SQL ASSIGNMENT

STUDENT INFORMATION SYSTEM(SIS)

Task 1. Database Design:

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

**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.**

**a. Students**

**b. Courses**

**c. Enrollments**

**d. Teacher**

**e. Payments**

**CREATE TABLE Students (**

**student\_id INT PRIMARY KEY,**

**first\_name VARCHAR(50),**

**last\_name VARCHAR(50),**

**date\_of\_birth DATE,**

**email VARCHAR(100),**

**phone\_number VARCHAR(20)**

**);**

**CREATE TABLE Teacher (**

**teacher\_id INT PRIMARY KEY,**

**first\_name VARCHAR(50),**

**last\_name VARCHAR(50),**

**email VARCHAR(100)**

**);**

**CREATE TABLE Courses (**

**course\_id INT PRIMARY KEY,**

**course\_name VARCHAR(100),**

**credits INT,**

**teacher\_id INT,**

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

**);**

**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)**

**);**

**CREATE TABLE Payments (**

**payment\_id INT PRIMARY KEY,**

**student\_id INT,**

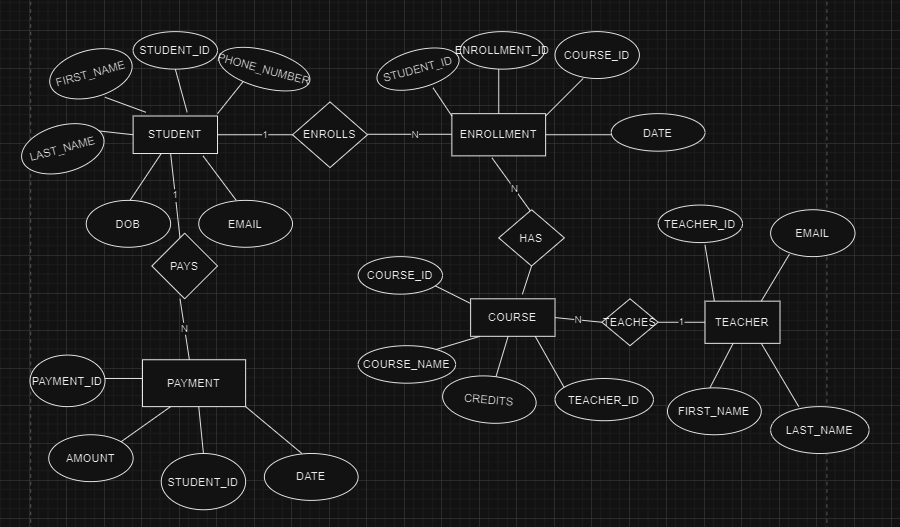
**amount DECIMAL(10, 2),**

**payment\_date DATE,**

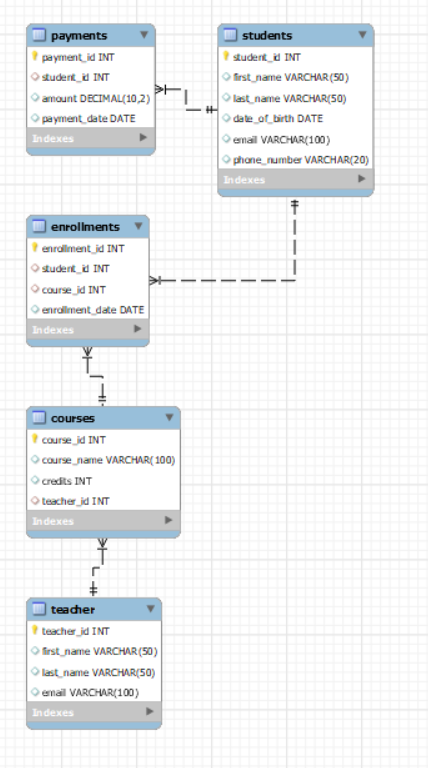
**FOREIGN KEY (student\_id) REFERENCES Students(student\_id)**

