# Assignments-2

A simplified Student Information System (SIS) database. The SIS database contains information about students, courses, and enrollments. Task is to perform various SQL operations on this database to retrieve and manipulate data.

## TASK - 1 DATABASE DESIGN

1) Create the database named 'SISDB'

```
mysql> CREATE DATABASE SISDB;
Query OK, 1 row affected (0.03 sec)
```

,

2 ) Define the schema for the students,Courses,Enrollment , Teachers and Payments table based on the provided schema .Write the SQLscript to create the mentioned tables with appropriate data type , constraints and relationship.

**Students**

```
mysql>  CREATE TABLE Students(
    -> student_id INT PRIMARY KEY,
    ->  first_name VARCHAR(25),
    ->  last_name VARCHAR(25),
    ->  date_of_birth DATE,
    ->  email VARCHAR(50),
    ->  phone_number BIGINT
    -> );
```

**Courses**

```
mysql> CREATE TABLE Courses(
    -> course_id INTEGER PRIMARY KEY,
    ->  course_name VARCHAR(50),
    ->  credits VARCHAR(50),
    ->  teacher_id INTEGER ,
    -> FOREIGN KEY(teacher_id) REFERENCES Teacher(teacher_id)
    -> );
```

**Enrollments**

```
mysql> 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)
    -> );
```

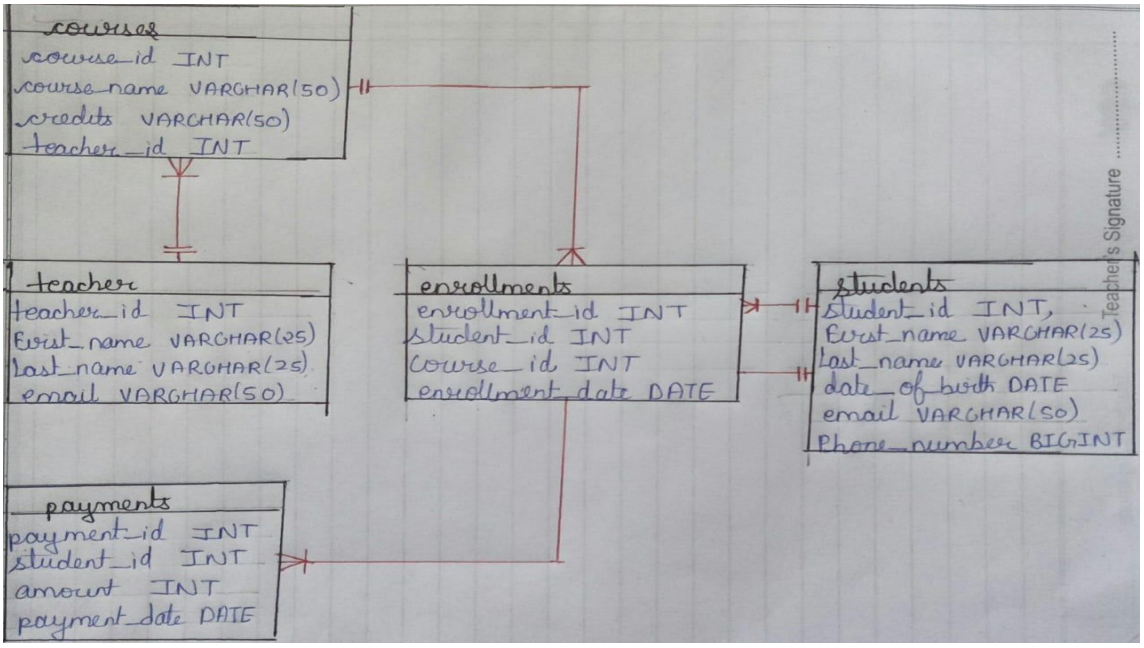
**Teacher**

```
mysql> CREATE TABLE Teacher(
    -> teacher_id INT PRIMARY KEY,
    -> first_name VARCHAR(25),
    -> last_name VARCHAR(25),
    -> email VARCHAR(50)
    -> );
```

**Payments**

```
ysql> CREATE TABLE Payments(
    -> payment_id INT PRIMARY KEY,
    -> student_id INT ,
    -> amount INTEGER,
    -> payment_date DATE,
    -> FOREIGN KEY(student_id) REFERENCES Students(student_id)
    -> );
```

3) create E-R(entity relationship ) diagram.



4) Insert at least 10 sample record into each of the following tables .

