<u>Part A</u>

Topic Explanation and E-R Diagram Design

Task:
Provide a brief explanation of the topic, outlining the scope and objectives of the database project.
Design an E-R (Entity-Relationship) diagram for the database, ensuring that there are at least five strong entities. This diagram should visually represent the entities, their attributes, and the relationships between them.

CREATE TABLE Statements

Task:
Write the necessary CREATE TABLE SQL statements for all the relations in the schema, based on the E-R diagram. These statements should include all constraint definitions such as primary keys, foreign keys, unique constraints, and not-null constraints.

INSERT INTO Statements

Task:
Write 5 different INSERT INTO statements for each table created by Member 2. This requires generating sample data that adheres to the constraints and relationships defined in the schema.

SQL Queries - Joins and Subqueries

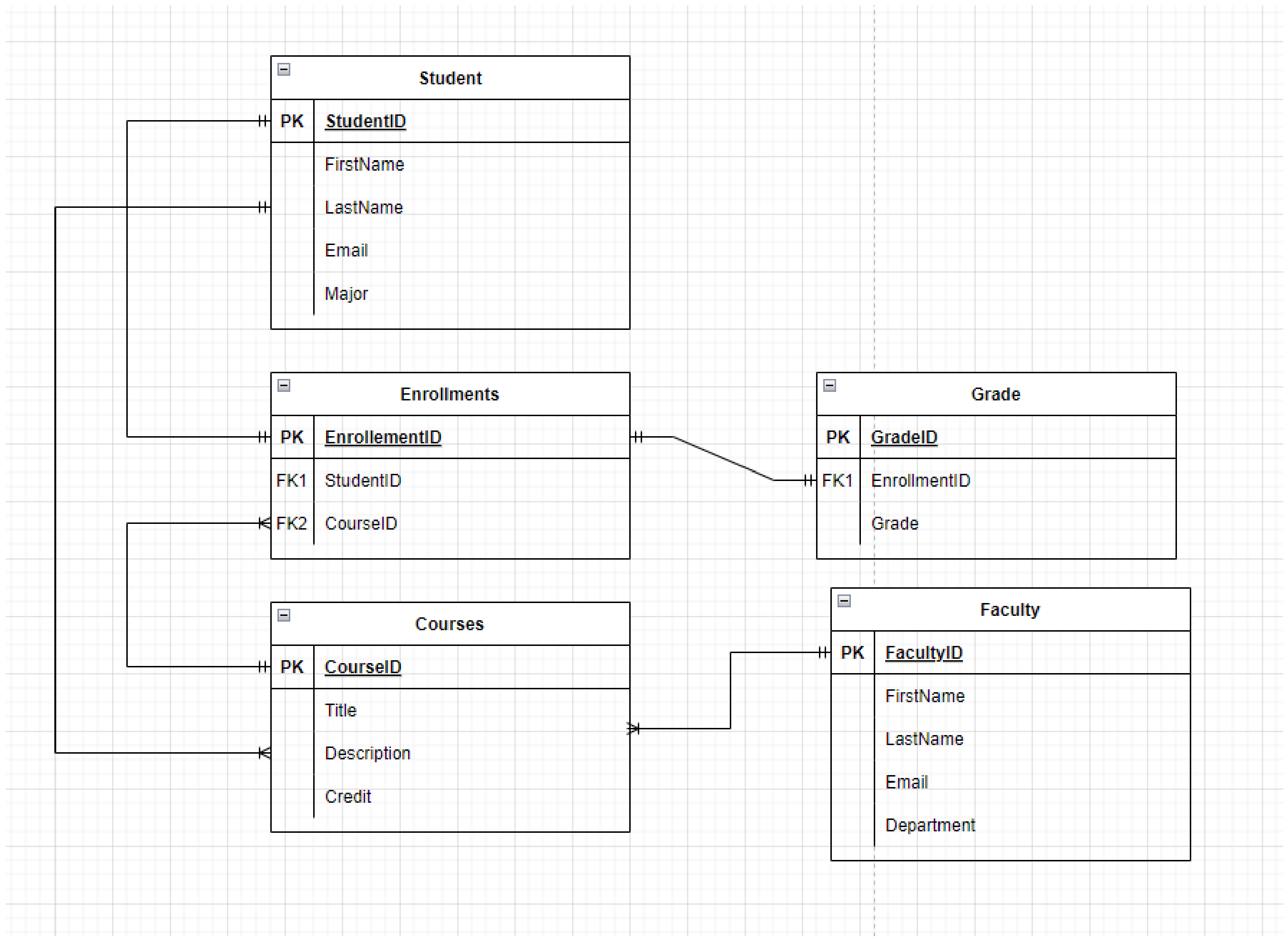
Task:
Write 3 SQL queries that use different types of joins (INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN) with conditions to retrieve related data from multiple tables.
Write 3 SQL queries that use nested or sub-queries to perform complex searches or calculations.