**);**

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

****

**EER DIAGRAM:-**

****

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

**CREATE TABLE Students (**

**student\_id INT PRIMARY KEY,**

**CREATE TABLE Teacher (**

**teacher\_id INT PRIMARY KEY,**

**CREATE TABLE Courses (**

**course\_id INT PRIMARY KEY,**

**course\_name VARCHAR(100),**

**credits INT,**

**teacher\_id INT,**

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

**);**

**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)**

**);**

**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)**

**);**

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

**i. Students**

**ii. Courses**

**iii. Enrollments**

**iv. Teacher**

**v. PaymentsINSERT INTO Students VALUES**

**(1, 'Varun', 'Reddy', '2002-04-17', 'varunreddy89782@gmail.com', '8978276693'),**

**(2, 'Jayanth', 'Yadav', '2002-06-27', 'jayanthyadav123@gmail.com', '951753159'),**

**(3, 'Pavan', 'Reddy', '1999-09-10', 'pavanreddy09@gmail.com', '7093456493'),**

**(4, 'Manish', 'Reddy', '2001-05-12', 'manishreddy17@gmail.com', '852741963'),**

**(5, 'Adarsh', 'Pakide', '2001-02-07', 'adarshpakide02@gmail.com', '741852963')**

**;**

**SELECT \* FROM Students;**

**INSERT INTO Teacher VALUES**

**(11, 'Prof', 'Sushma', 'sushma123@gmail.com'),**

**(12, 'Prof', 'VSVA Murthy', 'vsvsm03@gmail.com'),**

**(13, 'Prof', 'Neelima', 'nellima07@gmail.com'),**

**(14, 'Prof', 'Mahesh', 'uniquemahesh8@gmail.com'),**

**(15, 'Prof', 'Balchandra', 'balachandra52@gmail.com')**

**;**

**SELECT \* FROM Teacher;**

**INSERT INTO Courses VALUES**

**(101, 'Computer Science', 4, 11),**

**(102, 'Mathematics', 4, 12),**

**(103, 'English', 3, 13),**

**(104, 'Physics', 3, 14),**

**(105, 'chemistry', 3, 15)**

**;**

**SELECT \* FROM Courses;**

**INSERT INTO Enrollments VALUES**

**(21, 1, 101, '2023-01-15'),**

**(22, 2, 102, '2023-02-20'),**

**(23, 3, 103, '2023-03-05'),**

**(24, 4, 104, '2023-04-25'),**

**(25, 5, 105, '2023-05-30')**

**;**

**SELECT \* FROM Enrollments;**

**INSERT INTO Payments VALUES**

**(31, 1, 300.00, '2023-04-10'),**

**(32, 2, 200.50, '2023-05-15'),**

**(33, 3, 450.25, '2023-06-20'),**

**(34, 4, 400.20, '2023-07-25'),**

**(35, 5, 700.30, '2023-08-30')**

**;**

**SELECT \* FROM Payments;**

Task 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**

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

**VALUES (6, 'John', 'Doe', '1995-08-15', 'john.doe@example.com', '1234567890');**

**SELECT\*FROM Students;**

**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 Enrollments (enrollment\_id, student\_id, course\_id, enrollment\_date)**

**VALUES (26, 1, 102, '2023-09-15');**

**SELECT\*FROM Enrollments;**

**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 = 'rockstarbabu@gmail.com'**

**WHERE teacher\_id = 11;**

**SELECT\*FROM Teacher;**

**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 Enrollments**

**WHERE student\_id = 1 AND course\_id = 101;**

**SELECT\*FROM Enrollments;**

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

**UPDATE Courses**

**SET teacher\_id = 14**

**WHERE course\_id = 103;**

**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 Enrollments**

**WHERE student\_id = 1;**

**DELETE FROM Students**

**WHERE student\_id = 1;**

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

**UPDATE Payments**

**SET amount = 500.00**

**WHERE payment\_id = 31;**

**SELECT\*FROM Payments;**

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**

**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 = 1; -- Replace with the actual 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\_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;**

**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**

**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.**

**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.**

**SELECT**

**t.first\_name AS teacher\_first\_name,**

**t.last\_name AS teacher\_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.**

**SELECT**

**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\_id = 101; -- Replace with the actual 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**

**s.first\_name,**

**s.last\_name**

**FROM**

**Students s**

**LEFT JOIN**

**Payments 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**

**c.course\_id,**

**c.course\_name**

**FROM**

**Courses c**

**LEFT JOIN**

**Enrollments 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**

**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**

**GROUP BY**

**e1.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.**

**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.**

**SELECT**

**course\_id,**

**AVG(enrollment\_count) AS average\_students\_enrolled**

**FROM (**

**SELECT**

**course\_id,**

**COUNT(DISTINCT student\_id) AS enrollment\_count**

**FROM**

**Enrollments**

**GROUP BY**

**course\_id**

**) AS subquery**

**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**

**s.student\_id,**

**s.first\_name,**

**s.last\_name,**

**p.amount AS highest\_payment\_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.**

**SELECT**

**c.course\_id,**

**c.course\_name,**

**enrollment\_count**

**FROM**

**Courses c**

**JOIN (**

**SELECT**

**course\_id,**

**COUNT(student\_id) AS enrollment\_count**

**FROM**

**Enrollments**

**GROUP BY**

**course\_id**

**) AS subquery ON c.course\_id = subquery.course\_id**

**WHERE**

**enrollment\_count = (SELECT MAX(enrollment\_count) FROM (**

**SELECT**

**COUNT(student\_id) AS enrollment\_count**

**FROM**

**Enrollments**

**GROUP BY**

**course\_id**

**) AS max\_enrollments);**

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

**SELECT**

**t.teacher\_id,**

**t.first\_name,**

**t.last\_name,**

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

**t.teacher\_id, t.first\_name, t.last\_name;**

**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**

**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 c.course\_id)**

**FROM Courses c**

**);**

**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**

**Courses**

**);**

**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(student\_age) AS average\_age**

**FROM (**

**SELECT**

**student\_id,**

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

**FROM**

**Students**

**) AS subquery;**

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

**SELECT**

**course\_id,**

**course\_name**

**FROM**

**Courses**

**WHERE**

**course\_id NOT IN (**

**SELECT DISTINCT**

**course\_id**

**FROM**

**Enrollments**

**);**

**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**

**s.student\_id,**

**s.first\_name,**

**s.last\_name,**

**e.course\_id,**

**c.course\_name,**

**COALESCE(SUM(p.amount), 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**

**LEFT JOIN**

**Payments p ON e.student\_id = p.student\_id**

**AND e.course\_id = p.course\_id**

**GROUP BY**

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

**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**

**s.student\_id,**

**s.first\_name,**

**s.last\_name,**

**COUNT(p.payment\_id) AS payment\_count**

**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**

**HAVING**

**COUNT(p.payment\_id) > 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**

**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.**

**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.**

**SELECT**

**s.student\_id,**

**s.first\_name,**

**s.last\_name,**

**COALESCE(AVG(p.amount), 0) AS average\_payment\_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;**