SC1 1A) Imagine a College Management System which manages student information, classes, subjects, grades, attendance, and teacher assignments.
Students: Stores personal details of each student (student_ID, name, date_of_birth, gender, class_ID).
Classes: Lists the different classes in the school (class_ID, class_name, teacher_ID - optional, could be in a separate assignment table).
Subjects: Lists the subjects taught in the school (subject_ID, subject_name).(No. Of. Subjects= 4, Each carries 25 Marks).
Teachers: Stores personal details of teachers (teacher_ID, name, subject_ID - can teach multiple, so might need a linking table).
Marks: Records the marks obtained by students in different subjects for various exams (student_ID, class_ID,subject_ID, exam_name, marks).

```
CREATE TABLE Classes (
  class_ID VARCHAR(10) PRIMARY KEY,
  class_name VARCHAR(50) NOT NULL,
  teacher_ID VARCHAR(15) -- Optional, could be in a separate assignment table
  -- FOREIGN KEY (teacher_ID) REFERENCES Teachers(teacher_ID)
);

CREATE TABLE Students (
  student_ID VARCHAR(15) PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  date_of_birth DATE,
  gender VARCHAR(10),
  class_ID VARCHAR(10),
  FOREIGN KEY (class_ID) REFERENCES Classes(class_ID)
);
CREATE TABLE Subjects (
  subject_ID VARCHAR(10) PRIMARY KEY,
  subject_name VARCHAR(50) NOT NULL
);
CREATE TABLE Teachers (
  teacher_ID VARCHAR(15) PRIMARY KEY,
  name VARCHAR(100) NOT NULL
);

CREATE TABLE TeacherSubjects (
  teacher_ID VARCHAR(15),
  subject_ID VARCHAR(10),
  PRIMARY KEY (teacher_ID, subject_ID),
  FOREIGN KEY (teacher_ID) REFERENCES Teachers(teacher_ID),
  FOREIGN KEY (subject_ID) REFERENCES Subjects(subject_ID)
);


CREATE TABLE Marks (
  student_ID VARCHAR(15),
  class_ID VARCHAR(10),
    subject_ID VARCHAR(10),
    exam_name VARCHAR(50) NOT NULL,
    marks DECIMAL(5, 2),
    PRIMARY KEY (student_ID, class_ID, subject_ID, exam_name),
    FOREIGN KEY (student_ID) REFERENCES Students(student_ID),
    FOREIGN KEY (class_ID) REFERENCES Classes(class_ID),
    FOREIGN KEY (subject_ID) REFERENCES Subjects(subject_ID)
);

CREATE TABLE Attendance (
    student_ID VARCHAR(15),
    month VARCHAR(20) NOT NULL,
    status VARCHAR(10), -- e.g., 'Present', 'Absent'
    No_Of_Days_Present INT,
    PRIMARY KEY (student_ID, month),
    FOREIGN KEY (student_ID) REFERENCES Students(student_ID)
);

INSERT INTO Classes (class_ID, class_name) VALUES
('FY', 'First Year'),
('SY', 'Second Year'),
('TY', 'Third Year'),
('SE', 'Senior');

INSERT INTO Subjects (subject_ID, subject_name) VALUES
('SUB01', 'Mathematics'),
('SUB02', 'DBMS'),
('SUB03', 'CG'),
('SUB04', 'SE');

INSERT INTO Teachers (teacher_ID, name) VALUES
('T001', 'ABC'),
('T002', 'PQR'),
('T003', 'XYZ');

INSERT INTO TeacherSubjects (teacher_ID, subject_ID) VALUES
('T001', 'SUB01'),
('T001', 'SUB02'),
('T002', 'SUB03'),
('T003', 'SUB04');


INSERT INTO Students (student_ID, name, date_of_birth, gender, class_ID) VALUES
('S101', 'AAA', '2004-03-15', 'Female', 'SE'),
('S102', 'BBB', '2003-11-20', 'Male', 'TY'),
('S103', 'CCC', '2005-07-01', 'Male', 'FY'),
('S104', 'DDD', '2004-03-25', 'Female', 'SE'),
('S105', 'EEE', '2003-09-10', 'Female', 'SY');
```

```
INSERT INTO Marks (student_ID, class_ID, subject_ID, exam_name, marks) VALUES
('S101', 'SE', 'SUB01', 'Midterm', 23),
('S101', 'SE', 'SUB02', 'Midterm', 21),
('S101', 'SE', 'SUB03', 'Midterm', 20),
('S101', 'SE', 'SUB04', 'Midterm', 24),
('S102', 'TY', 'SUB01', 'Midterm', 18),
('S102', 'TY', 'SUB02', 'Midterm', 24),
('S102', 'TY', 'SUB03', 'Midterm', 22),
('S102', 'TY', 'SUB04', 'Midterm', 19),
('S103', 'FY', 'SUB01', 'Midterm', 16),
('S103', 'FY', 'SUB02', 'Midterm', 25),
('S103', 'FY', 'SUB03', 'Midterm', 18),
('S103', 'FY', 'SUB04', 'Midterm', 21),
('S104', 'SE', 'SUB01', 'Midterm', 15),
('S104', 'SE', 'SUB02', 'Midterm', 24),
('S104', 'SE', 'SUB03', 'Midterm', 10),
('S104', 'SE', 'SUB04', 'Midterm', 9),
('S105', 'SY', 'SUB01', 'Midterm', 6),
('S105', 'SY', 'SUB02', 'Midterm', 11),
('S105', 'SY', 'SUB03', 'Midterm', 12),
('S105', 'SY', 'SUB04', 'Midterm', 23);

INSERT INTO Attendance (student_ID, month, No_Of_Days_Present) VALUES
('S101', 'March',  20),
('S101', 'April',  22),
('S102', 'March', 18),
('S102', 'April',  15),
('S103', 'March',  21),
('S103', 'April',  19),
('S104', 'March', 22),
('S104', 'April', 21),
('S105', 'March', 16),
('S105', 'April', 23);
```

Part A:

1) Create a stored procedure to get the average marks of a specific class.

```
DELIMITER //
CREATE PROCEDURE GetAverageClassMarks(IN classId VARCHAR(10))
BEGIN
  SELECT   c.class_name,
    AVG(m.marks) AS average_marks
    FROM  Marks m
    JOIN
       Students s ON m.student_ID = s.student_ID
    JOIN
       Classes c ON s.class_ID = c.class_ID
    WHERE   s.class_ID = classId
    GROUP BY
       c.class_name; -- Add GROUP BY clause for the non-aggregated column
END //
DELIMITER ;
CALL GetAverageClassMarks('SE');
```

2) Create a View to Get Student marks for All Subjects.

```
CREATE VIEW StudentGrades AS
SELECT  s.student_ID,
   s.name AS student_name,
   c.class_name,   sub.subject_name,
   m.exam_name,    m.marks,
   CASE
      WHEN m.marks >= 90 THEN 'A'
      WHEN m.marks > 70 THEN 'B'
      WHEN m.marks >= 40 THEN 'C'
      ELSE 'D'
   END AS grade
FROM   Students s
JOIN
   Classes c ON s.class_ID = c.class_ID
JOIN
   Marks m ON s.student_ID = m.student_ID AND c.class_ID = m.class_ID
JOIN
   Subjects sub ON m.subject_ID = sub.subject_ID;
```

