**ASSIGNMENT ON**

**STUDENT INFORMATION SYSTEM**

SAI TEJA BHEEMIREDDY

**Task 1. Database Design:**

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

create database sisds;

show databases;

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 b. Courses c. Enrollments d. Teacher e.**

**Payments**

**create table Students**(

student\_id int primary key,

first\_name varchar(20) not null,

last\_name varchar(20) not null,

date\_of\_birth date not null,

email varchar(50) unique,

phone\_number bigint not null

)

**create table Courses**(

course\_id int Primary Key,

course\_name varchar(40) not null,

credits int,

teacher\_id int,

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

)

**create table Enrollments**(

enrollment\_id int Primary Key,

student\_id int not null,

FOREIGN KEY (student\_id) REFERENCES Students(student\_id), course\_id int not null,

FOREIGN KEY (course\_id) REFERENCES Courses(course\_id),

enrollment\_date date not null

)

**create table Teacher**(

teacher\_id int Primary Key,

first\_name varchar(20) not null,

last\_name varchar(20) not null,

email varchar(50) unique

)

**create table Payments**(

payment\_id int Primary Key,

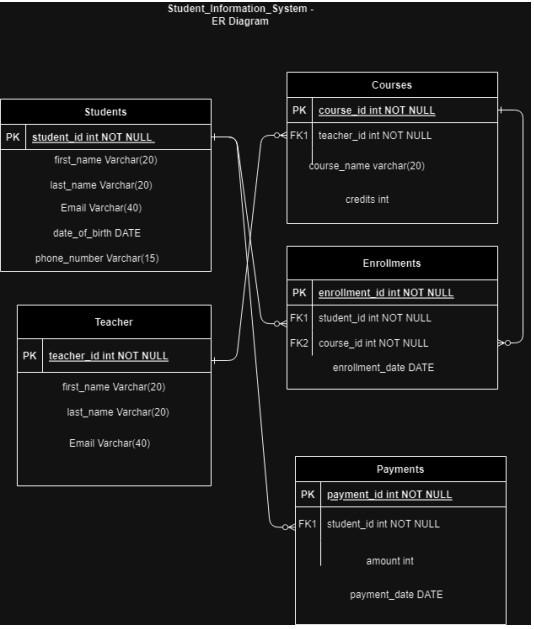
student\_id int not null,

FOREIGN KEY (student\_id) REFERENCES Students(student\_id), amount int ,

payment\_date date not null

)

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



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

student\_id int primary key

course\_id int Primary Key

enrollment\_id int Primary Key

teacher\_id int Primary Key

payment\_id int Primary Key

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

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

FOREIGN KEY (course\_id) REFERENCES Courses(course\_id)

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

1. **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 values** (1,'A','B','2002-12-14','A.B@gmail.com',9999999999), (2,'C','D','2001-12-14','C.D@gmail.com',8888888888), (3,'E','F','2000-12-14','E.F@gmail.com',7777777777), (4,'G','H','2002-11-07','G.H@gmail.com',6666666666), (5,'I','J','2001-11-07','I.J@gmail.com',5555555555), (6,'K','L','2000-11-07','K.L@gmail.com',4444444444), (7,'M','N','2002-03-02','M.N@gmail.com',3333333333), (8,'O','P','2001-03-02','O.P@gmail.com',2222222222), (9,'Q','R','2000-03-02','Q.R@gmail.com',1111111111), (10,'S','T','2002-01-03','S.T@gmail.com',1112223330) ;

**insert into Courses values**

(1,'OS',3,1),

(2,'DSA',4,2),

(3,'COMPUTER NETWORKS',3,3),

(4,'COMPUTER VISION',3,4),

(5,'DATA ANALYTICS',3,5),

(6,'BLOCKCHAIN',4,3),

(7,'ML',3,1),

(8,'ROBOTICS',4,6),

(9,'C++',4,7),

(10,'THEORY OF AUTOMATA',4,3)

;

