**Student Information System :**

**Task 1. Database Design :**

**1. Create the database named "SISDB"**

**CREATE DATABASE** sisdb;

**USE** sisdb;

1. **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** **:**

**CREATE TABLE** student(student\_id **INT PRIMARY KEY AUTO\_INCREMENT**,

first\_name **VARCHAR**(20), last\_name **VARCHAR**(20), dob **DATE**,

email **VARCHAR**(30), phone\_number **VARCHAR**(20));

**b. Teacher :**

**CREATE TABLE** teacher(teacher\_id **INT PRIMARY KEY**,

first\_name **VARCHAR**(20), last\_name **VARCHAR**(20), email **VARCHAR**(40));

**c. Course**

**CREATE TABLE** course(course\_id **INT PRIMARY KEY**,

course\_name **VARCHAR**(40), credits **INT**,teacher\_id **INT**,

**FOREIGN KEY**(teacher\_id) **REFERENCES** teacher(teacher\_id) **ON DELETE CASCADE**);

**d. Enrollment :**

**CREATE TABLE** enrollment(enrollment\_id **INT PRIMARY KEY**, student\_id **INT**,

**FOREIGN KEY**(student\_id) **REFERENCES** student(student\_id) **ON DELETE CASCADE**,

course\_id **INT** , **FOREIGN KEY**(course\_id) **REFERENCES** course(course\_id) **ON DELETE CASCADE**,enrollment\_date **DATE**);

