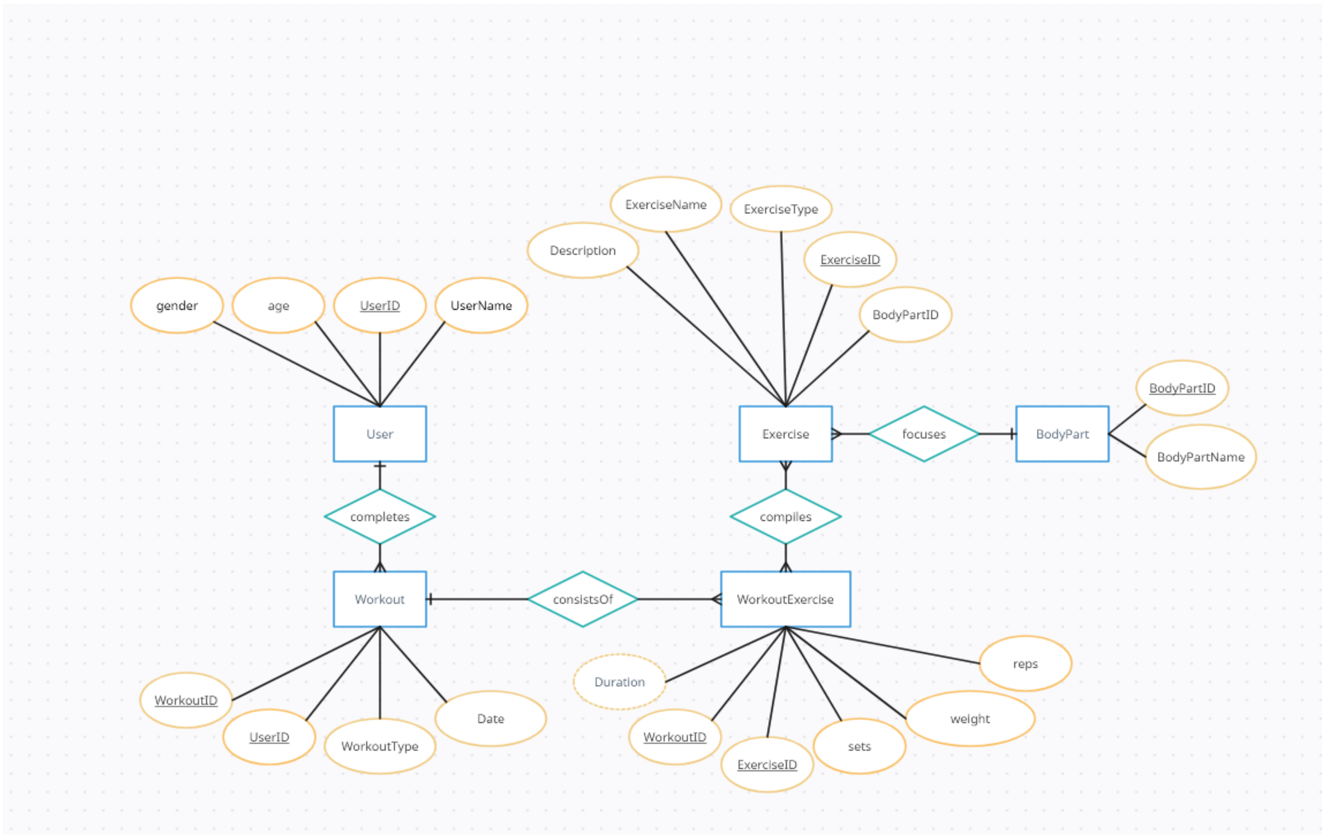
CS 2300 - Database Project
Daily Workout Tracker & Library Tool
By: Jack Jeep
10/10/2023

1. Problem Statement:

The designed database is a workout tracking system, allowing users to record and manage their workouts and the various exercises they are composed of. Though physical documentation has long been a standard within the industry, there are obvious downsides regarding sustainability, portability, and consistency. The designed database will allow users to disregard these former challengers and manage their workouts on a well-documented database.

2. Conceptual Database Design:

Below is the Entity-Relationship Diagram for the proposed schema:



Assumptions:

