**Mini Project: Student Management System**

Let's go through each SQL query from the provided solution and explain them in detail.

**Step 1: Database Setup and Sample Data**

This step involves creating the database and tables along with some sample data to work with.

Create Students table

CREATE TABLE Students ( student\_id INT PRIMARY KEY, student\_name VARCHAR(100), date\_of\_birth DATE,

email VARCHAR(100), major VARCHAR(50)

);

**Explanation:** This query creates a table named "Students" with columns for student\_id, student\_name, date\_of\_birth, email, and major. The student\_id is set as the primary key to ensure uniqueness.

Create Courses table CREATE TABLE Courses (

course\_id INT PRIMARY KEY, course\_name VARCHAR(100), credit\_hours INT,

instructor VARCHAR(100) );

**Explanation**: This query creates a table named "Courses" with columns for course\_id, course\_name, credit\_hours, and instructor. The course\_id is set as the primary key.

Insert sample data into Students table

INSERT INTO Students (student\_id, student\_name, date\_of\_birth, email, major) VALUES

(1, 'John Doe', '1998-07-15', 'john.doe@example.com', 'Computer Science'), (2, 'Jane Smith', '1999-03-22', 'jane.smith@example.com', 'Mathematics'),

(3, 'Michael Johnson', '2000-11-08', 'michael.johnson@example.com', 'Computer

Science'),

(4, 'Emily Williams', '2001-05-01', 'emily.williams@example.com', 'Physics');

**Explanation**: This query inserts sample data into the "Students" table. It adds four rows, each representing a student, with their respective student\_id, student\_name, date\_of\_birth, email, and major.

INSERT INTO Courses (course\_id, course\_name, credit\_hours, instructor) VALUES

(101, 'Database Management', 3, 'Prof. Brown'), (102, 'Programming Fundamentals', 4, 'Prof. White'), (103, 'Calculus I', 4, 'Prof. Smith'),

(104, 'Physics I', 4, 'Prof. Johnson');

**Explanation**: This query inserts sample data into the "Courses" table. It adds four rows, each representing a course, with their respective course\_id, course\_name, credit\_hours, and instructor.

**Step 2: SQL Queries**

Now, let's explain each SQL query based on the tasks mentioned in the project. Task 2: Retrieve Information

a. Retrieve the list of all students.

SELECT \* FROM Students;

**Explanation**: This query retrieves all rows and columns from the "Students" table, providing a list of all students in the database.

b. Retrieve the list of all courses.

SELECT \* FROM Courses;

**Explanation:** This query retrieves all rows and columns from the "Courses" table, providing a list of all courses in the database.

c. Retrieve the list of students majoring in a specific major (e.g., Computer Science).

SELECT \* FROM Students WHERE major = 'Computer Science';

**Explanation:** This query uses the WHERE clause to filter students based on their major. It retrieves all rows from the "Students" table where the "major" column is equal to 'Computer Science', providing a list of students majoring in Computer Science.

d. Retrieve the list of students who were born before a specific date.

SELECT \* FROM Students WHERE date\_of\_birth < '2000-01-01';

**Explanation:** This query uses the WHERE clause to filter students based on their date of birth. It retrieves all rows from the "Students" table where the "date\_of\_birth" column is earlier than '2000-01-01', providing a list of students born before this date.

e. Retrieve the list of courses taught by a specific instructor.

SELECT \* FROM Courses WHERE instructor = 'Prof. Smith';

**Explanation:** This query uses the WHERE clause to filter courses based on the instructor's name. It retrieves all rows from the "Courses" table where the "instructor" column is 'Prof. Smith', providing a list of courses taught by Professor Smith.

f. Retrieve the total number of students enrolled in each major.

SELECT major, COUNT(\*) AS total\_students FROM Students GROUP BY major;

**Explanation:** This query uses the GROUP BY clause to group students based on their major. It then uses the COUNT(\*) function to calculate the total number of students in each major. The result shows the major and the corresponding count of students enrolled in that major.

g. Retrieve the course with the highest number of credit hours.

SELECT \* FROM Courses ORDER BY credit\_hours DESC LIMIT 1;

**Explanation:** This query uses the ORDER BY clause to sort courses based on their credit\_hours in descending order. The LIMIT 1 ensures that only the top record (highest credit\_hours) is returned, providing the course with the highest number of credit hours.

h. Retrieve the oldest and youngest students in the database.

SELECT \* FROM Students ORDER BY date\_of\_birth ASC LIMIT 1; -- Oldest student

union

SELECT \* FROM Students ORDER BY date\_of\_birth DESC LIMIT 1; -- Youngest student

Explanation: These two queries retrieve the oldest and youngest students in the database separately. The first query uses ORDER BY date\_of\_birth ASC to sort students based on their date\_of\_birth in ascending order (oldest first), and LIMIT 1 ensures that only the top record is returned, providing the oldest student. The second query uses ORDER BY date\_of\_birth DESC to sort students based on their date\_of\_birth in descending order (youngest first), and LIMIT 1 ensures that only the top record is returned, providing the youngest student.

Task 3: Update Database

a. Add a new course to the Courses table.

INSERT INTO Courses (course\_id, course\_name, credit\_hours, instructor) VALUES (105, 'Algorithms', 3, 'Prof. Lee');

Explanation: This query uses the INSERT INTO statement to add a new course to the "Courses" table. It specifies the values for course\_id, course\_name, credit\_hours, and instructor for the new course.

b. Enroll a new student in the Students table.

INSERT INTO Students (student\_id, student\_name, date\_of\_birth, email, major)

VALUES (5, 'William Johnson', '2002-09-12', 'william.johnson@example.com', 'Computer Science');

**Explanation:** This query uses the INSERT INTO statement to enroll a new student in the "Students" table. It specifies the values for student\_id, student\_name, date\_of\_birth, email, and major for the new student.

c. Update the email address of a student based on their student\_id.

UPDATE Students SET email = 'Update.email@gmail.com' WHERE student\_id = 1;

**Explanation:** This query uses the UPDATE statement to modify the email address of a student in the "Students" table. It sets the email to 'jane.smith@gmail.com' for the student with student\_id = 2.

**Task 4: Nested Queries**

a. Retrieve the list of students who are enrolled in a specific course.

SELECT \* FROM Students WHERE student\_id IN (SELECT student\_id FROM Enrollments WHERE course\_id = 101);

**Explanation:** This query uses a nested query to retrieve the list of students who are enrolled in a specific course (course\_id = 101). The inner query (SELECT student\_id FROM Enrollments WHERE course\_id = 101) selects all student\_ids from a hypothetical "Enrollments" table where the course\_id matches 101. The outer query then uses the IN operator to retrieve all rows from the "Students" table where the student\_id is in the result of the inner query.

b. Retrieve the list of courses with enrollments greater than a certain number (e.g., 2).

SELECT \* FROM Courses WHERE course\_id IN (SELECT course\_id FROM Enrollments GROUP BY course\_id HAVING COUNT(\*) > 2);

**Explanation:** This query uses a nested query to retrieve the list of courses with enrollments greater than 2. The inner query (SELECT course\_id FROM Enrollments GROUP BY

course\_id HAVING COUNT(\*) > 2) first groups rows from the hypothetical "Enrollments" table by course\_id and then filters out groups with a count greater than 2. The outer query then uses the IN operator to retrieve all rows from the "Courses" table where the course\_id is in the result of the inner query.