**STUDENT INFORMATION SYSTEM**

**Task-1: Database Design**

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

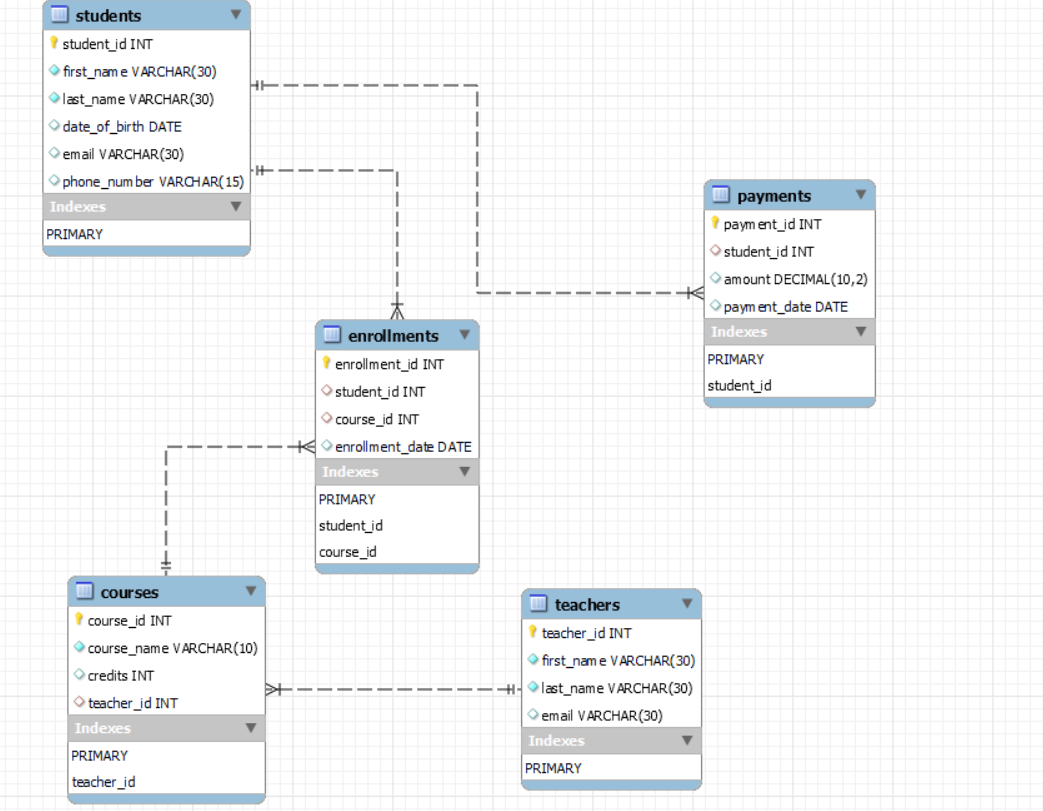
**Ans.**

CREATE DATABASE SISDB;

USE 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.**

**Ans:**

****

**Students table:**

CREATE TABLE students (

student\_id INT PRIMARY KEY,

first\_name VARCHAR(30) NOT NULL,

last\_name VARCHAR(30) NOT NULL,

date\_of\_birth DATE,

email VARCHAR(30),

phone\_number VARCHAR(15)

);

**Courses table:**

CREATE TABLE courses (

course\_id INT PRIMARY KEY,

course\_name VARCHAR(10) NOT NULL,

credits INT,

teacher\_id INT,

FOREIGN KEY(teacher\_id) REFERENCES teachers(teacher\_id)

);

**Enrollments table:**

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)

);

**Teachers table:**

CREATE TABLE teachers (

teacher\_id INT PRIMARY KEY,

first\_name VARCHAR(30) NOT NULL,

last\_name VARCHAR(30) NOT NULL,

email VARCHAR(30)

);