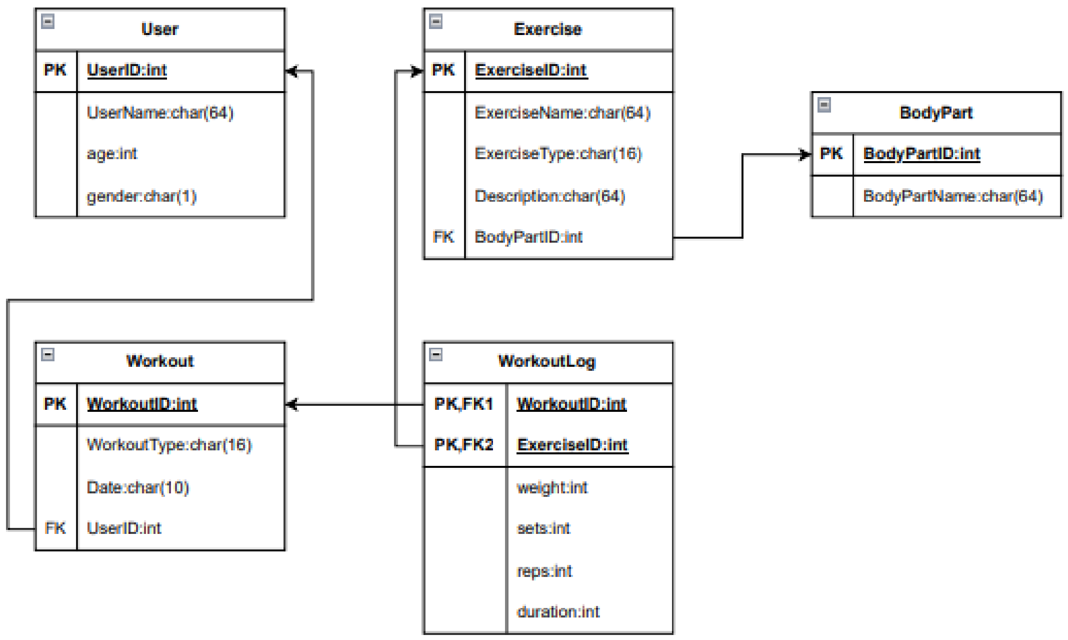
a. Users can add new exercises, and each exercise is associated with a specific body part.
b. Users can specify details like sets, reps, weight, and duration for each exercise within a workout.
c. The system assumes a user can have multiple workouts, and each workout can include multiple exercises.
d. The system assumes a body part can be associated with multiple exercises.

3. Functional Requirements:

The functional requirements are as follows:

1. User Management:
    a. Allows users to register and modify their accounts.
    b. Users can also update their workouts if changes are needed.
2. Workout Tracking:
    a. Users can create new workouts,
    b. Users can add new exercises completed to workouts.
3. Exercise Management:
    a. Add new exercises,
    b. View exercises.
    c. Modify existing exercises.
4. Data Recording:
    a. Record details of exercise during a given session.
5. Security:
    a. System is secure for users, ensuring that only those with access to a user's userName can make changes to workouts.
6. Analysis:
    a. Users should be able to view past workouts.
        i. Can be done by date, body part, and exercise.

4. Relational Database Design:



SUMMARY OF DATA TYPES:

Table	**Attribute**	**Type**	**Constraint**
User	UserID	int	PRIMARY KEY
User	UserName	VARCHAR(50)	UNIQUE
User	age	int	NOT NULL
User	gender	char(1)	NOT NULL
Workout	WorkoutID	int	PRIMARY KEY
Workout	WorkoutType	VARCHAR(50)	UNIQUE
Workout	Date	VARCHAR(50)	
Workout	UserID	int	FOREIGN KEY
WorkoutLog	WorkoutID	int	COMPOSITE PRIMARY KEY
WorkoutLog	ExerciseID	int	COMPOSITE PRIMARY KEY
WorkoutLog	set	int	NOT NULL
WorkoutLog	weight	int	NOT NULL
WorkoutLog	reps	int	NOT NULL
WorkoutLog	duration	int	
Exercise	ExerciseID	int	PRIMARY KEY
Exercise	ExerciseType	VARCHAR(50)	NOT NULL
Exercise	Description	VARCHAR(50)	
Exercise	BodyPartID	int	FOREIGN KEY
BodyPart	BodyPartID	int	PRIMARY KEY
BodyPart	BodyPartName	VARCHAR(50)	UNIQUE