### Students

| student_id | first_name | last_name | date_of_birth | email | phone_number |
|---|---|---|---|---|---|
| 1 | John | Doe | 2000-05-15 | john@example.com | 9234567890 |
| 2 | Jane | Smith | 2001-08-25 | jane@example.com | 9876543210 |
| 3 | Alice | Johnson | 1999-11-03 | alice@example.com | 9551112222 |
| 4 | Bob | Anderson | 2002-04-18 | bob@example.com | 9998887777 |
| 5 | Eva | Garcia | 2003-09-21 | eva@example.com | 8443332222 |
| 6 | Michael | Brown | 2000-12-08 | michael@example.com | 7776665555 |
| 7 | Sophia | Lee | 2001-07-14 | sophia@example.com | 9223334444 |
| 8 | Oliver | Martinez | 1998-02-28 | oliver@example.com | 7667778888 |
| 9 | sheldon | cooper | 1998-01-12 | bazinga@gamil.com | 7574832312 |
| 10 | racheal | green | 1999-07-10 | racheal@gamil.com | 9574432215 |

### Courses

| course_id | course_name | credits | teacher_id |
|---|---|---|---|
| 301 | C and C++ | 4 | 101 |
| 302 | Core Java | 3 | 102 |
| 303 | Python | 3 | 103 |
| 304 | C# | 4 | 104 |
| 305 | Mysql | 5 | 105 |
| 306 | JavaScript | 3 | 106 |
| 307 | PHP | 4 | 107 |
| 308 | Kotlin | 4 | 108 |
| 309 | MariaDB | 3 | 109 |
| 400 | PostgreSQL | 4 | 110 |

### Enrollments

| enrollment_id | student_id | course_id | enrollment_date |
|---|---|---|---|
| 501 | 1 | 301 | 2023-09-01 |
| 502 | 2 | 302 | 2023-09-03 |
| 503 | 3 | 303 | 2023-09-05 |
| 504 | 4 | 304 | 2023-09-07 |
| 505 | 5 | 305 | 2023-09-09 |
| 506 | 6 | 306 | 2023-09-11 |
| 507 | 7 | 307 | 2023-09-13 |
| 508 | 8 | 308 | 2023-09-15 |
| 509 | 9 | 309 | 2023-09-17 |
| 600 | 10 | 400 | 2023-09-19 |

### Teacher

| teacher_id | first_name | last_name | email |
|---|---|---|---|
| 101 | Emily | Johnson | emily@example.com |
| 102 | Michael | Smith | michael@example.com |
| 103 | Sophia | Williams | sophia@example.com |
| 104 | Daniel | Brown | daniel@example.com |
| 105 | Olivia | Miller | olivia@example.com |
| 106 | David | Wilson | david@example.com |
| 107 | Emma | Anderson | emma@example.com |
| 108 | William | Martinez | william@example.com |
| 109 | Samantha | Garcia | samantha@example.com |
| 110 | Aiden | Lopez | aiden@example.com |

**Payments**

| payment_id | student_id | amount | payment_date |
|---|---|---|---|
| 901 | 1 | 500 | 2023-10-01 |
| 902 | 2 | 750 | 2023-10-03 |
| 903 | 3 | 600 | 2023-10-05 |
| 904 | 4 | 900 | 2023-10-07 |
| 905 | 5 | 400 | 2023-10-09 |
| 906 | 6 | 550 | 2023-10-11 |
| 907 | 7 | 800 | 2023-10-13 |
| 908 | 8 | 700 | 2023-10-15 |
| 909 | 9 | 950 | 2023-10-17 |
| 1000 | 10 | 350 | 2023-10-19 |

5) Create appropriate Primary Key and Foreign Key constraints for referential integrity.

**Primary key**

| TABLE_NAME | CONSTRAINT_NAME |
|---|---|
| teacher | PRIMARY |
| students | PRIMARY |
| courses | PRIMARY |
| enrollments | PRIMARY |
| payments | PRIMARY |

**Foreign Key**

| TABLE_NAME | CONSTRAINT_NAME |
|---|---|
| courses | courses_ibfk_1 |
| enrollments | enrollments_ibfk_1 |
| enrollments | enrollments_ibfk_2 |
| payments | payments_ibfk_1 |

## TASK - 2 SELECT , WHERE , BETWEEN , AND , LIKE

1) Write a SQL query to insert a new Student into the"Students" table with the following criteria.

```
ysql> insert into students values(11,'John','Doe','1995-08-15','Doe@example.com','123456780');
uery OK, 1 row affected (0.03 sec)
```

2) Write a SQl query to enroll a student in a course .Choose an existing student and course and insert a record into the “Enrollment”  table with enrollment date.

```
mysql> insert into enrollments values(601,1,304,'2023-12-10');
Query OK, 1 row affected (0.02 sec)
```

3) Update the email address of a specific teacher in a “Teacher” table.Choose any teacher and modify their email address.

```
mysql> update teacher set email = "johnson@example.com" where teacher_id  = 101;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

4) write a SQl query to delete a specific enrollment record from the “Enrollments” table.Select an enrollment record based on the student and course.