3)Display the student ID, Name, Class Name and Marks of a student having highest Marks in given subject.

```
SELECT    s.student_ID,
   s.name AS student_name,
   c.class_name,   m.marks
FROM  Students s
JOIN
   Classes c ON s.class_ID = c.class_ID
JOIN
   Marks m ON s.student_ID = m.student_ID
WHERE
   m.subject_ID = 'SUB01' -- Specify the subject ID here
```

```
  AND m.marks = (
     SELECT MAX(marks)
     FROM Marks
     WHERE subject_ID = 'SUB01' -- Ensure the subject ID matches
  );
```

4)List the names and dates of birth of all students born in the month of March.

```
SELECT name, date_of_birth
FROM Students
WHERE MONTH(date_of_birth) = 3;
```

5) Find the average marks for each subject across all students for a specific exam.

```
SELECT    sub.subject_name,
  AVG(m.marks) AS average_marks
FROM  Marks m
JOIN
  Subjects sub ON m.subject_ID = sub.subject_ID
WHERE
  m.exam_name = 'Midterm' -- Specify the exam name you are interested in
GROUP BY     sub.subject_name;
```

6)Write an SQL query to find the total number of students in each class.

```
SELECT     c.class_name,
  COUNT(s.student_ID) AS total_students
FROM     Classes c
LEFT JOIN
  Students s ON c.class_ID = s.class_ID
GROUP BY     c.class_name;
```

SC1 Part B

1) Define a function to calculate the number of absent days for a given student in a specific month.

```
DELIMITER //
CREATE FUNCTION CalculateAbsentDays (
  studentID VARCHAR(20),
  month1 varchar(20)    )
RETURNS INT
DETERMINISTIC
BEGIN
  DECLARE absent int;

SET absent= 30 -( SELECT No_Of_Days_Present
  FROM Attendance
```

```
   WHERE student_ID = studentID
     AND month = month1 );
      RETURN absent;
END//
DELIMITER ;

SELECT CalculateAbsentDays(101, 'April');
```

2) Combine a list of all students in class 'SE' who have scored more than 20 marks in any subject.

```
SELECT DISTINCT s.student_ID, s.name,m.class_ID, m.subject_ID, m.marks
FROM Students s
JOIN Marks m ON s.student_ID = m.student_ID
WHERE m.class_ID = 'SE'  AND m.marks > 20;
```

3) Identify students whose names have the second letter as 'a'.

```
SELECT student_ID, name
FROM Students
WHERE name LIKE '_a%';
```

4) Calculate the average attendance percentage for all students in class 'FY' for the month of March.

```
SELECT AVG(a.No_Of_Days_Present) AS average_attendance_percentage
FROM
Attendance a
JOIN  Students s  on s.student_ID=a.student_ID
WHERE
   s.class_ID = 'FY'
   AND a.month='April';
```

5) List the names of students who have scored below 10 marks in any subject.

```
SELECT DISTINCT s.name
FROM Students s
JOIN Marks m ON s.student_ID = m.student_ID
WHERE m.marks < 10;
```

6) List the names of all students in the 'SE' class and their corresponding total marks in the "Midterm" exam.

```
SELECT    s.name,
   SUM(m.marks) AS total_midterm_marks
FROM    Students s
JOIN
   Marks m ON s.student_ID = m.student_ID
WHERE   m.class_ID = 'SE'
   AND m.exam_name = 'Midterm'
GROUP BY
   s.student_ID, s.name
ORDER BY
   total_midterm_marks DESC;
```

SC1 Part C

1) Create a View to Get Students' Attendance Summary

```
CREATE VIEW StudentAttendanceSummary AS
SELECT
   s.student_ID,
   s.name AS student_name,
   c.class_ID, a.No_Of_Days_Present
    AS total_present_days, (30 - a.No_Of_Days_Present) AS total_absent_days , a.month
FROM
   Students s
JOIN
   Attendance a ON s.student_ID = a.student_ID
JOIN
   Classes c ON s.class_ID = c.class_ID
GROUP BY
   s.student_ID, s.name, c.class_name, a.No_Of_Days_Present, a.month;


select * from StudentAttendanceSummary;
```

2) Find the number of students in each class.

```
SELECT
   c.class_name,
   COUNT(s.student_ID) AS number_of_students
FROM
   Classes c
LEFT JOIN
   Students s ON c.class_ID = s.class_ID
GROUP BY
```

```
    c.class_name
ORDER BY
    c.class_name;
```

3) Retrieve the names and student IDs of all students whose names begin with the letter 'D'.

```
SELECT student_ID, name
FROM Students
WHERE name LIKE 'D%';
```

4) Write a query to find the total number of subjects offered in the school.

```
SELECT COUNT(*) AS total_subjects_offered
FROM Subjects;
```

5)Create a trigger to insert the student name , old class id , new class id and timestamp automatically in StudentClassHistory table when a student is moved to a different class.

```
DELIMITER //

CREATE TRIGGER AfterUpdate
AFTER UPDATE ON Students
FOR EACH ROW
BEGIN

    IF OLD.class_ID <> NEW.class_ID THEN

        INSERT INTO StudentClassHistory (student_ID, old_class_ID, new_class_ID, change_timestamp)
        VALUES (NEW.student_ID, OLD.class_ID, NEW.class_ID, NOW());

    END IF;
END//

DELIMITER ;

CREATE TABLE StudentClassHistory (
    student_ID VARCHAR(20) NOT NULL,
    old_class_ID VARCHAR(20),
    new_class_ID VARCHAR(20),
    change_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (student_ID) REFERENCES Students(student_ID),
    FOREIGN KEY (old_class_ID) REFERENCES Classes(class_ID),
    FOREIGN KEY (new_class_ID) REFERENCES Classes(class_ID)
);
```

6) Generate a combined list of the top 3 highest marks in the "Midterm" exam across all subjects

```
SELECT
    s.name AS student_name,
    m.subject_ID,
    sub.subject_name,
    m.marks
FROM    Marks m
JOIN
    Students s ON m.student_ID = s.student_ID
JOIN
    Subjects sub ON m.subject_ID = sub.subject_ID
WHERE
    m.exam_name = 'Midterm'
ORDER BY    m.marks DESC
LIMIT 3;
```

SC2) Imagine a popular mobile application called "Fitness Tracking" used by fitness enthusiasts in the city. This application allows users to log their workouts, track their steps, monitor their heart rate, and set fitness goals. The application uses a MySQL database to store user data. Entities are given below:

- **Users:** Stores user profiles (user_ID, name, email, registration_date, goal_ID).
- **Goals:** Stores info of fitness goals (goal_ID,goal_type)
- **Users_Goal**: (user_ID, goal_ID, goal_type)
- **Workouts:** Records details of each workout session (workout_ID, user_ID, workout_type, start_time, end_time, duration_in_minutes).
- **WorkoutDetails:** Stores specific details for each workout, such as exercises performed, sets, reps, and weight used (workout_ID, exercise_name, sets, repetitions, weight).
- **Steps:** Tracks daily step counts for each user (step_ID, user_ID, tracking_date, step_count).

```
CREATE TABLE Goals (
    goal_ID INT PRIMARY KEY AUTO_INCREMENT,
    goal_type VARCHAR(100) NOT NULL
  );
```

```
INSERT INTO Goals (goal_type) VALUES
('Lose Weight'),
('Strength Training'),
('Improve Endurance'),
('Increase Flexibility'),
('Maintain Current Fitness'),
('Gain Muscle');

CREATE TABLE Users (
   user_ID INT PRIMARY KEY AUTO_INCREMENT,
   name VARCHAR(255) NOT NULL,
   email VARCHAR(255) UNIQUE NOT NULL,
   registration_date DATE NOT NULL
  );

INSERT INTO Users (name, email, registration_date) VALUES
('PQR', 'PQR@gmail.com', '2024-11-20'),
('ABC', 'ABC@gmail.com', '2025-01-15'),
('XYZ', 'XYZ@gmail.com', '2025-02-03'),
('ABCD', 'ABCD@gmail.com','2025-03-01'),
('PQRS', 'PQRS@gmail.com','2025-04-05');


CREATE TABLE Users_Goal(user_ID INT ,
  goal_ID INT ,
  goal_type VARCHAR(200),
  FOREIGN KEY (user_ID) REFERENCES Users(user_ID),
  FOREIGN KEY (goal_ID) REFERENCES Goals(goal_ID)
);

INSERT INTO Users_Goal (user_ID, goal_ID, goal_type)
select 1, 3 , Goals.goal_type
FROM Goals
WHERE goal_ID=3;

INSERT INTO Users_Goal (user_ID, goal_ID, goal_type)
select 1, 4 , Goals.goal_type
FROM Goals
WHERE goal_ID=4;

INSERT INTO Users_Goal (user_ID, goal_ID, goal_type)
select 2, 2 , Goals.goal_type
FROM Goals
WHERE goal_ID=2;

INSERT INTO Users_Goal (user_ID, goal_ID, goal_type)
select 2, 6 , Goals.goal_type
```

```
FROM Goals
WHERE goal_ID=6;

INSERT INTO Users_Goal (user_ID, goal_ID, goal_type)
select 3, 1 , Goals.goal_type
FROM Goals
WHERE goal_ID=1;

CREATE TABLE Workouts (
    workout_ID INT PRIMARY KEY AUTO_INCREMENT,
    user_ID INT NOT NULL,
    workout_type VARCHAR(100) NOT NULL,
    start_time DATETIME NOT NULL,
    end_time DATETIME NOT NULL,
    duration_in_minutes INT NOT NULL,
    FOREIGN KEY (user_ID) REFERENCES Users(user_ID)
);

INSERT INTO Workouts (user_ID, workout_type, start_time, end_time, duration_in_minutes) VALUES
(1, 'Running', '2025-04-11 07:00:00', '2025-04-11 07:45:00', 45),
(2, 'Weightlifting', '2025-04-11 18:30:00', '2025-04-11 19:30:00', 60),
(1, 'Yoga', '2025-04-12 10:00:00', '2025-04-12 11:00:00', 60),
(3, 'Cycling', '2025-04-12 16:00:00', '2025-04-12 17:30:00', 90),
(2, 'Cardio', '2025-04-13 12:00:00', '2025-04-13 12:30:00', 30),
(4, 'Walking', '2025-04-13 08:00:00', '2025-04-13 08:45:00', 45),
(5, 'Weightlifting', '2025-04-14 19:00:00', '2025-04-14 20:15:00', 75),
(1, 'Swimming', '2025-04-14 09:00:00', '2025-04-14 09:45:00', 45),
(3, 'Running', '2025-04-15 17:00:00', '2025-04-15 17:35:00', 35),
(4, 'Hiking', '2025-04-15 10:00:00', '2025-04-15 11:30:00', 90);

CREATE TABLE Steps (
    Sr_No INT PRIMARY KEY AUTO_INCREMENT,
    user_ID INT NOT NULL,
    tracking_date DATE NOT NULL,
    step_count INT UNSIGNED NOT NULL,
    UNIQUE KEY (user_ID, tracking_date),
    FOREIGN KEY (user_ID) REFERENCES Users(user_ID)
);

INSERT INTO Steps (user_ID, tracking_date, step_count) VALUES
(1, '2025-01-11', 10250),
(2, '2025-01-11', 6780),
(3, '2025-01-11', 9125),
(4, '2025-01-11', 5300),
(5, '2025-01-11', 8500),
(1, '2025-01-12', 11500),
(2, '2025-01-12', 7200),
(3, '2025-01-12', 9800),
(4, '2025-01-12', 5800),
(5, '2025-02-12', 8900),
(1, '2025-02-13', 9800),
(2, '2025-02-13', 6500),
(3, '2025-02-13', 8950),
(4, '2025-02-13', 5100),
(5, '2025-03-13', 8200);
```

Part A

1) Find the total duration of workouts for each user.

```
SELECT
   u.name AS user_name,
   SUM(w.duration_in_minutes) AS total_workout_duration_minutes
FROM   Users u
JOIN
   Workouts w ON u.user_ID = w.user_ID
GROUP BY   u.user_ID, u.name;
```

2) Create a stored procedure to retrieve all workouts for a specific user within a given date range.

```
DELIMITER //

CREATE PROCEDURE GetUserWorkoutsByDateRange(
   IN p_user_id INT,
   IN p_start_date DATE,
   IN p_end_date DATE
)
BEGIN
   SELECT    workout_ID,
      workout_type,   start_time,
      end_time,   duration_in_minutes
   FROM    Workouts
   WHERE
      user_ID = p_user_id
      AND DATE(start_time) >= p_start_date
      AND DATE(end_time) <= p_end_date;
END //

DELIMITER ;


 CALL GetUserWorkoutsByDateRange(1, '2025-04-01', '2025-04-11');
```

3) Retrieve a list of users along with the total number of workouts they have logged.

```
SELECT
   u.name AS User_Name,
   COUNT(w.workout_ID) AS Total_Workouts
FROM   Users u
LEFT JOIN
   Workouts w ON u.user_ID = w.user_ID
GROUP BY  u.user_ID, u.name
ORDER BY  Total_Workouts DESC;
```

4) Create a view that shows a summary of each user's workouts. This should include the user's name, workout type and total duration of all workouts.

```
CREATE VIEW UserWorkoutSummary AS
SELECT   u.name AS User_Name,
   w.workout_type AS Workout_Type,
   SUM(w.duration_in_minutes) AS Total_Duration_Minutes
FROM  Users u
JOIN
   Workouts w ON u.user_ID = w.user_ID
GROUP BY
   u.user_ID, u.name, w.workout_type;

SELECT * FROM UserWorkoutSummary;
```

5) Retrieve all users who have set a goal to lose weight (use the goal_type field to identify weight loss goals).

```
SELECT  u.user_ID,
   u.name,  u.email,
   u.registration_date
FROM  Users u
JOIN
   Users_Goal ug ON u.user_ID = ug.user_ID
WHERE
   ug.goal_type = 'Lose Weight';
```