APPLICATION PROGRAM DESIGN:

```
-- Create User table
CREATE TABLE IF NOT EXISTS User (
   UserID INTEGER AUTO_INCREMENT PRIMARY KEY,
   UserName VARCHAR(45) UNIQUE,
   Age INTEGER,
   Gender TEXT
);

-- Create Workout table
CREATE TABLE IF NOT EXISTS Workout (
   WorkoutID INTEGER AUTO_INCREMENT PRIMARY KEY,
   UserID INTEGER,
   WorkoutType VARCHAR(45),
   Date DATE,
   FOREIGN KEY (UserID) REFERENCES User(UserID)
);

-- Create BodyPart table
CREATE TABLE IF NOT EXISTS BodyPart (
   BodyPartID INTEGER AUTO_INCREMENT PRIMARY KEY,
   BodyPartName VARCHAR(45) UNIQUE
);

-- Create Exercise table
CREATE TABLE IF NOT EXISTS Exercise (
   ExerciseID INTEGER AUTO_INCREMENT PRIMARY KEY,
   ExerciseName VARCHAR(45),
   BodyPartID INTEGER,
   Description VARCHAR(45),
   FOREIGN KEY (BodyPartID) REFERENCES BodyPart(BodyPartID)
);

-- Create WorkoutExercise table
CREATE TABLE IF NOT EXISTS WorkoutExercise (
   WorkoutID INTEGER,
   ExerciseID INTEGER,
   Sets INTEGER,
   Reps INTEGER,
   Weight VARCHAR(50),
   Duration VARCHAR(50),
   PRIMARY KEY (WorkoutID, ExerciseID),
   FOREIGN KEY (WorkoutID) REFERENCES Workout(WorkoutID),
   FOREIGN KEY (ExerciseID) REFERENCES Exercise(ExerciseID)
);

-- INSERT A USER
DELIMITER //

CREATE PROCEDURE InsertUser(
   IN p_UserName TEXT,
   IN p_Age INT,
   IN p_Gender TEXT
)
```

```
BEGIN
    -- Insert a new user into the User table
    INSERT INTO User (UserName, Age, Gender)
    VALUES (p_UserName, p_Age, p_Gender);
END //

DELIMITER ;

-- INSERT A WORKOUT
DELIMITER //

CREATE PROCEDURE InsertWorkout(
    IN p_UserName VARCHAR(50),
    IN p_WorkoutType VARCHAR(20),
    IN p_Date DATE
)
BEGIN
    DECLARE v_UserID INT;

    -- Get the UserID for the specified user name
    SELECT UserID INTO v_UserID
    FROM User
    WHERE UserName = p_UserName
    LIMIT 1;

    -- If UserID is not found, raise an exception
    IF v_UserID IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'User not found';
    ELSE
        -- Insert workout directly using the resolved UserID
        INSERT INTO Workout (UserID, WorkoutType, Date)
        VALUES (v_UserID, p_WorkoutType, p_Date);
    END IF;
END //

DELIMITER ;


-- INSERT AN EXERCISE
DELIMITER //

CREATE PROCEDURE InsertExercise(
    IN p_ExerciseName VARCHAR(50),
    IN p_ExerciseType VARCHAR(50),
    IN p_BodyPartName VARCHAR(50)
)
BEGIN
    DECLARE v_BodyPartID INT;

    -- Get the BodyPartID for the specified body part name
    SELECT BodyPartID INTO v_BodyPartID
    FROM BodyPart
    WHERE BodyPartName = p_BodyPartName
    LIMIT 1;

    -- If BodyPartID is not found, raise an exception
    IF v_BodyPartID IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Body part not found';
    ELSE
        -- Insert exercise directly using the resolved BodyPartID
        INSERT INTO Exercise (ExerciseName, Description, BodyPartID)
        VALUES (p_ExerciseName, p_ExerciseType, v_BodyPartID);
    END IF;
    SELECT * FROM workoutdb.exercise;
END //

DELIMITER ;


-- INSERT AN EXERCISE FOR A WORKOUT
DELIMITER //

CREATE PROCEDURE InsertWorkoutExercise(
    IN p_UserName VARCHAR(50),
    IN p_Date DATE,
    IN p_ExerciseName VARCHAR(50),
    IN p_Sets INT,
    IN p_Reps INT,
    IN p_Weight VARCHAR(50),
    IN p_Duration VARCHAR(50)
)
BEGIN
    DECLARE v_UserID INT;
    DECLARE v_ExerciseID INT;
    DECLARE v_WorkoutID INT;

    -- Get the UserID for the specified user name
    SELECT UserID INTO v_UserID
    FROM User
    WHERE UserName = p_UserName
    LIMIT 1;

    -- If UserID is not found, raise an exception
    IF v_UserID IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'User not found';
    ELSE
        -- Get the ExerciseID for the specified exercise name
        SELECT ExerciseID INTO v_ExerciseID
        FROM Exercise
        WHERE ExerciseName = p_ExerciseName
        LIMIT 1;

        -- If ExerciseID is not found, raise an exception
        IF v_ExerciseID IS NULL THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Exercise not found';
        ELSE
            -- Get the WorkoutID for the specified user, date, and exercise
            SELECT WorkoutID INTO v_WorkoutID
            FROM Workout
            WHERE UserID = v_UserID AND Date = p_Date
            LIMIT 1;

            -- If WorkoutID is not found, raise an exception
            IF v_WorkoutID IS NULL THEN
                SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'Workout not found';
            ELSE
                -- Insert workout exercise directly using the resolved UserID, ExerciseID, and WorkoutID
                INSERT INTO WorkoutExercise (WorkoutID, ExerciseID, Sets, Reps, Weight, Duration)
                VALUES (v_WorkoutID, v_ExerciseID, p_Sets, p_Reps, p_Weight, p_Duration);
            END IF;
        END IF;
    END IF;
    CALL ReviewPastWorkoutsByDate(p_UserName, p_Date);
END //

DELIMITER ;




-- REVIEW PAST WORKOUTS BY DATE