```
mysql> delete from enrollments where student_id  = 1 and course_id  = 304;
Query OK, 1 row affected (0.04 sec)
```

5) Update the “Courses” table to assign a specific teacher to a course .Choose any course and teacher from the respective table .

```
mysql> update Courses set teacher_id  = 110 where course_id  = 305;
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

6) Update the payment amount for a specific payment record in the “Payments” table .Choose any payment record and modify the payment amount.

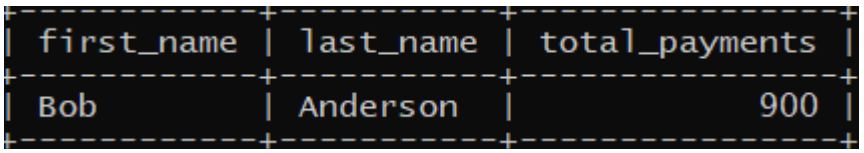
```
ysql> update payments set amount  = 400 where payment_id  = 1000;
uery OK, 1 row affected (0.02 sec)
ows matched: 1  Changed: 1  Warnings: 0
```

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

```
mysql> delete enrollments
    -> from enrollments
    -> join students on students.student_id  = enrollments.student_id where enrollments.student_id = 2;
Query OK, 1 row affected (0.04 sec)
```

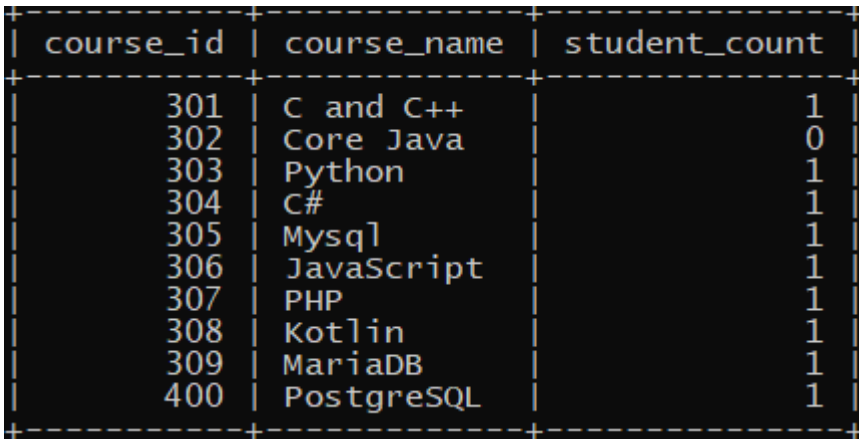
## TASK 3 - AGGREGATE FUNCTIONS, HAVING, ORDER BY, GROUP BY and JOINS:

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.



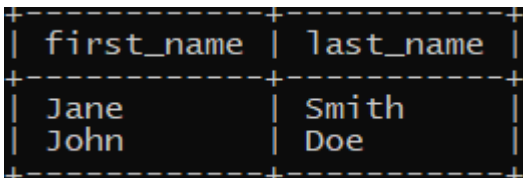
| first_name | last_name | total_payments |
|---|---|---|
| Bob | Anderson | 900 |

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.



| course_id | course_name | student_count |
|---|---|---|
| 301 | C and C++ | 1 |
| 302 | Core Java | 0 |
| 303 | Python | 1 |
| 304 | C# | 1 |
| 305 | Mysql | 1 |
| 306 | JavaScript | 1 |
| 307 | PHP | 1 |
| 308 | Kotlin | 1 |
| 309 | MariaDB | 1 |
| 400 | PostgreSQL | 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.



| first_name | last_name |
|---|---|
| Jane | Smith |
| John | Doe |

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.

| first_name | last_name | course_name |
|---|---|---|
| John | Doe | C and C++ |
| Alice | Johnson | Python |
| Bob | Anderson | C# |
| Eva | Garcia | Mysql |
| Michael | Brown | JavaScript |
| Sophia | Lee | PHP |
| Oliver | Martinez | Kotlin |
| sheldon | cooper | MariaDB |
| racheal | green | PostgreSQL |

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.