6) Retrieve the total number of workouts a user performed.

```
SELECT   u.name AS User_Name,
   COUNT(w.workout_ID) AS Total_Workouts
FROM  Users u
LEFT JOIN
   Workouts w ON u.user_ID = w.user_ID
GROUP BY  u.user_ID, u.name
```

```
ORDER BY   Total_Workouts DESC;
```

SC2 Part B

1) Retrieve users who have a 'Improve Endurance' goal and have logged 'Running' or 'Cycling' workouts.

```
SELECT DISTINCT
   u.user_ID,  u.name,  u.email
FROM  Users u
JOIN
   Users_Goal ug ON u.user_ID = ug.user_ID
JOIN
   Workouts w ON u.user_ID = w.user_ID
WHERE
   ug.goal_type = 'Improve Endurance'
   AND w.workout_type IN ('Running', 'Cycling');
```

2) Calculate the average daily step count for each user.

```
SELECT   u.user_ID,
   u.name AS User_Name,
   AVG(s.step_count) AS Average_Daily_Steps
FROM  Users u
LEFT JOIN
   Steps s ON u.user_ID = s.user_ID
GROUP BY
   u.user_ID, u.name
ORDER BY
   Average_Daily_Steps DESC;
```

3) List users who logged a workout on a day they also tracked more than 8,000 steps.

```
SELECT DISTINCT
   u.user_ID,   u.name,   u.email
FROM  Users u
JOIN
   Workouts w ON u.user_ID = w.user_ID
JOIN
   Steps s ON u.user_ID = s.user_ID
WHERE
   DATE(w.start_time) = s.tracking_date
   AND s.step_count > 5000;
```

4)Create a view showing each user's name and the date of their most recent workout.

```
CREATE VIEW UserLastWorkout AS
SELECT
   u.name AS User_Name,
   MAX(w.start_time) AS Last_Workout_Date
FROM   Users u
LEFT JOIN
   Workouts w ON u.user_ID = w.user_ID
GROUP BY   u.user_ID, u.name;
```

5) Create a Trigger to ensure that the end_time of a workout in the Workouts table is always after the start_time.

```
DELIMITER //

CREATE TRIGGER EnsureValidWorkoutTimes
BEFORE INSERT ON Workouts
FOR EACH ROW
BEGIN
   IF NEW.end_time <= NEW.start_time THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Workout end time must be after the start time.';
   END IF;
END //

DELIMITER ;

INSERT INTO Workouts (user_ID, workout_type, start_time, end_time, duration_in_minutes) VALUES
(3, 'Running', '2025-04-11 07:45:00', '2025-04-11 07:00:00', 45);
```

6) Identify users who have set a 'Lose Weight' goal and have logged more than 1 workouts.

```
SELECT   u.user_ID,   u.name,   u.email
FROM     Users u
JOIN
   Users_Goal ug ON u.user_ID = ug.user_ID
JOIN
   Workouts w ON u.user_ID = w.user_ID
WHERE   ug.goal_type = 'Lose Weight'
GROUP BY
   u.user_ID, u.name, u.email
HAVING   COUNT(w.workout_ID) > 1;
```

SC2 Part C

1) Find the user who has taken the most total steps

```
SELECT u.name AS user_name, SUM(s.step_count) AS total_steps
FROM Steps s
JOIN Users u ON s.user_ID = u.user_ID
GROUP BY u.name
ORDER BY total_steps DESC
LIMIT 1;
```

2) Find all users who have set a specific fitness goal type ('Increase Flexibility").

```
SELECT DISTINCT
   u.user_ID,    u.name,    u.email
FROM    Users u
JOIN
   Users_Goal ug ON u.user_ID = ug.user_ID
WHERE
   ug.goal_type = 'Increase Flexibility';
```

3) Find the average duration of each workout type

```
SELECT workout_type, AVG(duration_in_minutes) AS average_duration
FROM Workouts
GROUP BY workout_type;
```

4) Create a Function to calculate the total steps taken by a user within a specific date range.

```
DELIMITER //

CREATE FUNCTION GetTotalStepsInDateRange(userId INT, startDate DATE, endDate DATE)
RETURNS INT
DETERMINISTIC
BEGIN
   DECLARE totalSteps INT;

   SELECT SUM(step_count)
   INTO totalSteps
   FROM Steps
   WHERE user_ID = userId
    AND tracking_date >= startDate
    AND tracking_date <= endDate;

   IF totalSteps IS NULL THEN
      SET totalSteps = 0;
   END IF;
   RETURN totalSteps;
END //

DELIMITER ;
```

5) Retrieve all workouts where the duration was greater than 60 minutes.

```
SELECT     workout_ID,     user_ID,
   workout_type,     start_time,     end_time,
   duration_in_minutes
FROM     Workouts
WHERE     duration_in_minutes > 60;
```

6) Retrieve all workout sessions performed by a specific user on a given date, including the workout type, start time, and end time.

```
INSERT INTO Workouts (user_ID, workout_type, start_time, end_time, duration_in_minutes) VALUES
(1, 'Yoga', '2025-04-11 08:00:00', '2025-04-11 08:45:00', 45);

SELECT     w.workout_type,
   w.start_time,   w.end_time
FROM  Workouts w
WHERE
   w.user_ID = 1 AND DATE(w.start_time) = '2025-04-11';
```

SC 2 Part D

1) Identify Users Active on a Specific Date.

```
SELECT DISTINCT
   u.user_ID,    u.name
FROM    Users u
WHERE
   u.user_ID IN (SELECT user_ID FROM Workouts WHERE DATE(start_time) = '2025-04-11')
   OR u.user_ID IN (SELECT user_ID FROM Steps WHERE tracking_date = '2025-04-11');
```

2) Determine the most common goal_type

```
SELECT    g.goal_type,
   COUNT(ug.user_ID) AS user_count
FROM    Goals g
JOIN
   Users_Goal ug ON g.goal_ID = ug.goal_ID
GROUP BY    g.goal_type
```

```
ORDER BY    user_count DESC
LIMIT 1;
```

3)Find Users Who Registered in the Last Month

```
SELECT   user_ID,  name
FROM    Users
WHERE
   registration_date >= '2025-03-13' AND registration_date <= '2025-04-13';
```

4)Create a view to get a summary of each user's workouts, including their name, the workout type, and the duration.

```
CREATE VIEW UserWorkoutSummary AS
SELECT      u.user_ID,     u.name AS user_name,
   w.workout_ID,   w.workout_type,
   w.start_time,    w.end_time,
   w.duration_in_minutes
FROM   Users u
JOIN
   Workouts w ON u.user_ID = w.user_ID;
```