DELIMITER //

CREATE PROCEDURE ReviewWorkoutsByDate(
    IN p_UserName VARCHAR(50),
    IN p_Date DATE
)
BEGIN
    -- Get the UserID for the specified user name
    DECLARE v_UserID INT;
    SELECT UserID INTO v_UserID FROM User WHERE UserName = p_UserName LIMIT 1;

    -- If UserID is not found, raise an exception
    IF v_UserID IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'User not found';
    ELSE
        -- Retrieve past workouts based on UserID and Date
        SELECT w.Date, w.WorkoutType, e.ExerciseName, we.Sets, we.Reps, we.Weight, we.Duration
        FROM Workout w
        JOIN WorkoutExercise we ON w.WorkoutID = we.WorkoutID
        JOIN Exercise e ON we.ExerciseID = e.ExerciseID
        WHERE w.UserID = v_UserID AND w.Date = p_Date;
    END IF;
END //
```

```
DELIMITER ;


-- REVIEW WORKOUT BY BODY PART NAME
DELIMITER //

CREATE PROCEDURE ReviewWorkoutsByBodyPartName(
  IN p_UserName VARCHAR(50),
  IN p_BodyPartName VARCHAR(50)
)
BEGIN
  -- Get the UserID for the specified user name
  DECLARE v_UserID INT;
  SELECT UserID INTO v_UserID FROM User WHERE UserName = p_UserName LIMIT 1;

  -- If UserID is not found, raise an exception
  IF v_UserID IS NULL THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'User not found';
  ELSE
    -- Retrieve past workouts based on UserID and BodyPartName
    SELECT w.Date, w.WorkoutType, e.ExerciseName, we.Sets, we.Reps, we.Weight, we.Duration
    FROM Workout w
    JOIN WorkoutExercise we ON w.WorkoutID = we.WorkoutID
    JOIN Exercise e ON we.ExerciseID = e.ExerciseID
    JOIN BodyPart bp ON e.BodyPartID = bp.BodyPartID
    WHERE w.UserID = v_UserID AND bp.BodyPartName = p_BodyPartName;
  END IF;
END //

-- REVIEW WORKOUT BY EXERCISE NAME
DELIMITER //

CREATE PROCEDURE ReviewWorkoutsByExerciseName(
  IN p_UserName VARCHAR(50),
  IN p_ExerciseName VARCHAR(50)
)
BEGIN
  -- Get the UserID for the specified user name
  DECLARE v_UserID INT;
  SELECT UserID INTO v_UserID FROM User WHERE UserName = p_UserName LIMIT 1;

  -- If UserID is not found, raise an exception
  IF v_UserID IS NULL THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'User not found';
  ELSE
    -- Retrieve past workouts based on UserID and ExerciseName
    SELECT w.Date, w.WorkoutType, e.ExerciseName, we.Sets, we.Reps, we.Weight, we.Duration
    FROM Workout w
    JOIN WorkoutExercise we ON w.WorkoutID = we.WorkoutID
    JOIN Exercise e ON we.ExerciseID = e.ExerciseID
    WHERE w.UserID = v_UserID AND e.ExerciseName = p_ExerciseName;
  END IF;
END //

DELIMITER ;

DELIMITER ;


-- ----------- DELETE USER
DELIMITER //

CREATE PROCEDURE DeleteUser(
  IN p_UserName VARCHAR(50)
)
BEGIN
  -- Get the UserID for the specified user name
  DECLARE v_UserID INT;
  SELECT UserID INTO v_UserID FROM User WHERE UserName = p_UserName LIMIT 1;

  -- If UserID is not found, raise an exception
  IF v_UserID IS NULL THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'User not found';
  ELSE
    -- Delete user and associated data using the WHERE clause with a key column
    DELETE FROM WorkoutExercise
    WHERE WorkoutID IN (SELECT WorkoutID FROM Workout WHERE UserID = v_UserID);

    DELETE FROM Workout WHERE UserID = v_UserID;

    DELETE FROM User WHERE UserID = v_UserID;
  END IF;
END //

DELIMITER ;




-- ---------- MODIFY USER
DELIMITER //

CREATE PROCEDURE ModifyUser(
  IN p_UserName VARCHAR(50),
  IN p_NewName VARCHAR(50),
  IN p_NewAge INT,
  IN p_NewGender VARCHAR(10)

)
BEGIN
  -- Modify user details using the WHERE clause with a key column
  UPDATE User
  SET
    UserName = p_NewName,
    Age = p_NewAge,
        Gender = p_NewGender
    WHERE UserName = p_UserName;

END //

DELIMITER ;

-- ------------ DELETE EXERCISE
DELIMITER //

CREATE PROCEDURE DeleteExercise(
    IN p_ExerciseName VARCHAR(50)
)
BEGIN
    -- Get the ExerciseID for the specified exercise name
    DECLARE v_ExerciseID INT;
    SELECT ExerciseID INTO v_ExerciseID FROM Exercise WHERE ExerciseName = p_ExerciseName LIMIT 1;

    -- If ExerciseID is not found, raise an exception
    IF v_ExerciseID IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Exercise not found';
    ELSE
        -- Delete exercise using the WHERE clause with a key column
        DELETE FROM Exercise WHERE ExerciseID = v_ExerciseID;
    END IF;
    SELECT * FROM workoutdb.exercise;
END //

DELIMITER ;

-- ------------ MODIFY EXERCISE
DELIMITER //

CREATE PROCEDURE ModifyExercise(
    IN p_ExerciseName VARCHAR(50),
    IN p_NewExerciseName VARCHAR(50),
    IN p_NewExerciseDescription VARCHAR(50),
    IN p_NewBodyPartName VARCHAR(50)
)
BEGIN
    -- Get the ExerciseID for the specified exercise name
    DECLARE v_ExerciseID INT;
    SELECT ExerciseID INTO v_ExerciseID FROM Exercise WHERE ExerciseName = p_ExerciseName LIMIT 1;

    -- If ExerciseID is not found, raise an exception
    IF v_ExerciseID IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Exercise not found';