SQL Queries - Set Operations and Aggregates

Task:
Write 3 SQL queries that use set operations (UNION, INTERSECT, EXCEPT/MINUS) to combine results from different queries.

Write 3 SQL queries that perform aggregate operations (such as COUNT, SUM, AVG, MIN, MAX) and must include joins to aggregate data from multiple tables.

Part B



Part C

```
CREATE TABLE Students (
   StudentID NUMBER PRIMARY KEY,
   FirstName VARCHAR2(50),
   LastName VARCHAR2(50),
   Email VARCHAR2(100),
   Major VARCHAR2(50)
);

CREATE TABLE Courses (
   CourseID NUMBER PRIMARY KEY,
   Title VARCHAR2(100),
   Description CLOB,
   Credits NUMBER
);


CREATE TABLE Enrollments (
   EnrollmentID NUMBER PRIMARY KEY,
   StudentID NUMBER,
   CourseID NUMBER,
   CONSTRAINT fk_student FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
   CONSTRAINT fk_course FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);


CREATE TABLE Grades (
   GradeID NUMBER PRIMARY KEY,
   EnrollmentID NUMBER,
   Grade CHAR(2),
   CONSTRAINT fk_enrollment FOREIGN KEY (EnrollmentID) REFERENCES
Enrollments(EnrollmentID)
);


CREATE TABLE Faculty (
   FacultyID NUMBER PRIMARY KEY,
   FirstName VARCHAR2(50),
   LastName VARCHAR2(50),
   Email VARCHAR2(100),
   Department VARCHAR2(100)
);
```

Part D

```
INSERT INTO Students (StudentID, FirstName, LastName, Email, Major) VALUES (1, 'John',
'Doe', 'johndoe@email.com', 'Computer Science');
INSERT INTO Students (StudentID, FirstName, LastName, Email, Major) VALUES (2, 'Alice',
'Johnson', 'alicej@email.com', 'Mathematics');
INSERT INTO Students (StudentID, FirstName, LastName, Email, Major) VALUES (3, 'Bob',
'Smith', 'bobj@email.com', 'Physics');
INSERT INTO Students (StudentID, FirstName, LastName, Email, Major) VALUES (4, 'Carol',
'Lee', 'caroll@email.com', 'Chemistry');
INSERT INTO Students (StudentID, FirstName, LastName, Email, Major) VALUES (5, 'Dave',
'Brown', 'daveb@email.com', 'Biology');
```

```
INSERT INTO Courses (CourseID, Title, Description, Credits) VALUES (101, 'Introduction to
Computer Science', 'Basics of computing and programming', 3);
INSERT INTO Courses (CourseID, Title, Description, Credits) VALUES (102, 'Calculus I',
'Differential and Integral Calculus', 4);
INSERT INTO Courses (CourseID, Title, Description, Credits) VALUES (103, 'Physics 101',
'Fundamentals of Physics', 4);
INSERT INTO Courses (CourseID, Title, Description, Credits) VALUES (104, 'Organic
Chemistry', 'Principles of Organic Chemistry', 3);
INSERT INTO Courses (CourseID, Title, Description, Credits) VALUES (105, 'Cell Biology',
'Study of Cellular Functions', 3);

INSERT INTO Enrollments (EnrollmentID, StudentID, CourseID) VALUES (1, 1, 101);
INSERT INTO Enrollments (EnrollmentID, StudentID, CourseID) VALUES (2, 2, 102);
INSERT INTO Enrollments (EnrollmentID, StudentID, CourseID) VALUES (3, 3, 103);
INSERT INTO Enrollments (EnrollmentID, StudentID, CourseID) VALUES (4, 4, 104);
INSERT INTO Enrollments (EnrollmentID, StudentID, CourseID) VALUES (5, 5, 105);

INSERT INTO Grades (GradeID, EnrollmentID, Grade) VALUES (1, 1, 'A');
INSERT INTO Grades (GradeID, EnrollmentID, Grade) VALUES (2, 2, 'B');
INSERT INTO Grades (GradeID, EnrollmentID, Grade) VALUES (3, 3, 'C');
INSERT INTO Grades (GradeID, EnrollmentID, Grade) VALUES (4, 4, 'D');
INSERT INTO Grades (GradeID, EnrollmentID, Grade) VALUES (5, 5, 'F');



INSERT INTO Faculty (FacultyID, FirstName, LastName, Email, Department) VALUES (1,
'Jane', 'Smith', 'janesmith@email.com', 'Computer Science');
INSERT INTO Faculty (FacultyID, FirstName, LastName, Email, Department) VALUES (2,
'Emily', 'Davis', 'emilyd@email.com', 'Mathematics');
INSERT INTO Faculty (FacultyID, FirstName, LastName, Email, Department) VALUES (3,
'Nathan', 'Williams', 'nathanw@email.com', 'Physics');
INSERT INTO Faculty (FacultyID, FirstName, LastName, Email, Department) VALUES (4,
'Olivia', 'Brown', 'oliviab@email.com', 'Chemistry');
INSERT INTO Faculty (FacultyID, FirstName, LastName, Email, Department) VALUES (5,
'James', 'Miller', 'jamesm@email.com', 'Biology');
```