5) Create a cursor for the first three users listed in the Users table (based on their user_ID), display their user_ID and name

```
mysql> CREATE PROCEDURE ListFirstThreeUsers()
    -> BEGIN
    ->     -- Declare variables
    ->     DECLARE done INT DEFAULT FALSE;
    ->     DECLARE user_id_var INT;
    ->     DECLARE user_name_var VARCHAR(255);
    ->     DECLARE counter INT DEFAULT 0;
    ->     DECLARE limit_count INT DEFAULT 3;
    ->
    ->     -- Declare a cursor for the first three users
    ->     DECLARE user_cursor CURSOR FOR
    ->        SELECT user_ID, name
    ->        FROM Users
    ->        ORDER BY user_ID
    ->        LIMIT 3;
    ->
    ->     -- Declare handler for when the cursor is finished
    ->     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    ->
    ->     -- Open the cursor
    ->     OPEN user_cursor;
    ->
    ->     -- Loop through the first three users
    ->     user_loop: LOOP
    ->        FETCH user_cursor INTO user_id_var, user_name_var;
    ->
    ->        IF done THEN
    ->           LEAVE user_loop;
    ->        END IF;
    ->
    ->        -- Display the user ID and name
    ->        SELECT CONCAT('User ID: ', user_id_var, ', Name: ', user_name_var);
    ->
    ->     END LOOP user_loop;
    ->
    ->     -- Close the cursor
    ->     CLOSE user_cursor;
    ->
    -> END //
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
mysql> CALL ListFirstThreeUsers();
+------------------------------------------------------------+
| CONCAT('User ID: ', user_id_var, ', Name: ', user_name_var) |
+------------------------------------------------------------+
| User ID: 1, Name: PQR                                      |
+------------------------------------------------------------+
1 row in set (0.03 sec)

+------------------------------------------------------------+
| CONCAT('User ID: ', user_id_var, ', Name: ', user_name_var) |
+------------------------------------------------------------+
| User ID: 2, Name: ABC                                      |
+------------------------------------------------------------+
1 row in set (0.04 sec)

+------------------------------------------------------------+
| CONCAT('User ID: ', user_id_var, ', Name: ', user_name_var) |
+------------------------------------------------------------+
| User ID: 3, Name: XYZ                                      |
+------------------------------------------------------------+
1 row in set (0.04 sec)

Query OK, 0 rows affected (0.04 sec)
```

SC3 Imagine a online restaurant needs a system to manage the menu, track customer orders and generate sales reports. They use a MySQL database for this. Entities are as follows:

- **Menu:** Stores details of all the items offered by the restaurant (item_ID, item_name, category, price).
- **Customers:** Stores information about registered customers (customer_ID, name, contact_number).
- **Orders:** Records details of each order placed (order_ID, customer_ID, order_date_time).
- **OrderItems:** Lists the individual items included in each order (order_ID, item_ID, quantity, unit_price).

```
CREATE TABLE Menu (
    item_ID INT PRIMARY KEY AUTO_INCREMENT,
    item_name VARCHAR(255) NOT NULL,
    category VARCHAR(100) NOT NULL,
    price DECIMAL(10, 2) NOT NULL
);

INSERT INTO Menu (item_name, category, price) VALUES
('Margherita Pizza', 'Pizza', 12.99),
('Pepperoni Pizza', 'Pizza', 14.99),
('Veggie Burger', 'Burgers', 9.50),
('Chicken Burger', 'Burgers', 10.75),
('Caesar Salad', 'Salads', 8.25),
('Greek Salad', 'Salads', 9.00),
('Coke', 'Drinks', 2.00),
('Lemonade', 'Drinks', 2.50),
('Chocolate Cake', 'Desserts', 6.50),
('Ice Cream Sundae', 'Desserts', 7.00);

CREATE TABLE Customers (
    customer_ID INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    contact_number VARCHAR(20) UNIQUE
);

INSERT INTO Customers (name, contact_number) VALUES
('ABC', '9876543210'),
('PQR', '8765432109'),
('XYZ', '7654321098'),
('DEF', '6543210987');


CREATE TABLE Orders (
    order_ID INT PRIMARY KEY AUTO_INCREMENT,
    customer_ID INT,
    order_date_time DATETIME NOT NULL,
    FOREIGN KEY (customer_ID) REFERENCES Customers(customer_ID)
);

INSERT INTO Orders (customer_ID, order_date_time) VALUES
(1, '2025-04-11 19:30:00'),
(2, '2025-04-11 20:15:00'),
(1, '2025-04-10 12:00:00'),
(3, '2025-04-11 21:00:00'),
(NULL, '2025-04-11 18:45:00');

CREATE TABLE OrderItems (
    order_ID INT,
    item_ID INT,
    item_name VARCHAR(200),
    quantity INT NOT NULL,
    unit_price DECIMAL(10, 2) NOT NULL,
    PRIMARY KEY (order_ID, item_ID),
    FOREIGN KEY (order_ID) REFERENCES Orders(order_ID),
    FOREIGN KEY (item_ID) REFERENCES Menu(item_ID)
);

INSERT INTO OrderItems(order_ID, item_ID, item_name, quantity, unit_price)
select 1, 1, Menu.item_name, 2, 12.99
FROM Menu
WHERE item_ID=1;

INSERT INTO OrderItems(order_ID, item_ID, item_name, quantity, unit_price)
select 1, 2, Menu.item_name, 3, 15.00
FROM Menu
WHERE item_ID=2;

INSERT INTO OrderItems(order_ID, item_ID, item_name, quantity, unit_price)
select 1, 3, Menu.item_name, 1, 25.00
FROM Menu
WHERE item_ID=3;
```

Part A

1)Find the total sales for each menu category.

```
SELECT  m.category,
    SUM(oi.quantity * oi.unit_price) AS total_sales
FROM  Menu m
JOIN
    OrderItems oi ON m.item_ID = oi.item_ID
GROUP BY   m.category
ORDER BY  total_sales DESC;
```

2) Create a stored procedure to retrieve the details of a specific order given its order ID.

```
DELIMITER //
CREATE PROCEDURE GetOrderDetails(IN order_id_param INT)
BEGIN
    SELECT     o.order_ID,    c.name AS customer_name,
       c.contact_number AS customer_contact,
    o.order_date_time,  m.item_name,   oi.quantity,
    oi.unit_price,   (oi.quantity * oi.unit_price) AS item_total
  FROM   Orders o
  JOIN
    Customers c ON o.customer_ID = c.customer_ID
  JOIN
    OrderItems oi ON o.order_ID = oi.order_ID
  JOIN
    Menu m ON oi.item_ID = m.item_ID
  WHERE   o.order_ID = order_id_param;
END //
DELIMITER ;

 CALL GetOrderDetails(1);
```

3) Retrieve a list of all orders along with the customer's name.

```
SELECT   o.order_ID,   o.order_date_time,
 c.name AS customer_name
FROM    Orders o
LEFT JOIN
  Customers c ON o.customer_ID = c.customer_ID;
```