    ELSE
        -- Modify exercise details using the WHERE clause with a key column
        UPDATE Exercise
        SET
            ExerciseName = p_NewExerciseName,
            Description = p_NewExerciseDescription,
            BodyPartID = (SELECT BodyPartID FROM BodyPart WHERE BodyPartName = p_NewBodyPartName
LIMIT 1)
        WHERE ExerciseID = v_ExerciseID;
    END IF;
    SELECT * FROM workoutdb.exercise;
END //

DELIMITER ;


-- -------------- DELETE WORKOUT
DELIMITER //

CREATE PROCEDURE DeleteWorkout(
    IN p_UserName VARCHAR(50),
    IN p_Date DATE
)
BEGIN
    -- Get the UserID for the specified user name
    DECLARE v_UserID INT;
    DECLARE v_WorkoutID INT;

    SELECT UserID INTO v_UserID FROM User WHERE UserName = p_UserName LIMIT 1;

    IF v_UserID IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'User not found';
    ELSE
        SELECT WorkoutID INTO v_WorkoutID FROM Workout WHERE UserID = v_UserID AND Date = p_Date
LIMIT 1;

        IF v_WorkoutID IS NULL THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Workout not found';
        ELSE
            -- Delete workout using the WHERE clause with key columns
            DELETE FROM Workout WHERE WorkoutID = v_WorkoutID;
        END IF;
    END IF;
END //

DELIMITER ;




-- ----------- MODIFY WORKOUT
DELIMITER //

CREATE PROCEDURE ModifyWorkout(
    IN p_UserName VARCHAR(50),
    IN p_Date DATE,
    IN p_NewWorkoutType VARCHAR(20)
)
BEGIN
    -- Get the UserID for the specified user name
    DECLARE v_UserID INT;
    DECLARE v_WorkoutID INT;

    SELECT UserID INTO v_UserID FROM User WHERE UserName = p_UserName LIMIT 1;

    IF v_UserID IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'User not found';
    ELSE
        SELECT WorkoutID INTO v_WorkoutID FROM Workout WHERE UserID = v_UserID AND Date = p_Date
LIMIT 1;

        IF v_WorkoutID IS NULL THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Workout not found';
        ELSE
            -- Modify workout details using the WHERE clause with key columns
            UPDATE Workout
            SET WorkoutType = p_NewWorkoutType
            WHERE WorkoutID = v_WorkoutID;
        END IF;
    END IF;
    CALL ReviewPastWorkoutsByDate(p_UserName, p_Date);
END //

DELIMITER ;





-- ------------ DELETE WORKOUTEXERCISE
DELIMITER //

CREATE PROCEDURE DeleteWorkoutExercise(
    IN p_UserName VARCHAR(50),
    IN p_Date DATE,
    IN p_ExerciseName VARCHAR(50)
)
BEGIN
    -- Get the UserID, ExerciseID, and WorkoutID for the specified user, date, and exercise
    DECLARE v_UserID INT;
    DECLARE v_ExerciseID INT;
    DECLARE v_WorkoutID INT;

    SELECT UserID INTO v_UserID FROM User WHERE UserName = p_UserName LIMIT 1;
    SELECT ExerciseID INTO v_ExerciseID FROM Exercise WHERE ExerciseName = p_ExerciseName LIMIT 1;

    IF v_UserID IS NULL OR v_ExerciseID IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'User or Exercise not found';
    ELSE
        SELECT WorkoutID INTO v_WorkoutID FROM Workout WHERE UserID = v_UserID AND Date = p_Date
LIMIT 1;

        IF v_WorkoutID IS NULL THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Workout not found';
        ELSE
            -- Delete workout exercise using the WHERE clause with key columns
            DELETE FROM WorkoutExercise
            WHERE WorkoutID = v_WorkoutID AND ExerciseID = v_ExerciseID;
        END IF;
    END IF;
END //

DELIMITER ;

-- ----------- MODIFY WORKOUTEXERCISE
DELIMITER //

CREATE PROCEDURE ModifyWorkoutExercise(
    IN p_UserName VARCHAR(50),
    IN p_Date DATE,
    IN p_ExerciseName VARCHAR(50),
    IN p_NewSets INT,
    IN p_NewReps INT,
    IN p_NewWeight VARCHAR(50),
    IN p_NewDuration VARCHAR(50)
)
BEGIN
    -- Get the UserID, ExerciseID, and WorkoutID for the specified user, date, and exercise
    DECLARE v_UserID INT;
    DECLARE v_ExerciseID INT;
    DECLARE v_WorkoutID INT;

    SELECT UserID INTO v_UserID FROM User WHERE UserName = p_UserName LIMIT 1;
    SELECT ExerciseID INTO v_ExerciseID FROM Exercise WHERE ExerciseName = p_ExerciseName LIMIT 1;

    IF v_UserID IS NULL OR v_ExerciseID IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'User or Exercise not found';
    ELSE
        SELECT WorkoutID INTO v_WorkoutID FROM Workout WHERE UserID = v_UserID AND Date = p_Date
LIMIT 1;

        IF v_WorkoutID IS NULL THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Workout not found';
        ELSE
            -- Modify workout exercise details using the WHERE clause with key columns
            UPDATE WorkoutExercise
            SET
                Sets = p_NewSets,
            Reps = p_NewReps,
            Weight = p_NewWeight,
            Duration = p_NewDuration
        WHERE WorkoutID = v_WorkoutID AND ExerciseID = v_ExerciseID;
      END IF;
   END IF;
   CALL ReviewPastWorkoutsByDate(p_UserName, p_Date);
END //

DELIMITER ;
```