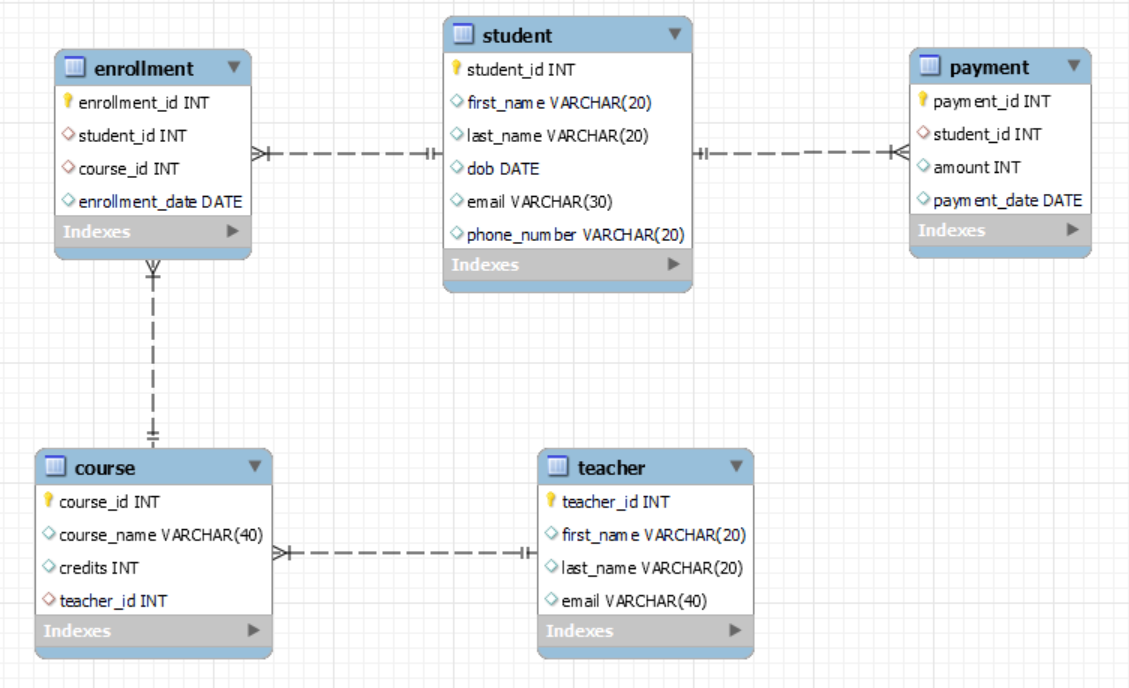
**e. Payment**

**CREATE TABLE** payment(payment\_id **INT PRIMARY KEY**, student\_id **INT**,

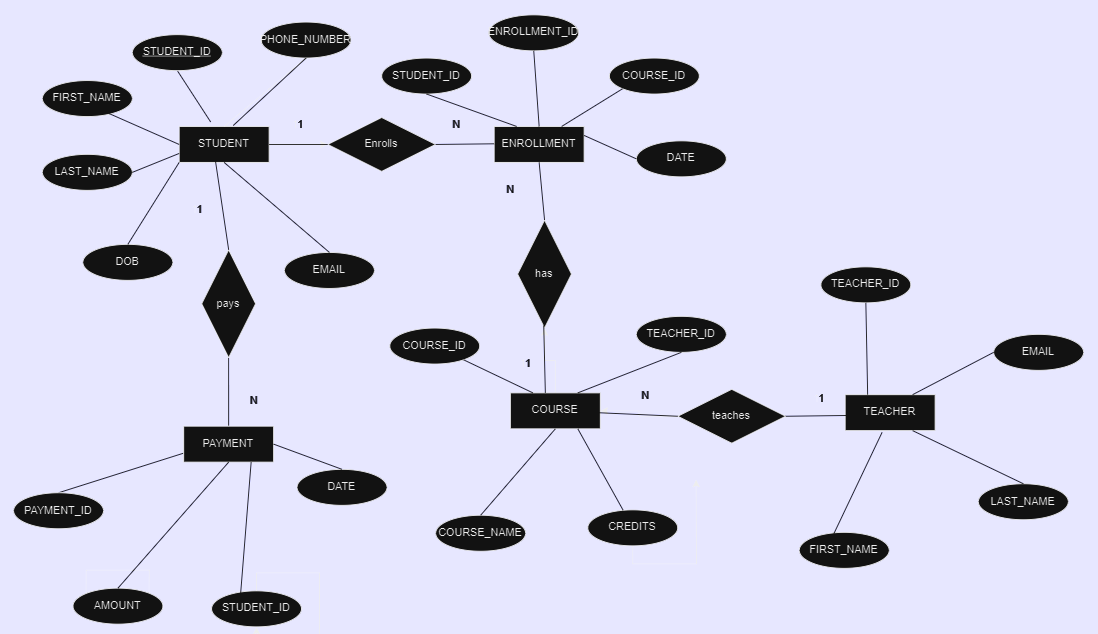
**FOREIGN KEY**(student\_id) **REFERENCES** student(student\_id) **ON DELETE CASCADE**,

amount **INT**, payment\_date **DATE**);

1

****

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

****

1. **Create appropriate Primary Key and Foreign Key constraints for referential integrity.**

**Ans :** Created appropriate keys during the table creation itself.

1. **Insert at least 10 sample records into each of the following tables.**
2. **Student :**

**INSERT INTO** student(first\_name, last\_name,dob,email,phone\_number) **VALUES**

('Nick','Joseph','2004-06-06','joseph@gamil.com','9876543210'),

('Robert','Frost','2004-09-11','robert@gmail.com','9876543211'),

('Prem', 'Kumar','2004-01-01', 'premkumar@gmail.com','9876543212'),

('Arjun', 'Chandra', '2004-02-04','arjun@gmail.com','9876543213'),

('Shiv', 'Kumar', '2004-03-03','shiv@gmail.com','9876543214'),

('Mahesh', 'Vardhan', '2004-09-09', 'mahesh@gmail.com','9876543215'),

('Praveen', 'varma', '2004-11-15', 'praveen@gamil.com', '9876543216'),

('Prasanth', 'kishore', '2004-10-26', 'prasanth@gmail.com', '9876543217'),

('Luke', 'Thomson', '2004-07-07', 'luke@gmail.com', '9876543218'),

('John', 'Peter', '2004-08-13', 'john@gmail.com','9876543219'),

('Raj','Patel','2004-01-09','raj@gmail.com','9876543220'),

('Maya','Singh','2004-06-01','mayasingh@gmail.com','9876543221'),

('Ankit','Verma','2004-11-10','ankit@gmail.com','9876543222'),

('Priya','Kapoor','2004-12-26','priyakapoor@gmail.com','9876543223'),

('Rahul','desai','2004-03-21','rahuldesai@gmail.com','9876543224'),

('Nandini','Reddy','2004-02-21','nandini@gmail.com','9876543225'),

('Arjun','Mehta','2004-03-03','arjunamehta@gmail.com','9876543226'),

('Simran','Khanna','2004-05-07','simrankhanna@gmail.com','9876543227'),

('Vikram','vardan','2004-05-4','vikramvardan@gmail.com','9876543228');

1. **Course :**

**INSERT INTO** course **VALUES**

(1, 'Introduction To Psychology',3,9),

(2, 'Calcus',4,8),

(3, 'English Compositioin',3,7),

(4, 'Organic Chemistry',4,6),

(5, 'Computer Science Fundamentals',3,5),

(6, 'World History : Ancient Civilizations',3,null),

(7, 'Environmental Science',4,3),

(8, 'Business Ethics',3,2),

(9, 'Spanish Language and Culture', 5, null),

(10, 'Marketing Principles',3,10);

