Assignment – 2

Student Information System (SIS)

Task 1. Database Design:

1. Create the database named "SISDB"

```
create database 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 b. Courses c. Enrollments d. Teacher e. Payments

```
create table Students(  student_id int,
first_name text,
last_name  text,
date_of_birth date,
email varchar(100),
phone_number int,
primary key(student_id));
```

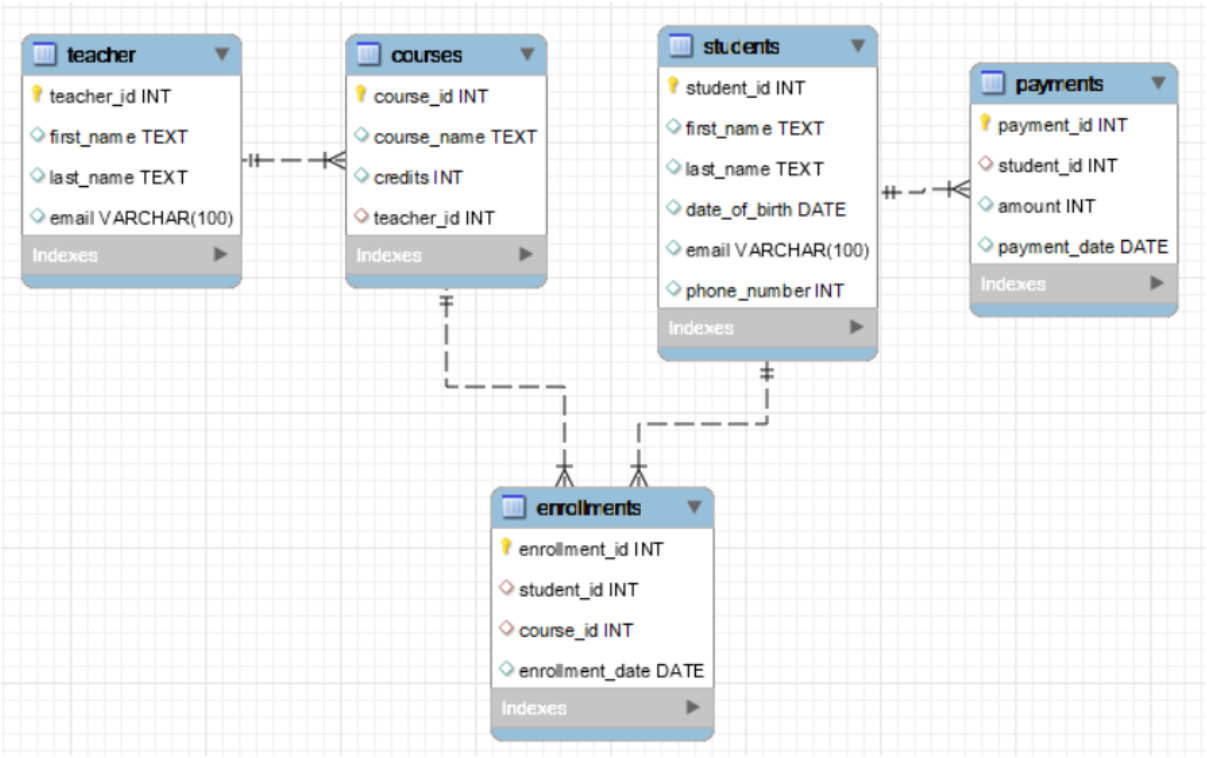
```
create table Teacher(teacher_id int,
first_name text,
last_name  text,
email  varchar(100),
primary key(teacher_id));
```

```
create table Courses(course_id int,
course_name text,
credits int,
teacher_id int,
primary key(course_id),
foreign key(teacher_id) references teacher(teacher_id));
```

```
create table Enrollments( enrollment_id int,
student_id int,
course_id int,
enrollment_date date,
primary key(enrollment_id),
foreign key(student_id) references Students(student_id),
foreign key(course_id) references Courses(course_id));
```

```
create table Payments( payment_id int,
student_id int,
amount     int,
payment_date date,
primary key(payment_id),
foreign key(student_id) references Students(student_id));
```

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



4. Insert at least 10 sample records into each of the following tables. i. Students ii. Courses iii. Enrollments iv. Teacher v. Payments