INSTALLATION INSTRUCTIONS:

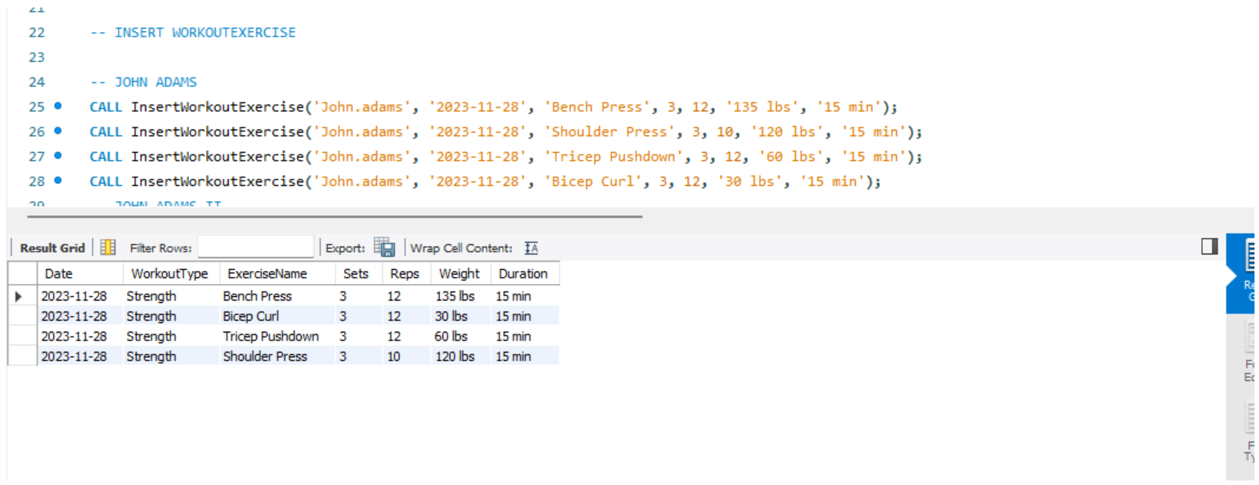
The database presented was intended for use on the Window Operating System. The system provided was built on MySQL 8.0 and is intended for the local instance framework provided by MySQL. Users can start by first downloading the MySQL platform on their windows device. They can then establish a local instance using their machine, and running the provided SQL script (WorkoutDBSQL.sql, also shown above) and the database and its relevant procedures will be established.