**Payments table:**

CREATE TABLE payments (

payment\_id INT PRIMARY KEY,

student\_id INT,

amount DECIMAL(10,2),

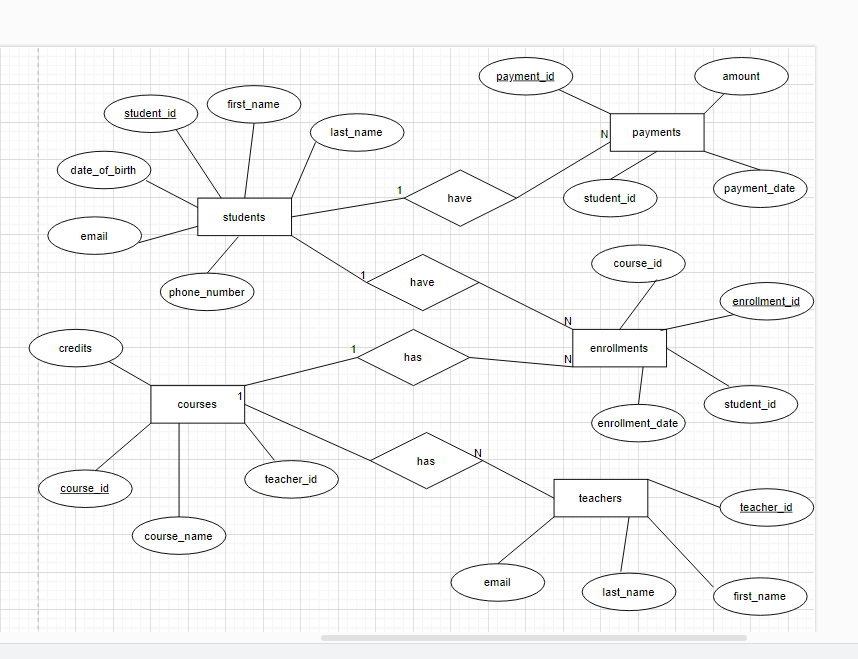
payment\_date DATE,

FOREIGN KEY(student\_id) REFERENCES students(student\_id)

);

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

**Ans.**

****

**Entity:** Students

Teachers

Courses

Enrollments

Payments

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

**Ans:**

Students Table:

student\_id INT PRIMARY KEY

Teachers Table:

teacher\_id INT PRIMARY KEY

Courses Table:

course\_id INT PRIMARY KEY,

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

Enrollments Table:

enrollment\_id INT PRIMARY KEY,

FOREIGN KEY(student\_id) REFERENCES students(student\_id),

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

Payments Table:

payment\_id INT PRIMARY KEY,

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

For the Students table there is a Primary key.

For the teachers table there is a Primary Key.

For the course table there are references to the teacher table.

For the enrollment table there references to the student table and course table.

For the payments table there are references to the student table.

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

**Ans:**

**i. Students:**

INSERT INTO students VALUES

(1,'Vamsi','Muppana','2002-09-27','vamsi@gmail.com', '9898912345'),

(2,'Luffy','MonkeyD','2001-02-20','luffy@gmail.com', '5498911935'),

(3,'Zoro','Roronoa','2000-07-07','zoro@gmail.com', '7898912345'),

(4,'Sanji','Vinsmoke','2002-08-24','sanji@gmail.com', '6898912345'),

(5,'Naruto','Uzumaki','2001-04-11','naruto@gmail.com', '9998912345'),

(6,'Kakashi','Hatake','2000-06-23','kakashi@gmail.com', '8888912345'),

(7,'Gojo','Satoru','1999-03-27','gojo@gmail.com','6677912345'),

(8,'Yuji','Itadori','2002-05-11','yuji@gmail.com','7788912345'),

(9,'Sasuke','Uchiha','2001-04-10','sasuke@gmail.com','8934012345'),

(10,'Eren','Yeager','2000-10-11','eren@gmail.com','9837912345');

**ii. Courses**

INSERT INTO courses VALUES

(1001,'CS',3,101),

(1002,'ALGORITHMS',3,102),

(1003,'DBMS',3,103),

(1004,'DSA',4,104),

(1005,'JAVA',3,105),

(1006,'PYTHON',4,106),

(1007,'SQL',3,107),

(1008,'C#',4,108),

(1009,'WEB TECH',4,109),

(1010,'UML',3,110);

**iii. Enrollments**

INSERT INTO Enrollments VALUES

(201, 1, 1001, '2023-01-01'),

(202, 2, 1002, '2023-02-01'),

(203, 3, 1003, '2023-03-01'),

(204, 4, 1004, '2023-04-01'),

(205, 5, 1005, '2023-05-01'),

(206, 6, 1006, '2023-06-01'),

(207, 7, 1007, '2023-07-01'),

(208, 8, 1008, '2023-08-01'),

(209, 9, 1009, '2023-09-01'),

(210,10, 1010,'2023-10-01');

**iv. Teacher**

INSERT INTO teachers VALUES

(101,'ab','cd','abc@gmail.com'),

(102,'cda','wed','wed@gmail.com'),

(103,'gta','ihs','ihs@gmail.com'),

(104,'mvk','urw','urw@gmail.com'),

(105,'twy','Uim','uim@gmail.com'),

(106,'Kai','ahe','ahe@gmail.com'),

(107,'ers','sis','sis@gmail.com'),

(108,'hih','qwe','qwe@gmail.com'),

(109,'lop','ipl','ipl@gmail.com'),

(110,'fcv','bns','bsn@gmail.com');

**v. Payments**

INSERT INTO Payments VALUES

(301, 1, 100.00, '2023-01-15'),

(302, 2, 150.50, '2023-02-20'),

(303, 3, 75.25, '2023-03-25'),

(304, 4, 200.75, '2023-04-30'),

(305, 5, 300.75, '2023-05-27'),

(306, 6, 500.85, '2023-06-18'),

(307, 7, 859.75, '2023-07-31'),

(308, 8, 456.75, '2023-08-29'),

(309, 9, 320.75, '2023-09-07'),

(310, 10, 890.75,'2023-10-10');

**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**

**Ans.**

INSERT INTO students (student\_id,first\_name,last\_name, date\_of\_birth, email, phone\_number) VALUES

