrollments e on s.student_id = e.student_id
                                    join courses c on c.course_id = e.course_id
group by s.student_id
having count(e.course_id)>1;

/* output:
   student_name     student_id   course_id

*/
```

-- 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 concat(t.first_name,t.last_name)as Teacher_name,t.last_name
FROM Teacher t
LEFT JOIN Courses c ON t.teacher_id = c.teacher_id
WHERE c.course_id IS NULL;

/*output:
        Teacher_name
```

```
        ArjitSingh
        LunaKalix
   */

-- task 4:

-- 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(s.student_id) as average_student,c.course_name
from students s join enrollments e on s.student_id = e.student_id
                                    join courses c on c.course_id = e.course_id
          group by c.course_name;
/*output:
        average_studentcourse_name
        3.5000  Art
        2.0000  English Literature
        1.0000  Mathematics
        5.0000  Music
   */

-- 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,max(amount) as highest_payment
from Payments
where amount in(select max(amount) from Payments);

/*output:
        student_id      highest_payment
        2       3700
   */
```

-- 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,c.course_id
from Courses c JOIN Enrollments e ON c.course_id=e.course_id
group by c.course_id
order by count(*) desc
limit 1;
```

```
/*output:
        course_name     course_id
        Art             4
   */
```

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

```
select sum(p.amount) as total_payment,concat(t.first_name,t.last_name) as teacher_name
from Payments p JOIN Students s on p.student_id=s.student_id
          JOIN Enrollments e on s.student_id=e.student_id
          JOIN Courses c on e.course_id=c.course_id
          JOIN Teacher t on c.course_id=t.teacher_id
          group by t.teacher_id;
```

```
/*output:
        total_payment   teacher_name
        1500    ArjitSingh
        3700    hermioneGranger
        2590    RonWeasley
        1200    LunaKalix
   */
```

-- 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 e.*
from Enrollments e
JOIN Courses c on e.course_id = c.course_id
where c.course_name=ALL(select course_name from Courses);
```

/*output:

empty ,no data

*/

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

```
select concat(t.first_name,t.last_name) as Teacher_name
from Teacher t
where t.teacher_id NOT IN(
select t.teacher_id
from Teacher t Join Courses c ON c.teacher_id=t.teacher_id
group by c.course_name);
```

/*output:

Teacher_name

ArjitSingh

LunaKalix

*/

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

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

/*

output

course_name

biology

*/

-- 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 sum(p.amount) as total_payment,s.student_id,c.course_name,e.enrollment_id
from Payments p JOIN Students s on p.student_id=s.student_id
        JOIN Enrollments e on s.student_id=e.student_id
        JOIN Courses c on e.course_id=c.course_id
        group by s.student_id,c.course_name;
```

/*

total_payment	student_id	course_name	enrollment_id
1500	1	Mathematics	1
3700	2	English Literature	2
1700	3	Art	6
890	4	Art	4
1200	5	Music	5

*/

-- 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
FROM Payments
```

```
GROUP BY student_id
HAVING COUNT(*) > 1;

/*output:
empty ,no data
*/
```

-- 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
student_id,
SUM(amount) AS total_payment
FROM Payments
GROUP BY student_id;

/*output:
```

student_id	total_payment
1	1500
2	3700
3	1700
4	890
5	1200

```
*/
```

-- 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 enrolled_students
FROM Courses c
```

```
LEFT JOIN Enrollments e ON c.course_id = e.course_id
GROUP BY c.course_id;

/*ouput:
        course_name     enrolled_students
        Mathematics             1
        English Literature      1
        Biology                 0
        Art                     2
        Music                   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;

/*output:
        average_payment_amount
        1798.0000
   */
```