USER MANUAL:

While I would've liked to have established a full UI for this application, it simply wasn't achievable given various challenges faced. As such, I have made it fairly easy for users to query all the information necessary to use this database to its full potential.

```
-- ---------------------------------- ADD NEW USER & WORKOUT INFO ----------------------------------------------------------------------
-- INSERT USER
-- CALL InsertUser('john_doe', 33, 'Male');

-- INSERT WORKOUT
-- CALL InsertWorkout('john_doe','Strength', '2023-12-5');

-- INSERT EXERCISE
-- CALL InsertExercise('Shoudler Press', 'Upward pressing exercise', 'Shoulders');
-- OR VIEW CURRENT EXERCIES
-- SELECT * FROM workoutdb.exercise;

-- INSERT WORKOUTEXERCISE
-- CALL InsertWorkoutExercise('Alex Jones Sr.', '2023-11-23', 'Shoulder Press', 3, 12, '120 lbs', '30 min');

-- ------------------------------------- REVIEW WORKOUTS ----------------------------------------------------
-- REVIEW BY DATE AND ID
-- CALL ReviewPastWorkoutsByDate('Alex Jones Sr.', '2023-11-23');

-- REVIEW BY BODY PART NAME AND USERNAME
-- ReviewPastWorkoutsByBodyPartName('john_doe', 'Legs');

-- REVIEW BY EXERCISE NAME AND ID
-- Review past workouts by exercise name
-- CALL ReviewPastWorkoutsByExerciseName('john_doe', 'Split Squat');

-- ------------------------------------ DELETE QUERIES -------------------------------------------------------
-- Delete user with UserName = 'john_doe'
-- CALL DeleteUser('john_doe');

-- Delete exercise with ExerciseName = 'Bench Press'
-- CALL DeleteExercise('Deadlift');

-- Delete workouts for user with UserName = 'john_doe' on date '2023-11-22'
-- CALL DeleteWorkout('Alex Jones', '2023-11-22');
-- Delete workout exercises for user with UserName = 'john_doe', on date '2023-11-22', and for exercise 'Bench Press'
-- CALL DeleteWorkoutExercise('Alex Jones Sr.', '2023-11-23', 'Bicep Curl');


-- ------------------------------- MODIFY QUERIES ---------------------------------------------------------------------
-- Modify user with UserName = 'john_doe'
-- CALL ModifyUser('Alex Jones', 'Alex Jones Sr.', 55, 'Male');


-- Modify exercise with ExerciseName = 'Bench Press'
-- CALL ModifyExercise('Deadlift', 'Deadlift', 'Hip hinging exercise', 'Legs');


-- Modify workouts for user with UserName = 'john_doe' on date '2023-11-22'
-- CALL ModifyWorkout('Alex Jones Sr.', '2023-11-23', 'Mobility');


-- Modify workout exercises for user with UserName = 'john_doe', on date '2023-11-22', and for exercise 'Bench Press'
-- CALL ModifyWorkoutExercise('Alex Jones Sr.', '2023-11-23', 'Preacher Curl', 3, 125, '30 lbs', '20 min');
```

The above provides all the relevant information needed to access, add, modify, and remove data.

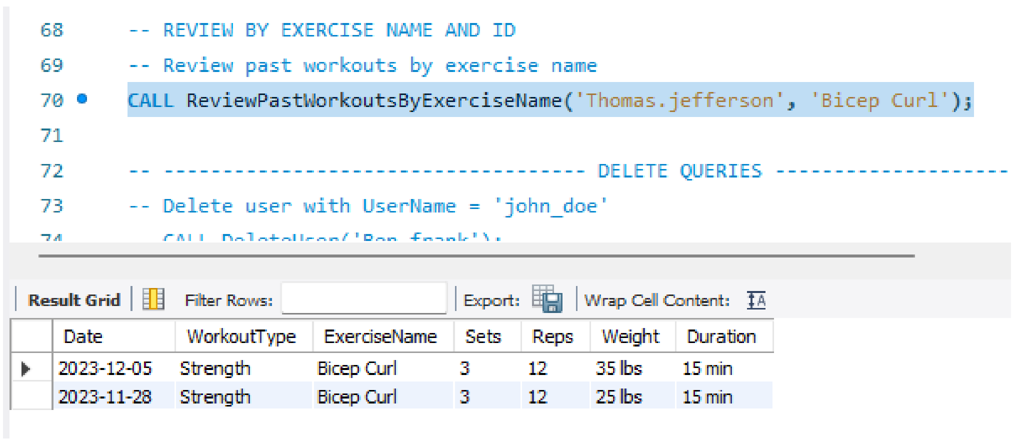


```
-- INSERT WORKOUTEXERCISE

-- JOHN ADAMS
CALL InsertWorkoutExercise('John.adams', '2023-11-28', 'Bench Press', 3, 12, '135 lbs', '15 min');
CALL InsertWorkoutExercise('John.adams', '2023-11-28', 'Shoulder Press', 3, 10, '120 lbs', '15 min');
CALL InsertWorkoutExercise('John.adams', '2023-11-28', 'Tricep Pushdown', 3, 12, '60 lbs', '15 min');
CALL InsertWorkoutExercise('John.adams', '2023-11-28', 'Bicep Curl', 3, 12, '30 lbs', '15 min');
```

Date	WorkoutType	ExerciseName	Sets	Reps	Weight	Duration
2023-11-28	Strength	Bench Press	3	12	135 lbs	15 min
2023-11-28	Strength	Bicep Curl	3	12	30 lbs	15 min
2023-11-28	Strength	Tricep Pushdown	3	12	60 lbs	15 min
2023-11-28	Strength	Shoulder Press	3	10	120 lbs	15 min

Inserting, deleting, or modifying an exercise for a given workout will provide the user with a summary of the workout on that particular date. This will also occur when the user modifies a workout for a particular date. The user can also just CALL a review of a workout as per the REVIEWPASTWORKOUTSBYDATE procedure.