//Additional values inserted

```
INSERT INTO OrderItems(order_ID, item_ID, item_name, quantity, unit_price)
select 2, 8, Menu.item_name, 2, 50.00
FROM Menu  WHERE item_ID=8;

INSERT INTO OrderItems(order_ID, item_ID, item_name, quantity, unit_price)
select 3, 6, Menu.item_name, 4, 75.00
FROM Menu
WHERE item_ID=6;

INSERT INTO OrderItems(order_ID, item_ID, item_name, quantity, unit_price)
select 3, 7, Menu.item_name, 1, 35.00
FROM Menu
WHERE item_ID=7;

INSERT INTO OrderItems(order_ID, item_ID, item_name, quantity, unit_price)
select 4, 4, Menu.item_name, 4, 90.00
FROM Menu
WHERE item_ID=4;

INSERT INTO OrderItems(order_ID, item_ID, item_name, quantity, unit_price)
select 5, 5, Menu.item_name, 1, 45.00
FROM Menu
WHERE item_ID=5;
```

4) Identify menu items that have never been ordered.

```
SELECT m.item_name
FROM Menu m
LEFT JOIN OrderItems oi ON m.item_ID = oi.item_ID
WHERE oi.item_ID IS NULL;
```

5) Create a view to display top 3 selling items.

```
CREATE VIEW TopSellingItems AS
SELECT   m.item_name,
   SUM(oi.quantity) AS total_quantity_ordered
FROM   OrderItems oi
JOIN
   Menu m ON oi.item_ID = m.item_ID
GROUP BY   m.item_name
ORDER BY  total_quantity_ordered DESC
LIMIT 3;
```

6) Retrieve all orders placed on the current day.

//Adding additional record

```
INSERT INTO Orders (customer_ID, order_date_time) VALUES
(4, '2025-04-12 12:10:00'),
(2, '2025-04-12 12:05:00'),
(1, '2025-04-12 12:01:00'),
(3, '2025-04-12 12:01:00');

SELECT    o.order_ID,
   c.name AS customer_name,
   o.order_date_time
FROM   Orders o
LEFT JOIN
   Customers c ON o.customer_ID = c.customer_ID
WHERE
   DATE(o.order_date_time) = CURDATE();
```

SC3 Part B

1) Calculate the total price of each item in order_ID 1.

```
SELECT item_name, quantity, unit_price, quantity * unit_price AS total_price
```

```
FROM OrderItems WHERE order_ID = 1;
```

2)Find the names of all customers who have placed an order

```
SELECT DISTINCT c.name
FROM Customers c
JOIN Orders o ON c.customer_ID = o.customer_ID
WHERE o.customer_ID IS NOT NULL;
```

3)Create a view that shows the order ID, item name, quantity, and the total price for each item in an order.

```
CREATE VIEW DetailedOrderItems AS
SELECT  oi.order_ID,
   oi.item_name,   oi.quantity,
   oi.unit_price,  (oi.quantity * oi.unit_price) AS total_item_price
FROM   OrderItems oi;
```

4) Find the names of customers who have placed orders containing items from both the 'Pizza' and 'Burgers' categories.

```
SELECT DISTINCT c.customer_ID,c.name
FROM Customers c
WHERE c.customer_ID IN (
   SELECT o.customer_ID
   FROM Orders o
   JOIN OrderItems oi ON o.order_ID = oi.order_ID
   WHERE oi.item_name IN (SELECT item_name FROM Menu WHERE category = 'Pizza')
)
AND c.customer_ID IN (
   SELECT o.customer_ID
   FROM Orders o
   JOIN OrderItems oi ON o.order_ID = oi.order_ID
   WHERE oi.item_name IN (SELECT item_name FROM Menu WHERE category = 'Burgers')
);
```

5)Find the menu items that have never been included in any order.

```
SELECT item_name
FROM Menu
WHERE item_ID NOT IN (SELECT DISTINCT item_ID FROM OrderItems WHERE item_ID IS NOT
NULL);
```

6) Create a function to check if an item is in a specific category.

```
DELIMITER //
```

```
CREATE FUNCTION IsItemInCategory(item_name VARCHAR(255), category_name VARCHAR(100))
RETURNS VARCHAR(200)
DETERMINISTIC
BEGIN
  DECLARE item_count INT;

  -- Check if the item exists in the specified category
  SELECT COUNT(*) INTO item_count
  FROM Menu
  WHERE item_name = item_name AND category = category_name;

  -- Return TRUE if the item is found, FALSE otherwise
  IF item_count > 0 THEN
    RETURN CONCAT(item_name ,' is in ' ,category_name);
  ELSE
    RETURN CONCAT(item_name ,' is not in ' ,category_name);
  END IF;
END //
DELIMITER ;
```

SC3 Part C)

1) Create a stored procedure that takes an item name, category, and price as input and inserts a new record into the Menu table.

```
DELIMITER //
CREATE PROCEDURE AddNewMenuItem (
  IN item_name VARCHAR(255),
  IN category VARCHAR(100),
  IN price DECIMAL(10, 2)
)
BEGIN
  -- Insert the new menu item into the Menu table
  INSERT INTO Menu (item_name, category, price)
  VALUES (item_name, category, price);
END //
DELIMITER ;
```

2) Find the highest and lowest price from the Menu table, along with the names of the items.

```
SELECT
  MAX(price) AS highest_price,
  (SELECT item_name FROM Menu ORDER BY price DESC LIMIT 1) AS highest_price_item,
  MIN(price) AS lowest_price,
  (SELECT item_name FROM Menu ORDER BY price ASC LIMIT 1) AS lowest_price_item
```

```
FROM Menu;
```

3) Count the number of items in each category in the Menu table.

```
SELECT   category,
   COUNT(*) AS number_of_items
FROM   Menu
GROUP BY  category;
```

4)Calculate the average number of items per order.

```
SELECT
   AVG(item_count) AS average_items_per_order
FROM ( SELECT   order_ID,
      COUNT(*) AS item_count
   FROM   OrderItems
   GROUP BY   order_ID ) AS items_per_order;
```

5)Create a view that summarizes the sales of each menu item, including the item name, the total quantity sold, and the total revenue generated.

```
CREATE VIEW MenuItemSalesSummary AS
SELECT  m.item_name,
   SUM(oi.quantity) AS total_quantity_sold,
   SUM(oi.quantity * oi.unit_price) AS total_revenue
FROM   Menu m
JOIN
   OrderItems oi ON m.item_ID = oi.item_ID
GROUP BY  m.item_name;
```

6)Get the names of all items in a specific order, given the order ID.

```
SELECT   oi.item_name
FROM  OrderItems oi
WHERE   oi.order_ID = 1;
```

SC4 Imagine an online marketplace manages a vast inventory of Home appliances products, process customer orders, and track the fulfillment process. They use a MySQL database to manage their products, inventory, orders, and shipping information. Entities are listed below:

- **Products:** Stores detailed information about each product listed on the platform (product_ID, name, description, price, category_ID).
- **Categories:** Lists the different product categories (category_ID, category_name).