```
insert into Students(student_id, first_name, last_name, date_of_birth, email, phone_number)
values (1, 'John', 'Wick', '1990-05-15', 'john.wick@email.com', 12345879),
(2, 'Jane', 'Smith', '1992-08-22', 'jane.smith@email.com', 98765432),
(3, 'Alice', 'Johnson', '1991-12-03', 'alice.johnson@email.com', 55512378),
(4, 'Bob', 'Miller', '1993-06-28', 'bob.miller@email.com', 55598765),
(5, 'Eva', 'Jones', '1992-02-18', 'eva.jones@email.com', 55555512),
(6, 'Chris', 'Robinson', '1994-07-05', 'chris.robinson@email.com', 55555556),
(7, 'Sophia', 'Davis', '1990-09-10', 'sophia.davis@email.com', 55555590),
(8, 'Matthew', 'Taylor', '1993-11-25', 'matthew.taylor@email.com', 55555534),
(9, 'Olivia', 'Clark', '1995-04-30', 'olivia.clark@email.com', 55555578),
(10, 'Daniel', 'Wright', '1991-08-14', 'daniel.wright@email.com', 55555523);

insert into Enrollments(enrollment_id, student_id, course_id, enrollment_date) values
(1, 1, 101, '2024-01-14'),
(2, 2, 102, '2024-01-15'),
(3, 3, 103, '2024-01-16'),
(4, 4, 104, '2024-01-17'),
(5, 5, 105, '2024-01-24'),
(6, 6, 106, '2024-01-25'),
(7, 7, 107, '2024-01-26'),
(8, 8, 108, '2024-01-27'),
(9, 9, 109, '2024-01-28'),
(10, 10, 110, '2024-01-29');
```

```sql
insert into courses(course_id, course_name, credits, teacher_id) values
(101, 'Introduction to Programming', 3, 1),
(102, 'Database Management', 4, 2),
(103, 'Web Development', 4, 1),
(104, 'Data Analysis', 3, 2),
(105, 'Machine Learning Fundamentals', 4, 5),
(106, 'Digital Marketing Strategies', 3, 6),
(107, 'Graphic Design Principles', 2, 7),
(108, 'Project Management Basics', 3, 8),
(109, 'Ethical Hacking Techniques', 4, 9),
(110, 'Music Composition Workshop', 2, 10);

insert into Payments(payment_id, student_id, amount, payment_date) values
(1, 1, 500, '2024-01-20'),
(2, 2, 750, '2024-01-21'),
(3, 3, 600, '2024-01-22'),
(4, 4, 800, '2024-01-23'),
(5, 5, 450, '2024-02-01'),
(6, 6, 600, '2024-02-02'),
(7, 7, 300, '2024-02-03'),
(8, 8, 400, '2024-02-04'),
(9, 9, 550, '2024-02-05'),
(10, 10, 350, '2024-02-06');


insert into teacher(teacher_id, first_name, last_name, email) values
(1, 'Professor', 'Johnson', 'prof.johnson@email.com'),
(2, 'Dr.', 'Smith', 'dr.smith@email.com'),
(3, 'Ms.', 'Williams', 'ms.williams@email.com'),
(4, 'Prof.', 'Brown', 'prof.brown@email.com'),
(5, 'Dr.', 'Anderson', 'dr.anderson@email.com'),
(6, 'Professor', 'Taylor', 'prof.taylor@email.com'),
(7, 'Mrs.', 'Harrison', 'mrs.harrison@email.com'),
(8, 'Mr.', 'Baker', 'mr.baker@email.com'),
(9, 'Dr.', 'Mitchell', 'dr.mitchell@email.com'),
(10, 'Prof.', 'Turner', 'prof.turner@email.com');
```

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

```sql
insert into students(student_id, first_name, last_name, date_of_birth, email, phone_number)
values(11, "John","Deo","995-08-15","john.doe@example.com", 12345678);
```

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(11,11,101,'2024-01-17');
```

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 = "prof.john@gmail.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 enrollment_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 = 103;
```

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 = 700 where payment_id = 1;
```