1. **Enrollment :**

**INSERT INTO** enrollment **VALUES**

(1,19,1,'2023-11-05'),

(2,17,3,'2023-11-07'),

(3,15,5,'2023-11-08'),

(4,13,7,'2023-11-08'),

(5,11,8,'2023-11-09'),

(6,11,7,'2023-11-09'),

(7,9,2,'2023-11-10'),

(8,9,3,'2023-11-10'),

(9,7,3,'2023-11-11'),

(10,5,1,'2023-11-11'),

(11,3,1,'2023-11-12'),

(12,1,3,'2023-11-13'),

(13,2,5,'2023-11-14'),

(14,4,7,'2023-11-14'),

(15,6,3,'2023-11-14');

1. **Teacher :**

**INSERT INTO** teacher **VALUES**

(1,'Aryan','Gupta','aryan@gmail.com'),

(2,'Anjali','Kapoor','anjalikapoor@gmail.com'),

(3,'Rohit','Singh','rohitsingh@gmail.com'),

(4,'Priyanka','Sharma','priyankasharma@gmail.com'),

(5,'Karthik','Reddy','karthikreddy@gmail.com'),

(6,'Ananya','Kapoor','ananyakapoor@gmail.com'),

(7,'Rahul','Khanna','rahulkhanna@gmail.com'),

(8,'Meera','Patel','meerapatel@gmail.com'),

(9,'Siddharth','Verma','siddharth@gmail.com'),

(10,'Preeti','Malhotra','preetimalhotra@gmail.com');

1. **Payment**

**INSERT INTO** payment **VALUES**

(101,18,35000,'2023-10-12'),

(102,16,45000,'2023-10-15'),

(103,14,40000,'2023-10-15'),

(104,12,41000,'2023-10-17'),

(105,10,38000,'2023-10-18'),

(106,8,35000,'2023-10-19'),

(107,9,43000,'2023-10-21'),

(108,10,29000,'2023-10-22'),

(109,6,50000,'2023-10-22'),

(110,18,15000,'2023-10-22'),

(111,4,35000,'2023-10-25'),

(112,2,3000,'2023-10-26'),

(113,16,10000,'2023-10-26'),

(114,1,50000,'2023-10-27'),

(115,5,45000,'2023-10-29');

**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:**
2. **First Name: John**
3. **Last Name: Doe**
4. **c. Date of Birth: 1995-08-15**
5. **d. Email: john.doe@example.com**
6. **e. Phone Number: 1234567890**

**INSERT INTO** student(first\_name,last\_name,dob,email,phone\_number) **VALUES**

('John','Doe','1995-08-15','john.doe@example.com','1234567890');

1. **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** enrollment **VALUES**(016,20,1,'2023-11-15');

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

**UPDATE** teacher **SET** email = 'aryangupta@gmail.com' **WHERE** teacher\_id = 1;

1. **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** enrollment **WHERE** student\_id=16 **AND** course\_id = 1;

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

**UPDATE** course **SET** teacher\_id = 6 **WHERE** course\_id = 2;

1. **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** student **WHERE** student\_id = 17;

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

**UPDATE** payment **SET** amount = 33000 **WHERE** student\_id = 10;

**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** payment.student\_id,**CONCAT**(first\_name,student.last\_name) AS 'Name',

**SUM**(amount) 'Total Amount' **FROM** payment

**JOIN** student **ON** payment.student\_id = student.student\_id

**GROUP BY** student\_id

**ORDER BY** student\_id;

1. **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 'Course', count(enrollment\_id) 'Count'

**FROM** course c **JOIN** enrollment **USING**(course\_id)

**GROUP BY** course\_id;

1. **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**(s.first\_name,' ',s.last\_name) AS Name

**FROM** student s

**LEFT JOIN** enrollment **USING**(student\_id)

**WHERE** enrollment\_id is null

**GROUP BY** s.student\_id;

1. **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 'First Name', s.last\_name 'Last Name',

course.course\_name 'Course' **FROM** student s

**JOIN** enrollment **USING**(student\_id)

**JOIN** course **USING**(course\_id)

**ORDER BY** student\_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 CONCAT**(first\_name,' ',last\_name) 'Name Of Teacher',

course.course\_name 'Course'

**FROM** teacher **JOIN** course **USING**(teacher\_id)

**ORDER BY** 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 CONCAT**(first\_name,' ', last\_name) 'Name',

enrollment.enrollment\_date 'Date', course.course\_name 'Course' **FROM** student