- **Inventory:** Tracks the stock levels for each product in their warehouse(s) (inventory_ID, product_ID, warehouse_ID, quantity_in_stock, last_stock_update).
- **Warehouses:** Lists the different warehouse locations (warehouse_ID, warehouse_name, address).
- **Orders:** Records customer orders (order_ID, customer_ID, order_date, shipping_address, order_status).
- **OrderItems:** Lists the individual products included in each order (order_item_ID, order_ID, product_ID, quantity_ordered, unit_price).
- **Shipments:** Tracks the shipment details for each order (shipment_ID, order_ID, shipping_carrier, tracking_number, shipment_date).

```
CREATE TABLE Categories (
  category_ID INT PRIMARY KEY,
  category_name VARCHAR(255) NOT NULL
);

INSERT INTO Categories (category_ID, category_name) VALUES
  (1, 'Refrigerators'),
  (2, 'Washing Machines'),
  (3, 'Dishwashers'),
  (4, 'Ovens'),
  (5, 'Cooktops');

CREATE TABLE Products (
  product_ID INT PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  description TEXT,
  price DECIMAL(10, 2) NOT NULL,
  category_ID INT,
  FOREIGN KEY (category_ID) REFERENCES Categories(category_ID)
);

INSERT INTO Products (product_ID, name, description, price, category_ID) VALUES
  (101, 'Refrigerator A1', 'Large capacity, stainless steel', 1200.00, 1),
  (102, 'Washing Machine B2', 'Front load, high efficiency', 800.00, 2),
  (103, 'Dishwasher C3', 'Built-in, quiet operation', 600.00, 3),
  (104, 'Oven D4', 'Convection, digital control', 900.00, 4),
  (105, 'Cooktop E5', 'Gas, 5 burners', 500.00, 5),
  (106, 'Refrigerator A2', 'Medium capacity, white', 950.00, 1),
  (107, 'Washing Machine B3', 'Top load, standard efficiency', 650.00, 2);

CREATE TABLE Warehouses (
  warehouse_ID INT PRIMARY KEY,
  warehouse_name VARCHAR(255) NOT NULL,
  address VARCHAR(255) NOT NULL
);
```

```
INSERT INTO Warehouses (warehouse_ID, warehouse_name, address) VALUES
   (1, 'Warehouse Alpha', '123 Main St, Anytown'),
   (2, 'Warehouse Beta', '456 Oak Ave, Somecity'),
   (3, 'Warehouse Gamma', '789 Pine Ln, Othertown');

CREATE TABLE Inventory (
   inventory_ID INT PRIMARY KEY,
   product_ID INT,
   warehouse_ID INT,
   quantity_in_stock INT NOT NULL,
   last_stock_update TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (product_ID) REFERENCES Products(product_ID),
   FOREIGN KEY (warehouse_ID) REFERENCES Warehouses(warehouse_ID)
);

INSERT INTO Inventory (inventory_ID, product_ID, warehouse_ID, quantity_in_stock) VALUES
   (1, 101, 1, 50),
   (2, 102, 1, 30),
   (3, 103, 2, 20),
   (4, 104, 2, 40),
   (5, 105, 3, 15),
   (6, 101, 2, 10),
   (7, 106, 1, 25),
   (8, 107, 3, 18);

CREATE TABLE Orders (
   order_ID INT PRIMARY KEY,
   customer_ID INT NOT NULL,
   order_date DATE NOT NULL,
   shipping_address VARCHAR(255) NOT NULL,
   order_status VARCHAR(50) NOT NULL
);

INSERT INTO Orders (order_ID, customer_ID, order_date, shipping_address, order_status) VALUES
   (1001, 201, '2023-01-15', '789 Pine Ln, Othertown', 'Shipped'),
   (1002, 202, '2023-02-20', '456 Oak Ave, Somecity', 'Delivered'),
   (1003, 201, '2023-03-10', '789 Pine Ln, Othertown', 'Processing'),
   (1004, 203, '2023-04-05', '321 Elm St, Anytown', 'Pending'),
   (1005, 202, '2023-05-12', '456 Oak Ave, Somecity', 'Shipped');

CREATE TABLE OrderItems (
   order_item_ID INT PRIMARY KEY,
   order_ID INT,
   product_ID INT,
   quantity_ordered INT NOT NULL,
   unit_price DECIMAL(10, 2) NOT NULL,
   FOREIGN KEY (order_ID) REFERENCES Orders(order_ID),
   FOREIGN KEY (product_ID) REFERENCES Products(product_ID)
);

INSERT INTO OrderItems (order_item_ID, order_ID, product_ID, quantity_ordered, unit_price) VALUES
   (1, 1001, 101, 2, 1200.00),
   (2, 1001, 103, 1, 600.00),
   (3, 1002, 102, 1, 800.00),
   (4, 1003, 104, 3, 900.00),
   (5, 1004, 105, 1, 500.00),
   (6, 1005, 102, 2, 800.00),
   (7, 1005, 107, 1, 650.00);

CREATE TABLE Shipments (
   shipment_ID INT PRIMARY KEY,
   order_ID INT,
   shipping_carrier VARCHAR(255),
   tracking_number VARCHAR(255),
   shipment_date DATE,
   FOREIGN KEY (order_ID) REFERENCES Orders(order_ID)
);

INSERT INTO Shipments (shipment_ID, order_ID, shipping_carrier, tracking_number, shipment_date)
VALUES
   (1, 1001, 'FedEx', '1234567890', '2023-01-18'),
   (2, 1002, 'UPS', '0987654321', '2023-02-22'),
   (3, 1005, 'DHL', '5678901234', '2023-05-14');
```

Part A

1) Find how many products are listed under each category.

```
SELECT  c.category_name,
   COUNT(p.product_ID) AS product_count
FROM  Categories c
JOIN
   Products p ON c.category_ID = p.category_ID
GROUP BY  c.category_name
ORDER BY   product_count DESC;
```

2) Create a stored procedure to retrieve the current stock level of a specific product in a given warehouse.

```
DELIMITER //
CREATE PROCEDURE GetProductStockLevel (
   IN p_id INT,  IN wr_id INT )
BEGIN
    SELECT quantity_in_stock
    FROM Inventory
   WHERE p_ID = product_id AND wr_ID = warehouse_id;
```

```
END //
DELIMITER ;

CALL GetProductStockLevel(101, 1);
```

3) Retrieve a list of all orders along with the names of the products ordered in each order.

```
SELECT   o.order_ID,
  o.order_date,  o.customer_ID,
  GROUP_CONCAT(p.name SEPARATOR ', ') AS products_ordered
FROM  Orders o
JOIN
  OrderItems oi ON o.order_ID = oi.order_ID
JOIN
  Products p ON oi.product_ID = p.product_ID
GROUP BY
  o.order_ID, o.order_date, o.customer_ID
ORDER BY   o.order_ID;
```

4) Create a view to show products with low stock (less than 10 units total)

```
CREATE VIEW LowStockProducts AS
SELECT   p.product_ID,
  p.name,  SUM(i.quantity_in_stock) AS total_stock
FROM Products p
JOIN
  Inventory i ON p.product_ID = i.product_ID
GROUP BY   p.product_ID, p.name
HAVING   total_stock < 20;
```

5)Count how many orders exist in each status (e.g., Pending, Shipped, Delivered).

```
SELECT   order_status,
  COUNT(*) AS order_count
FROM   Orders
GROUP BY   order_status
ORDER BY    order_count DESC;
```