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 students.first_name,students.last_name, SUM(payments.amount) as total_payments from students
left join payments on students.student_id = payments.student_id
GROUP BY students.student_id;
```

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) as total_students from enrollments
right join courses on enrollments.course_id = courses.course_id
GROUP BY courses.course_id;
```

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 Students.first_name, Students.last_name from Students
left join Enrollments on Students.student_id = Enrollments.student_id
where Enrollments.student_id is null;
```

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 Students.first_name, Students.last_name,courses.course_name from Students
left join Enrollments on Students.student_id = Enrollments.student_id
left join Courses on Enrollments.course_id = Courses.course_id;
```

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 teacher.first_name,teacher.last_name,courses.course_name from teacher
left join courses on teacher.teacher_id = courses.teacher_id;
```

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 students.first_name,students.last_name,enrollments.enrollment_date from students
left join  enrollments on students.student_id = enrollments.student_id
left join courses on courses.course_id = enrollments.course_id;
```

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 students.first_name,students.last_name,payments.student_id from students
left join payments on students.student_id = payments.student_id
where payments.student_id is null;
```

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 courses.course_id, courses.course_name from courses
join enrollments on courses.course_id = enrollments.course_id
where enrollments.course_id is null;
```

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 students.first_name,students.last_name from students
join enrollments on students.student_id = enrollments.student_id
group by students.student_id
having count(enrollments.course_id) > 1;
```

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 teacher.first_name,teacher.last_name from teacher
left join courses on teacher.teacher_id = courses.teacher_id
where courses.teacher_id is null;
```

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 course_id, avg(num_students) as avg_students
from (
    select Courses.course_id, count(Enrollments.student_id) as num_students
    from Courses
    left join Enrollments on Courses.course_id = Enrollments.course_id
    group by Courses.course_id
) as CourseEnrollments
group by course_id;
```

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 students.first_name,students.last_name  from students where student_id =
(select student_id from payments
 group by student_id
 order by max(amount ) desc
 limit 1);
```

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, MAX(enrollmentcount) as maxenrollments
from (
    select course_id, COUNT(student_id) as enrollmentcount
    from Enrollments
   group by course_id
) as course_enrollments
group by course_id;
```

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

```
select teacher_id, sum(amount) as totalpayments
from (
    select e.course_id, p.amount, c.teacher_id
    from Payments p
    join Enrollments e on p.student_id = e.student_id
    join Courses c on e.course_id = c.course_id
) as teacherpayments
group by teacher_id;
```

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(distinct course_id) as num_courses
from Enrollments
group by student_id
having num_courses = (select count(distinct course_id) from Courses);
```

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
);
```

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(year(curdate()) - year(date_of_birth)) as age
from students;
```

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 distinct course_id from enrollments
);
```

9. Calculate the total payments made by each student for each course they are enrolled in. Use subqueries and aggregate functions to sum payments.

```
154 •  select enrollments.student_id,enrollments.course_id,
155    (select course_name from courses where course_id = enrollments.course_id) as NameOfCourse,
156    (select sum(amount) from payments where student_id = enrollments.student_id and course_id = enrollments.course_id)
157    as totalPayments
158    from enrollments;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

student_id	course_id	NameOfCourse	totalPayments
2	102	Database Management	750
4	104	Data Analysis	800
5	105	Machine Learning Fundamentals	450
6	106	Digital Marketing Strategies	600
7	107	Graphic Design Principles	300
8	108	Project Management Basics	400
9	109	Ethical Hacking Techniques	550
10	110	Music Composition Workshop	350
11	101	Introduction to Programming	NULL

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.

```sql
select first_name,last_name from students 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.

```sql
select first_name, last_name,
(select sum(amount) from Payments where student_id = Students.student_id) as total_payments
from Students;
```

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.

```sql
select Courses.course_name, count(Enrollments.student_id) as numOfstudents
from Courses
left join Enrollments on Courses.course_id = Enrollments.course_id
group by Courses.course_name;
```

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.

```sql
select students.first_name,students.last_name,avg(payments.amount) as averagePayment from students
left join payments on students.student_id = payments.student_id
group by students.student_id;
```