Part E

I) 3 joins (with conditions)

Query to get a list of students and the courses they're enrolled in.

```
SELECT Students.FirstName, Students.LastName, Courses.Title FROM Students
JOIN Enrollments ON Students.StudentID = Enrollments.StudentID
JOIN Courses ON Enrollments.CourseID = Courses.CourseID;
```

Query to find which faculty member is teaching a specific course.

```
-- Assuming a FacultyAssignments table exists linking faculty to courses
SELECT Faculty.FirstName, Faculty.LastName, Courses.Title FROM Faculty
JOIN FacultyAssignments ON Faculty.FacultyID = FacultyAssignments.FacultyID
JOIN Courses ON FacultyAssignments.CourseID = Courses.CourseID
WHERE Courses.Title = 'Introduction to Computer Science';
```

Query to find which faculty member is teaching a specific course.

```
-- Assuming a FacultyAssignments table exists linking faculty to courses
SELECT Faculty.FirstName, Faculty.LastName, Courses.Title FROM Faculty
JOIN FacultyAssignments ON Faculty.FacultyID = FacultyAssignments.FacultyID
JOIN Courses ON FacultyAssignments.CourseID = Courses.CourseID
WHERE Courses.Title = 'Introduction to Computer Science';
```

Query to list students and their grades for a specific course.

```
SELECT Students.FirstName, Students.LastName, Grades.Grade FROM Students
JOIN Enrollments ON Students.StudentID = Enrollments.StudentID
JOIN Grades ON Enrollments.EnrollmentID = Grades.EnrollmentID
JOIN Courses ON Enrollments.CourseID = Courses.CourseID
WHERE Courses.Title = 'Introduction to Computer Science';
```

II) Nested SubQueries

Query to find students who are majoring in the same field as 'John Doe'.

```
SELECT * FROM Students
WHERE Major = (SELECT Major FROM Students WHERE FirstName = 'John' AND LastName
= 'Doe');
```

Query to find courses that have no students enrolled.

```
SELECT * FROM Courses
WHERE CourseID NOT IN (SELECT CourseID FROM Enrollments);
```

Query to find the average grade for each course.

```
SELECT CourseID, (SELECT AVG(Grade) FROM Grades WHERE Grades.EnrollmentID =
Enrollments.EnrollmentID) AS AvgGrade
FROM Enrollments
GROUP BY CourseID;
```

III) set operations

Query to find all students who are either in 'Computer Science' or 'Mathematics' majors but not both.

```
(SELECT StudentID FROM Students WHERE Major = 'Computer Science')
UNION
(SELECT StudentID FROM Students WHERE Major = 'Mathematics');
```

Query to find students who are enrolled in both 'Computer Science' and 'Mathematics' courses.

```
(SELECT StudentID FROM Enrollments WHERE CourseID = (SELECT CourseID FROM
Courses WHERE Title = 'Introduction to Computer Science'))
INTERSECT
(SELECT StudentID FROM Enrollments WHERE CourseID = (SELECT CourseID FROM
Courses WHERE Title = 'Advanced Mathematics'));
```

Query to find all courses that are not offered this semester.

```
(SELECT CourseID FROM Courses)
EXCEPT
(SELECT CourseID FROM Enrollments);
```

IV) Aggregate operations

Query to count the number of students enrolled in each course.

```
SELECT Courses.Title, COUNT(Enrollments.StudentID) AS StudentCount FROM Courses
JOIN Enrollments ON Courses.CourseID = Enrollments.CourseID
GROUP BY Courses.Title;
```

Query to find the course with the highest average grade.

```
SELECT Courses.Title, AVG(Grades.Grade) AS AvgGrade FROM Courses
JOIN Enrollments ON Courses.CourseID = Enrollments.CourseID
JOIN Grades ON Enrollments.EnrollmentID = Grades.EnrollmentID
```

```
GROUP BY Courses.Title
ORDER BY AvgGrade DESC
LIMIT 1;
```

Query to calculate the average GPA of students in the 'Computer Science' department.

```
SELECT AVG(GPA) FROM (
   SELECT Students.StudentID, AVG(Grades.Grade) AS GPA FROM Students
   JOIN Enrollments ON Students.StudentID = Enrollments.StudentID
   JOIN Grades ON Enrollments.EnrollmentID = Grades.EnrollmentID
   WHERE Students.Major = 'Computer Science'
   GROUP BY Students.StudentID
) AS StudentGPAs;
```