6) Calculate the total number of products ordered by each customer.

```
SELECT  o.customer_ID,
  COUNT(oi.product_ID) AS total_products_ordered
FROM   Orders o
JOIN
  OrderItems oi ON o.order_ID = oi.order_ID
GROUP BY  o.customer_ID
ORDER BY   total_products_ordered DESC;
```

SC4 Part B

1) Find the total quantity in stock for each product across all warehouses.

```
SELECT    p.product_ID,
  p.name,   SUM(i.quantity_in_stock) AS total_quantity_in_stock
FROM   Products p
JOIN
  Inventory i ON p.product_ID = i.product_ID
GROUP BY    p.product_ID, p.name
ORDER BY   total_quantity_in_stock DESC;
```

2) Create a view to show the total price of each order

```
CREATE VIEW OrderTotalsView AS
SELECT    o.order_ID,
  o.order_date,    o.customer_ID,
  SUM(oi.quantity_ordered * oi.unit_price) AS total_price
FROM   Orders o
JOIN
  OrderItems oi ON o.order_ID = oi.order_ID
GROUP BY
  o.order_ID, o.order_date, o.customer_ID
ORDER BY   o.order_ID;
```

3) Combine a list of products in the 'Refrigerators' category and a list of products that have a price greater than 1000.

```
SELECT product_ID, name, price
FROM Products
WHERE category_ID = (SELECT category_ID FROM Categories WHERE category_name =
'Refrigerators')
UNION
SELECT product_ID, name, price
FROM Products
WHERE price > 700;
```

4) Create a cursor to loop through products and display their total stock.

```
DELIMITER //
CREATE PROCEDURE DisplayProductStock()
BEGIN
  DECLARE product_id INT;
  DECLARE product_name VARCHAR(255);
  DECLARE total_stock INT;
  DECLARE done INT DEFAULT FALSE;

   DECLARE product_cursor CURSOR FOR
    SELECT p.product_ID, p.name
    FROM Products p;
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

   OPEN product_cursor;

   read_loop: LOOP
      FETCH product_cursor INTO product_id, product_name;
      IF done THEN
         LEAVE read_loop;
      END IF;

      SELECT SUM(quantity_in_stock) INTO total_stock
      FROM Inventory
      WHERE product_ID = product_id;

      SELECT product_name, total_stock;
   END LOOP read_loop;

   CLOSE product_cursor;
END //
DELIMITER ;

CALL DisplayProductStock();
```

5) Show products with their stock, ordered by product name

```
SELECT p.product_id, p.name, SUM(i.quantity_in_stock) AS total_stock
FROM Products p
JOIN Inventory i ON p.product_id = i.product_id
GROUP BY p.product_id, p.name
ORDER BY p.name;
```

6) Calculate the total sales for each product based on the quantity ordered and unit price.

```
SELECT    p.product_ID,
   p.name AS product_name,
   SUM(oi.quantity_ordered * oi.unit_price) AS total_sales
FROM   Products p
JOIN
   OrderItems oi ON p.product_ID = oi.product_ID
GROUP BY   p.product_ID, p.name
ORDER BY    total_sales DESC;
```

SC 4 Part C)

// Additional enteries

```
INSERT INTO Shipments (shipment_ID, order_ID, shipping_carrier, tracking_number, shipment_date)
VALUES
```

```
  (4, 1004, 'FedEx', '1234567893', '2023-01-22');
```

1) Calculate how many shipments were handled by each carrier.

```
SELECT    shipping_carrier,
  COUNT(*) AS shipment_count
FROM  Shipments
GROUP BY  shipping_carrier
ORDER BY   shipment_count DESC;
```

2)List products, ordered by price in ascending order.

```
 SELECT *  FROM Products ORDER BY price ASC;
```

3) Create a trigger to prevent orders from being created if the quantity of products ordered exceeds the quantity in stock.

```
DELIMITER //
CREATE TRIGGER prevent_order_exceed_stock
BEFORE INSERT ON OrderItems
FOR EACH ROW
BEGIN
  DECLARE available_stock INT;
  SELECT quantity_in_stock INTO available_stock
  FROM Inventory
  WHERE product_ID = NEW.product_ID;
  IF NEW.quantity_ordered > available_stock THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Insufficient stock available for this product.';
  END IF;
END //
DELIMITER ;
```

4) Create a view to display the product name, current quantity in stock, and the name of the warehouse for all products.

```
CREATE VIEW ProductWarehouseStockView AS
SELECT   p.name AS product_name,
  i.quantity_in_stock,   w.warehouse_name
FROM   Products p
JOIN  Inventory i ON p.product_ID = i.product_ID
JOIN   Warehouses w ON i.warehouse_ID = w.warehouse_ID;
```

5) List all orders placed on a specific date.

```
SELECT * FROM Orders WHERE order_date = '2023-01-15';
```

6) Calculate the average quantity ordered for each product.

```
SELECT   product_ID,
   AVG(quantity_ordered) AS average_quantity
FROM   OrderItems
GROUP BY   product_ID;
```

SC4 Part D

1)Define a function to check if a product is currently out of stock (total quantity across all warehouses is zero).

```
DELIMITER //
CREATE FUNCTION IsProductOutOfStock(product_id INT) RETURNS VARCHAR(20)
DETERMINISTIC
BEGIN
   DECLARE total_stock INT;

   SELECT SUM(quantity_in_stock) INTO total_stock
   FROM Inventory
   WHERE product_ID = product_id;

   IF total_stock = 0 THEN
      RETURN 'Out of Stock';
   ELSE
      RETURN 'Available';
   END IF;
END //
DELIMITER ;

 SELECT IsProductOutOfStock(101);
```

2) Find the total sales for each category

```
SELECT    c.category_name,
   SUM(oi.quantity_ordered * oi.unit_price) AS total_sales
FROM    Categories c
JOIN
   Products p ON c.category_ID = p.category_ID
JOIN
   OrderItems oi ON p.product_ID = oi.product_ID
GROUP BY   c.category_name;
```

3) Implement a trigger to automatically decrease the quantity_in_stock in the Inventory table when a new order item is added

```
DELIMITER //
CREATE TRIGGER decrease_stock_on_order
AFTER INSERT ON OrderItems
FOR EACH ROW
BEGIN
```

```
   UPDATE Inventory
   SET quantity_in_stock = quantity_in_stock - NEW.quantity_ordered
   WHERE product_ID = NEW.product_ID;
END //
DELIMITER ;
```

4)Combine a list of products in the 'Washing Machines' category and a list of products that have a price greater than 900.

```
SELECT   product_ID, name,
   price, category_ID
FROM   Products
WHERE  category_ID = (SELECT category_ID
      FROM    Categories
      WHERE    category_name = 'Washing Machines' )
UNION
SELECT  product_ID, name, price, category_ID
FROM  Products
WHERE  price > 900;
```

5) Get all products in a specific category

```
SELECT * FROM Products WHERE category_ID = 1;
```

6) Find the total revenue generated by each product.

```
SELECT   p.name AS product_name,
   SUM(oi.quantity_ordered * oi.unit_price) AS total_revenue
FROM Products p
JOIN OrderItems oi ON p.product_ID = oi.product_ID
GROUP BY p.product_ID, p.name
ORDER BY total_revenue DESC;
```