**insert into Payments values**

(1,1,40000,'2023-01-15'),

(2,2,35000,'2023-02-10'),

(3,3,30000,'2023-03-05'),

(4,4,45000,'2023-04-01'),

(5,5,50000,'2023-05-15'),

(6,6,35000,'2023-06-10'),

(7,7,40000,'2023-07-05'),

(8,8,55000,'2023-08-01'),

(9,9,60000,'2023-09-15'),

(10,10,70000,'2023-10-10')

;

**insert into Teacher values**

(1,'Amit','Kumar','amit.kumar@gmail.com'),

(2,'Priya','Patel','priya.patel@gmail.com'),

(3,'Raj','Sharma','raj.sharma@gmail.com'),

(4,'Deepak','Gupta','deepak.gupta@gmail.com'),

(5,'Anita','Singh','anita.singh@gmail.com'),

(6,'Vijay','Yadav','vijay.yadav@gmail.com'),

(7,'Pooja','Mishra','pooja.mishra@gmail.com'),

(8,'Rahul','Verma','rahul.verma@gmail.com'),

(9,'Neha','Malik','neha.malik@gmail.com'),

(10,'Arjun','Das','arjun.das@gmail.com')

;

**insert into Enrollments values**

(1,7,5,'2023-07-05'),

(2,5,2,'2023-07-05'),

(3,5,9,'2023-07-05'),

(4,4,10,'2023-07-05'),

(5,4,1,'2023-07-05'),

(6,4,4,'2023-07-05'),

(7,9,6,'2023-07-05'),

(8,9,2,'2023-07-05'),

(9,10,4,'2023-03-17'),

(10,10,3,'2023-03-17'),

(11,10,1,'2023-03-17'),

(12,8,2,'2023-03-17'),

(13,8,9,'2023-03-17'),

(14,8,7,'2023-03-17'),

(15,1,6,'2023-03-17'),

(16,1,10,'2023-03-17'),

(17,1,1,'2023-01-23'),

(18,6,5,'2023-01-23'),

(19,6,4,'2023-01-23'),

(20,6,3,'2023-01-23'),

(21,2,8,'2023-01-23'),

(22,2,7,'2023-01-23'),

(23,2,10,'2023-01-01'),

(24,3,8,'2023-01-01')

;

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

**e. Phone Number: 1234567890**

**QUERY:** INSERT INTO Students (first\_name, last\_name, date\_of\_birth, email, phone\_number) VALUES ('John', 'Doe', '1995-08-15', 'john.doe@example.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.**

**QUERY:** INSERT INTO Enrollments (student\_id, course\_id, enrollment\_date) VALUES ((SELECT student\_id FROM Students WHERE first\_name = 'John' AND last\_name = 'Doe'), (SELECT course\_id FROM Courses WHERE course\_name = 'Data Analytics'), '2023-11-28’ );

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

**QUERY:**UPDATE Teacher SET email = "amith.kumar@gmail.com" where first\_name = "amit";

**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.**

**QUERY:**DELETE FROM Enrollments WHERE student\_id = (SELECT student\_id FROM Students WHERE first\_name = 'John' AND last\_name = 'Doe') AND course\_id = (SELECT course\_id FROM Courses WHERE course\_name = 'Data Analytics');

**5.Update the "Courses" table to assign a specific teacher to a course.**

**Choose any course and teacher from the respective tables.**

**QUERY:**UPDATE Courses SET teacher\_id = (SELECT teacher\_id FROM Teacher WHERE first\_name = 'Amit' AND last\_name = 'Kumar') WHERE course\_name = 'OS';

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.**

**QUERY:**DELETE FROM Enrollments WHERE student\_id IN (SELECT student\_id FROM Students WHERE first\_name = 'John' AND last\_name = 'Doe'); DELETE FROM Students WHERE first\_name = 'John' AND last\_name = 'Doe’;

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

**QUERY:**UPDATE payments set amount = 100000 where payment\_id = 2;

**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.**

**QUERY:**SELECT s.first\_name, s.last\_name, SUM(p.amount) FROM Students s JOIN Payments p ON s.student\_id = p.student\_id WHERE s.student\_id = 1;