(11,'John','Doe','1995-08-15','john.doe@gmail.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.**

**Ans.**

INSERT INTO enrollments (enrollment\_id, student\_id, course\_id, enrollment\_date) VALUES

(211,1,1005,'2023-12-08');

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

**Ans.**

UPDATE teachers

SET email='url@gmail.com'

WHERE teacher\_id=105;

**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.**

**Ans.**

DELETE FROM enrollments

WHERE student\_id=2

AND course\_id=1002;

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

**Ans.**

UPDATE courses

SET teacher\_id=101

WHERE course\_id=1001;

**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.**

**Ans.**

DELETE FROM enrollments

WHERE student\_id=209;

DELETE FROM payments

WHERE student\_id = 9;

DELETE FROM students

WHERE student\_id=9;

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

**Ans.**

UPDATE payments

SET amount=527.45

WHERE payment\_id=303;

**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.**

**Ans.**

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

FROM students s

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

WHERE s.student\_id=5

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

**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.**

**Ans.**

SELECT c.course\_id,c.course\_name, COUNT(e.student\_id) AS enrolled\_student\_count

FROM courses c

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

GROUP BY c.course\_id, c.course\_name;

**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.**

**Ans.**

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

FROM students s

LEFT JOIN enrollments 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.**

**Ans.**

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 c.course\_id=e.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.**

**Ans.**

SELECT t.first\_name,t.last\_name,c.course\_name

FROM teachers t

JOIN courses 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.**

**Ans.**

SELECT s.student\_id,s.first\_name,s.last\_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="DSA";

**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.**

**Ans.**

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

FROM students s

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

WHERE p.payment\_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.**

**Ans.**

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

FROM courses c

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

WHERE e.enrollment\_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.**

**Ans.**

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

FROM enrollments e1

JOIN enrollments e2 ON e1.student\_id= e2.student\_id AND e1.enrollment\_id <>e2.enrollment\_id

JOIN students s ON e1.student\_id=s.student\_id

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

HAVING COUNT(DISTINCT 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.**

**Ans.**

SELECT t.teacher\_id,t.first\_name,t.last\_name

FROM teachers t

LEFT JOIN courses c ON t.teacher\_id= c.teacher\_id

WHERE c.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 subqueries to achieve this.**

**Ans.**

SELECT AVG(enrolled\_student\_count) AS avg\_students\_per\_course

FROM (

SELECT c.course\_id, COUNT(e.student\_id) AS enrolled\_student\_count

FROM courses c

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

GROUP BY c.course\_id) AS course\_counts;

**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.**

**Ans.**

SELECT s.student\_id,s.first\_name,s.last\_name,p.amount AS maximum\_amount

FROM students s

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

WHERE p.amount= (SELECT MAX(amount) FROM payments);

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

**Ans.**

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

FROM courses c

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

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

HAVING COUNT(e.student\_id)=( SELECT MAX(enrolled\_count)

FROM

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

FROM enrollments

GROUP BY course\_id)

AS course\_counts);

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

**Ans.**

SELECT teacher\_id,first\_name,last\_name,

(

SELECT COALESCE(SUM(p.amount), 0)

FROM payments p

WHERE p.student\_id IN

(

SELECT e.student\_id FROM enrollments e

WHERE e.course\_id IN

(

SELECT course\_id FROM courses c

WHERE c.teacher\_id = teachers.teacher\_id)

)

) AS total\_payments

FROM teachers;

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

**Ans.**

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

FROM students s

WHERE (SELECT COUNT(DISTINCT e.course\_id) FROM enrollments e) =

(SELECT COUNT(DISTINCT e2.course\_id) FROM enrollments e2

WHERE e2.student\_id = s.student\_id);

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

**Ans.**

SELECT t.teacher\_id,t.first\_name,t.last\_name

FROM teachers t

WHERE

NOT EXISTS ( SELECT 1 FROM courses c

WHERE c.teacher\_id=t.teacher\_id);

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

**Ans.**

SELECT AVG(student\_age) AS avergae\_age

FROM ( SELECT TIMESTAMPDIFF( YEAR, date\_of\_birth,CURDATE()) AS student\_age

FROM students)

AS student\_ages;

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

**Ans.**

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

FROM courses c

WHERE

NOT EXISTS ( SELECT 1 FROM enrollments e

WHERE e.course\_id= c.course\_id);

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

**Ans.**

SELECT s.student\_id,s.first\_name,s.last\_name,e.course\_id,c.course\_name,

COALESCE

(

(SELECT SUM(p.amount) FROM payments p WHERE p.student\_id = s.student\_id),0

)

AS total\_payments

FROM students s

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

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

**Ans.**

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

FROM students s

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

FROM payments

GROUP BY student\_id

HAVING payment\_count > 1

) p\_counts ON s.student\_id = p\_counts.student\_id;

**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.**

**Ans.**

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

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

FROM students s

LEFT JOIN payments 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.**

**Ans.**

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

FROM courses c

LEFT JOIN enrollments 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.**

**Ans.**

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

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

FROM students s

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

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