**JOIN** enrollment **USING**(student\_id)

**JOIN** course **USING**(course\_id)

**ORDER BY** enrollment.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 CONCAT**(first\_name,' ',last\_name) 'Name' **FROM** student

**LEFT JOIN** payment **USING**(student\_id)

**WHERE** payment\_id **IS NULL**

**ORDER BY** student\_id;

1. **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** course\_id, course\_name **FROM** course

**LEFT JOIN** enrollment **USING**(course\_id)

**WHERE** enrollment.enrollment\_id **IS NULL**

**GROUP By** course\_id;

1. **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 DISTINCT** a.student\_id'Student Id' **FROM** enrollment a

**JOIN** enrollment b **ON** a.student\_id= b.student\_id

**WHERE** a.course\_id!=b.course\_id;

1. **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**(first\_name,' ',last\_name) 'Name' **FROM** teacher

**LEFT JOIN** course USING(teacher\_id)

**WHERE** course.course\_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 sub queries to achieve this.**

**SELECT** course\_id, **AVG**(student\_count) 'Average No.Of Student'

**FROM**

(**SELECT** course\_id, **COUNT**(DISTINCT student\_id) **AS** student\_count

**FROM** enrollment

**GROUP BY** course\_id) **AS** bquerry

**GROUP BY** course\_id;

1. **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, total\_amount 'Amount'

**FROM** (

**SELECT** student\_id, **SUM**(amount) **AS** total\_amount

**FROM** payment

**GROUP BY** student\_id

**HAVING SUM**(amount) = (**SELECT** MAX(total\_amount)

**FROM** (

**SELECT SUM**(amount) **AS** total\_amount

**FROM** payment

**GROUP BY** student\_id

) **AS** aquerry

)) **AS** bquerry;

1. **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\_of\_course

**FROM**(

**SELECT** course\_id, **COUNT**(course\_id) **AS** count\_of\_course

**FROM** enrollment

**GROUP BY** course\_id

**HAVING COUNT**(course\_id) =(

**SELECT** max(Course\_count) AS Max\_count FROM(

**SELECT** count(course\_id)

AS **Course**\_count **FROM** enrollment

GROUP BY course\_id) **AS** aquerry)

) **AS** bquerry;

1. **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(course\_payment) **AS** total\_payment

**FROM** (

**SELECT** t.teacher\_id, p.amount AS course\_payment

**FROM** payment p

**JOIN** enrollment e **ON** p.student\_id = e.student\_id

**JOIN** course c **ON** e.course\_id = c.course\_id

**JOIN** teacher t **ON** c.teacher\_id = t.teacher\_id

) **AS** payment\_per\_course

**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

**FROM** enrollment

**GROUP BY** student\_id

**HAVING COUNT**(DISTINCT course\_id) = (

**SELECT COUNT**(DISTINCT course\_id)

**FROM** course

);

**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, **CONCAT**(first\_name,' ',last\_name) **AS** Name

**FROM** teacher

**WHERE** teacher\_id **NOT IN** (

**SELECT DISTINCT** teacher\_id

**FROM** course

**WHERE** teacher\_id **IS NOT NULL**

);

**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**(age) **AS** average\_age

**FROM** (

**SELECT FLOOR**(DATEDIFF(CURDATE(), dob) / 365) **AS** age

**FROM** student

) **AS** age\_querry;

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

**SELECT** course\_id,course\_name **FROM** course

**WHERE** course\_id **NOT IN**(

**SELECT DISTINCT** course\_id **FROM** enrollment

**WHERE** course\_id **IS NOT NULL**);

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

**SELECT** student\_id, course\_id,(

**SELECT SUM**(amount)

**FROM** payment p

**WHERE** e.student\_id = p.student\_id) **AS** Payment

**FROM** enrollment e

**ORDER BY** student\_id;

**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** (

**SELECT** student\_id, **COUNT**(student\_id) **AS** payment\_count

**FROM** payment

**GROUP BY** student\_id

) **AS** bquerry

**WHERE** payment\_count > 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.**

**SELECT CONCAT**(first\_name,' ',last\_name) 'Name',**SUM**(amount)

**FROM** student

**JOIN** payment

**USING**(student\_id)

**GROUP BY** student\_id;

**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** course\_name,count(enrollment.course\_id) 'Count'

**FROM** course **JOIN** enrollment

**USING**(course\_id)

**GROUP BY** course\_id;

**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** Student.student\_id, **AVG**(Payment.amount) **AS** Average\_Amount

**FROM** Student

**JOIN** Payment **ON** Student.student\_id = Payment.student\_id

**GROUP BY** Student.student\_id

**ORDER BY** student\_id;