**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.**

**QUERY:**SELECT c.course\_id, c.course\_name, COUNT(e.student\_id) 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.**

**QUERY:**SELECT 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.**

**QUERY:**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 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.**

**QUERY:**SELECT t.first\_name, t.last\_name, c.course\_name FROM Teacher 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.**

**QUERY:**SELECT c.course\_name, 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

ORDER BY c.course\_name;

**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.**

**QUERY:**SELECT CONCAT(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.**

**QUERY:**SELECT 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.**

**QUERY:**SELECT e1.student\_id, COUNT(e1.course\_id) AS C FROM Enrollments e1 JOIN Enrollments e2 ON e1.student\_id = e2.student\_id AND e1.course\_id <> e2.course\_id GROUP BY e1.student\_id HAVING C > 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.**

**QUERY:**SELECT t.teacher\_id, t.first\_name, t.last\_name FROM Teacher 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.**

**QUERY:**1 SELECT course\_id, AVG(num) FROM ( SELECT course\_id, COUNT(DISTINCT student\_id) as num FROM Enrollments GROUP BY course\_id) AS SQ 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.**

**QUERY:**SELECT s.student\_id, s.first\_name, s.last\_name, p.amount FROM Students s JOIN Payments p ON s.student\_id = p.student\_id WHERE p.amount = (SELECT MAX(amount) FROM Payments);

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

**QUERY:**SELECT c.course\_id, c.course\_name, COUNT(DISTINCT e.student\_id) AS C FROM Courses c JOIN Enrollments e ON c.course\_id = e.course\_id GROUP BY c.course\_id, c.course\_name ORDER BY C DESC LIMIT 1;

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

**QUERY:**SELECT CONCAT(t.first\_name," ",t.last\_name) AS teacher\_name, SUM(p.amount) AS total\_payments FROM teacher t LEFT JOIN Courses c ON t.teacher\_id = c.teacher\_id LEFT JOIN Enrollments e ON c.course\_id = e.course\_id LEFT JOIN Payments p ON e.student\_id = p.student\_id GROUP BY teacher\_name;

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

**QUERY:** SELECT student\_id,first\_name,last\_name FROM Students s WHERE (SELECT

COUNT(DISTINCT course\_id)FROM Courses) = (SELECT COUNT(DISTINCT course\_id)

FROM Enrollments e WHERE s.student\_id = e.student\_id);

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

**QUERY:** SELECT first\_name,last\_name FROM Teacher WHERE NOT EXISTS (SELECT 1 FROM Courses WHERE Courses.teacher\_id = Teacher.teacher\_id);

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

**QUERY:**SELECT AVG(age) FROM (SELECT TIMESTAMPDIFF(YEAR, date\_of\_birth, CURDATE()) AS age FROM Students) AS SQ;

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

**QUERY:**SELECT course\_id,course\_name FROM Courses WHERE NOT EXISTS (SELECT 1 FROM Enrollments WHERE Enrollments.course\_id = Courses.course\_id);

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

**QUERY:**SELECT s.first\_name,c.course\_name,SUM(p.amount) FROM Enrollments e JOIN Students s ON e.student\_id = s.student\_id JOIN Courses c ON e.course\_id = c.course\_id LEFT JOIN Payments p ON e.student\_id = p.student\_id GROUP BY s.first\_name, c.course\_name;

1. **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.**

**QUERY:**SELECT first\_name FROM students where student\_id in(SELECT student\_id FROM Payments GROUP BY student\_id HAVING COUNT(\*) > 1);

1. **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.**

**QUERY:**SELECT s.student\_id,s.first\_name,s.last\_name, SUM(p.amount) 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;

1. **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.**

**QUERY:**SELECT c.course\_name,COUNT(e.student\_id) FROM Courses c LEFT JOIN Enrollments e ON c.course\_id = e.course\_id GROUP BY c.course\_name;

1. **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.**

**QUERY:**SELECT s.first\_name,AVG(p.amount) FROM Students s LEFT JOIN Payments p ON s.student\_id = p.student\_id GROUP BY s.first\_name;