**Task 1. Database Design:**

**1. create the database named "SISDB”.**

CREATE DATABASE SISDB;

**2. Define the schema for the Students, Courses, Enrolments, 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),

email VARCHAR (30) UNIQUE,

phone\_number VARCHAR (10),

date\_of\_birth DATE

);

**b. Courses**

CREATE TABLE course (

course\_id INT PRIMARY KEY AUTO\_INCREMENT,

course\_name VARCHAR (30),

credits INT,

teacher\_id INT,

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

);

**c. Enrolments**

CREATE TABLE enrollment (

enrollment\_id INT PRIMARY KEY AUTO\_INCREMENT,

student\_id INT,

course\_id INT,

enrollment\_date DATE,

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

FOREIGN KEY (course\_id) REFERENCES course (course\_id) ON DELETE CASCADE

);

**d. Teacher**

CREATE TABLE teacher (

teacher\_id INT PRIMARY KEY AUTO\_INCREMENT,

first\_name VARCHAR(30),

last\_name VARCHAR(30),

email VARCHAR(30) UNIQUE

);

**e. Payments**

CREATE TABLE payment (

payment\_id INT PRIMARY KEY AUTO\_INCREMENT,

student\_id INT,

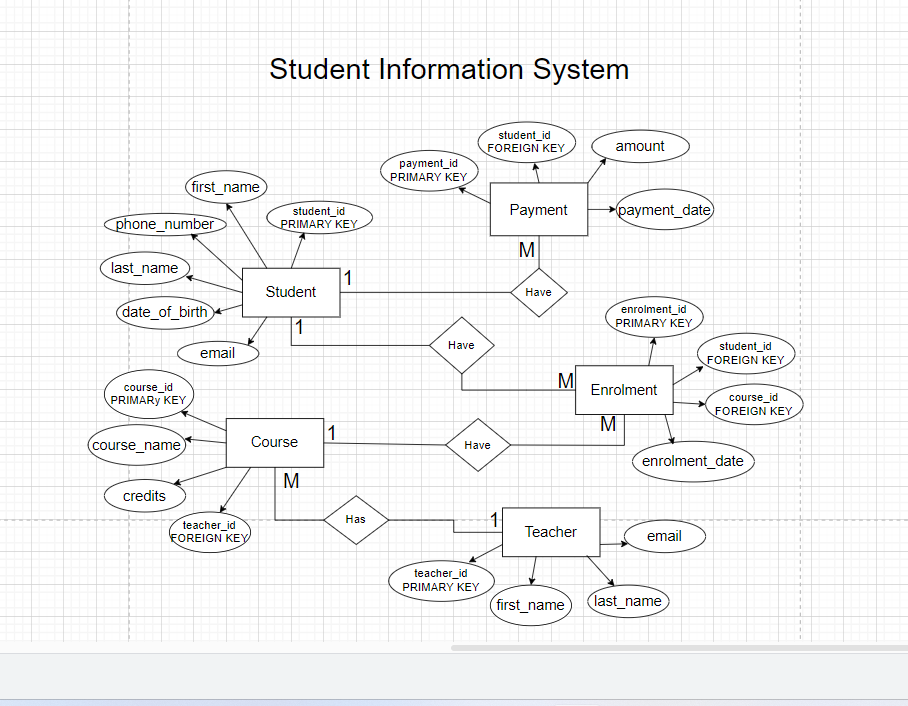
amount FLOAT,

payment\_date DATE,

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

);

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



**5. Insert at least 10 sample records into each of the following tables.**

**i. Students**

INSERT INTO student (

first\_name,last\_name,email,phone\_number,date\_of\_birth)

VALUES

("Nisha","Majhi","nishamajhi2020@gmail.com","9526656983","2001-09-20"),

("Venkata","Sai Teja","ventaka123@gmail.com","1234656983","2001-10-05"),

("Anurag","Kumar","anurag@gmail.com","9526656123","1999-01-20"),

("Sanket","jain","sanket@gmail.com","1226216983","2000-11-20"),

("Sai","Teja","sai@gmail.com","8996356983","1999-02-01");

**ii. Courses**

INSERT INTO course (

course\_name,credits,teacher\_id)

VALUES

("Java",5,2),

("Communication",2,1),

("JavaScript",4,5),

("Node",2,3),

("Express",2,4);

**iii. Enrolments**

INSERT INTO enrolment (

student\_id,course\_id,enrollment\_date)

VALUES

(1,1,"2023-11-16"),

(1,4,"2023-01-01"),

(2,2,"2023-10-10"),

(3,1,"2023-11-16"),

(4,1,"2023-11-16"),

(5,1,"2023-11-16"),

(2,5,"2023-10-01");

**iv. Teacher**

INSERT INTO teacher (

first\_name,last\_name,email)

VALUES

("Kamal","Pardesi","kamal@gmail.com"),

("Bala","Chandra","bala@gmail.com"),

("mahesh","babu","mahesh@gmail.com"),

("Allu","Arjun","allu@gmail.com"),

("Vijay","Thalapaty","vijay@gmail.com");

**v. Payments**

INSERT INTO payment (

student\_id,amount,payment\_date)

VALUES

(1,5000.50,"2023-06-01"),

(2,10000,"2019-06-20"),

(1,5000,"2023-07-01"),

(4,40000,"2022-11-11");

**Tasks 2: Select, Where, Between, LIKE, AND:**

**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](mailto:john.doe@example.com)

e. Phone Number: 1234567890

**Query –**

INSERT INTO student (

first\_name,last\_name,email,phone\_number,date\_of\_birth)