| first_name | last_name | course_name |
|---|---|---|
| John | Doe | C and C++ |
| Alice | Johnson | Python |
| Bob | Anderson | C# |
| Eva | Garcia | Mysql |
| Michael | Brown | JavaScript |
| Sophia | Lee | PHP |
| Oliver | Martinez | Kotlin |
| sheldon | cooper | MariaDB |
| racheal | green | PostgreSQL |

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.

| first_name | last_name | enrollment_date |
|---|---|---|
| Oliver | Martinez | 2023-09-15 |

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.

| first_name | last_name |
|---|---|
| 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.

| course_id | course_name |
|---|---|
| 302 | Core Java |

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.

```
mysql> SELECT DISTINCT e1.student_id, s.first_name, s.last_name
    -> FROM Enrollments e1
    -> JOIN Enrollments e2 ON e1.student_id = e2.student_id AND e1.course_id <> e2.course_id
    -> JOIN Students s ON e1.student_id = s.student_id;
Empty set (0.00 sec)
```

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.

| teacher_id | first_name | last_name |
|---|---|---|
| 105 | Olivia | Miller |

## TASK -4 SUBQUERY AND IT'S TYPES

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

| 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.

| student_id | first_name | last_name | highest_payment_amount |
|---|---|---|---|
| 9 | sheldon | cooper | 950 |

3) Retrieve a list of courses with the highest number of enrollments. Use subqueries to find the course(s) with the maximum enrollment count.

| course_id | course_name | max_enroll_count |
|---|---|---|
| 301 | C and C++ | 1 |
| 303 | Python | 1 |
| 304 | C# | 1 |
| 305 | Mysql | 1 |
| 306 | JavaScript | 1 |
| 307 | PHP | 1 |
| 308 | Kotlin | 1 |
| 309 | MariaDB | 1 |
| 400 | PostgreSQL | 1 |

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

| teacher_id | first_name | last_name | total_payments |
|---|---|---|---|
| 101 | Emily | Johnson | 500 |
| 102 | Michael | Smith | 0 |
| 103 | Sophia | Williams | 600 |
| 104 | Daniel | Brown | 900 |
| 105 | Olivia | Miller | 0 |
| 106 | David | Wilson | 550 |
| 107 | Emma | Anderson | 800 |
| 108 | William | Martinez | 700 |
| 109 | Samantha | Garcia | 950 |
| 110 | Aiden | Lopez | 800 |

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

```
Empty set (0.00 sec)
```

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

| teacher_id | first_name | last_name |
|---|---|---|
| 105 | Olivia | Miller |

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

| average_age |
|---|
| 23.3636 |

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

| course_id | course_name |
|---|---|
| 302 | Core Java |

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

| student_id | course_id | total_payments |
|---|---|---|
| 1 | 301 | 500 |
| 3 | 303 | 600 |
| 4 | 304 | 900 |
| 5 | 305 | 400 |
| 6 | 306 | 550 |
| 7 | 307 | 800 |
| 8 | 308 | 700 |
| 9 | 309 | 950 |
| 10 | 400 | 400 |

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.

Empty set (0.00 sec)

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.

| student_id | first_name | last_name | total_payments |
|---|---|---|---|
| 1 | John | Doe | 500 |
| 2 | Jane | Smith | 750 |
| 3 | Alice | Johnson | 600 |
| 4 | Bob | Anderson | 900 |
| 5 | Eva | Garcia | 400 |
| 6 | Michael | Brown | 550 |
| 7 | Sophia | Lee | 800 |
| 8 | Oliver | Martinez | 700 |
| 9 | sheldon | cooper | 950 |
| 10 | racheal | green | 400 |
| 11 | John | Doe | 0 |

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.

| course_name | student_count |
|---|---|
| C and C++ | 1 |
| Core Java | 0 |
| Python | 1 |
| C# | 1 |
| Mysql | 1 |
| JavaScript | 1 |
| PHP | 1 |
| Kotlin | 1 |
| MariaDB | 1 |
| PostgreSQL | 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.

| student_id | first_name | last_name | average_payment |
|---|---|---|---|
| 1 | John | Doe | 500.0000 |
| 2 | Jane | Smith | 750.0000 |
| 3 | Alice | Johnson | 600.0000 |
| 4 | Bob | Anderson | 900.0000 |
| 5 | Eva | Garcia | 400.0000 |
| 6 | Michael | Brown | 550.0000 |
| 7 | Sophia | Lee | 800.0000 |
| 8 | Oliver | Martinez | 700.0000 |
| 9 | sheldon | cooper | 950.0000 |
| 10 | racheal | green | 400.0000 |
| 11 | John | Doe | NULL |