VALUES

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

**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 enrollment(

student\_id,course\_id,enrollment\_date)

VALUES

(1,1,"2023-12-01");

**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="abc@gmail.com" where teacher\_id=5

**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 enrollment WHERE enrollment\_id=1;

**5. 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=2 where course\_id=5;

**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 student WHERE student\_id=6;

**7. 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=99 WHERE payment\_id=1

**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.**

SELECT DISTINCT CONCAT(s.first\_name," ",s.last\_name) AS Name,

sum(p.amount)

FROM student s JOIN payment p

ON s.student\_id = p.student\_id

GROUP BY p.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 c.course\_name as course,count(e.course\_id) AS count

FROM course c JOIN enrollment e

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

GROUP BY e.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 DISTINCT CONCAT (first\_name, " ", last\_name) AS Name

FROM student

WHERE student\_id NOT IN (

SELECT student\_id

FROM enrollment

);

SELECT CONCAT (s.first\_name, " ", s.last\_name) AS Name

FROM student s LEFT JOIN enrollment e

ON s.student\_id = e.student\_id

WHERE e.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 s.first\_name,s.last\_name,c.course\_name

FROM enrollment e JOIN student s

ON e.student\_id = s.student\_id

JOIN course c

ON e.course\_id = c.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 CONCAT (t.first\_name," ",t.last\_name) AS Name,c.course\_name

FROM teacher t JOIN course c

ON t.teacher\_id = c.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 (s.first\_name," ",s.last\_name) AS Name,c.course\_name AS course,e.enrollment\_date

FROM enrollment e JOIN student s

ON e.student\_id = s.student\_id

JOIN course c

ON e.course\_id = c.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 student\_id,first\_name,last\_name

FROM student WHERE student\_id NOT IN(

SELECT student\_id FROM payment);

SELECT s.student\_id, s.first\_name,s.last\_name

FROM student s LEFT JOIN payment p

ON s.student\_id = p.student\_id

WHERE p.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 course\_id,course\_name

FROM course

WHERE course\_id NOT IN(

SELECT course\_id FROM enrollment);

SELECT c.course\_id, c.course\_name

FROM course c LEFT JOIN enrollment e

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

WHERE e.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 student\_id,COUNT(course\_id)

FROM enrollment

GROUP BY student\_id

HAVING COUNT(course\_id) > 1;

SELECT e1.student\_id AS StudentId, COUNT(e1.course\_id) AS NumberOfStudent

FROM enrollment e1 JOIN enrollment e2

ON e1.student\_id=e2.student\_id AND e1.course\_id<>e2.course\_id

GROUP BY e1.student\_id

HAVING COUNT(e1.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 t.teacher\_id,CONCAT(t.first\_name," ",t.last\_name) AS teacherName

FROM teacher t LEFT JOIN course c

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

WHERE c.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(student\_count) AS average\_students\_per\_course

FROM (

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

FROM enrollment

GROUP BY course\_id

) AS course\_counts

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 student\_id

FROM payment where amount = (

SELECT MAX(amount)

FROM payment

);

**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, COUNT(\*) AS enrollment\_count

FROM enrollment

GROUP BY course\_id

HAVING COUNT(\*) = (

SELECT MAX(enrollment\_count)

FROM (

SELECT COUNT(\*) AS enrollment\_count

FROM enrollment

GROUP BY course\_id

) AS counts

);

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

SELECT enrollment.course\_id, course.course\_name, COUNT(\*) AS enrollment\_count

FROM enrollment

JOIN course ON enrollment.course\_id = course.course\_id

GROUP BY course\_id, course\_name

HAVING COUNT(\*) = (

SELECT MAX(enrollment\_count)

FROM (

SELECT COUNT(\*) AS enrollment\_count

FROM enrollment

GROUP BY course\_id

) AS counts

);

**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, first\_name, last\_name

FROM student

WHERE student\_id IN (

SELECT e.student\_id

FROM enrollment e

GROUP BY e.student\_id

HAVING COUNT(DISTINCT e.course\_id) = (

SELECT COUNT(\*)

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, first\_name, last\_name

FROM teacher

WHERE teacher\_id NOT IN (

SELECT DISTINCT teacher\_id

FROM course

);

**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 TIMESTAMPDIFF(YEAR, date\_of\_birth, CURDATE()) AS age

FROM student

) AS student\_ages;

**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

);

**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 e.student\_id, e.course\_id,

COALESCE(SUM(p.amount), 0) AS total\_payments\_for\_course

FROM enrollment e

LEFT JOIN payment p ON e.student\_id = p.student\_id

GROUP BY e.student\_id, e.course\_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 payment

GROUP BY student\_id

HAVING 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 s.student\_id, s.first\_name, s.last\_name, COALESCE(SUM(p.amount), 0) AS total\_payments

FROM student s

LEFT JOIN payment p ON s.student\_id = p.student\_id

GROUP BY s.student\_id, s.first\_name, s.last\_name;

**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\_id, c.course\_name, COUNT(e.student\_id) AS enrolled\_students\_count

FROM course c

LEFT JOIN enrollment e ON c.course\_id = e.course\_id

GROUP BY c.course\_id, c.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.**

SELECT s.student\_id, s.first\_name, s.last\_name, AVG(p.amount) AS average\_payment\_amount

FROM student s

LEFT JOIN payment p ON s.student\_id = p.student\_id

GROUP BY s.student\_id, s.first\_name, s.last\_name;