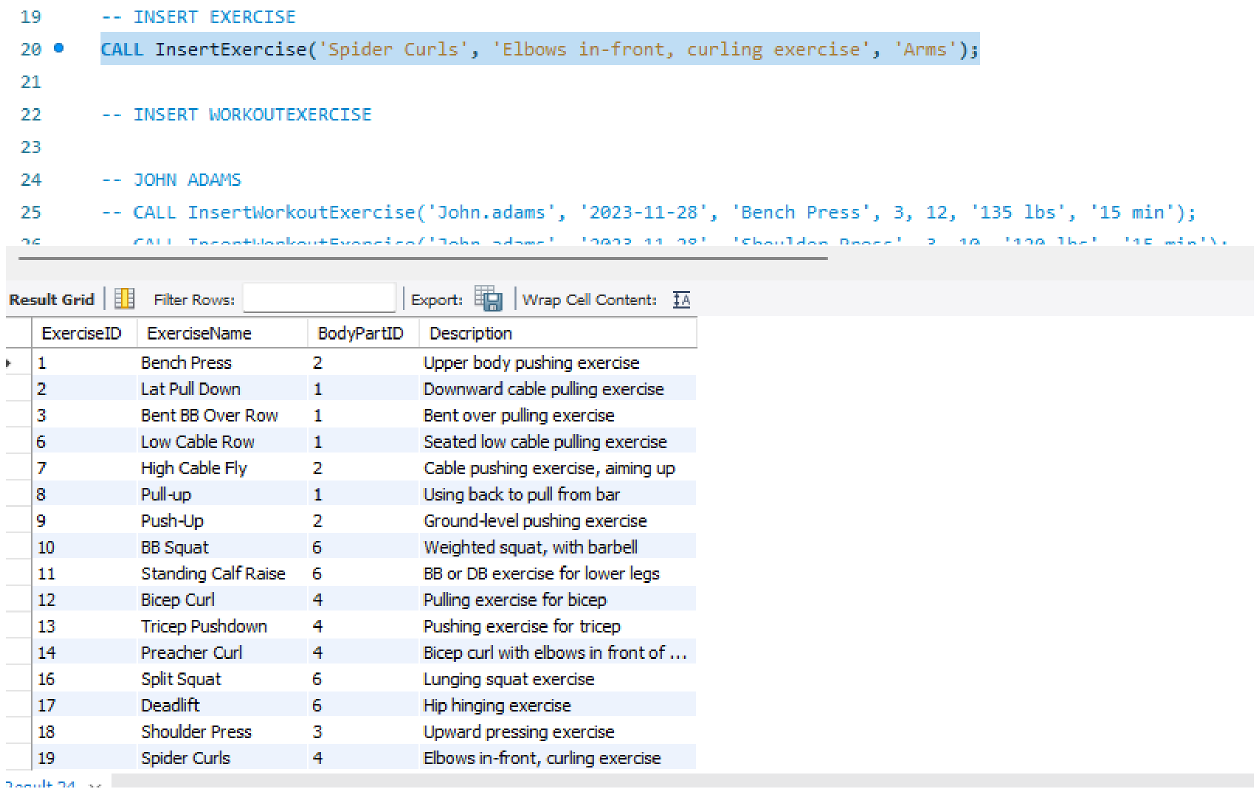
Similarly, the REVIEWPASTWORKOUTSBYBODYPARTNAME and REVIEWPASTWORKOUTSEXERCISENAME procedures will return tables containing all of the relevant exercise information for any exercise completed under the particular criteria. These allow users to track their progression of particular exercises and muscle groups over a period of time. The figures below demonstrate example tables generated.



```
-- REVIEW BY EXERCISE NAME AND ID
-- Review past workouts by exercise name
CALL ReviewPastWorkoutsByExerciseName('Thomas.jefferson', 'Bicep Curl');

-- ----------------------------------- DELETE QUERIES --------------------
-- Delete user with UserName = 'john_doe'
   CALL DeleteUser('Ron.frank');
```

Date	WorkoutType	ExerciseName	Sets	Reps	Weight	Duration
2023-12-05	Strength	Bicep Curl	3	12	35 lbs	15 min
2023-11-28	Strength	Bicep Curl	3	12	25 lbs	15 min

Finally, in the instance that a User deletes, modifies, or adds an exercise, they will be shown a table containing all exercises available for the database. As well as descriptions for each exercise.
The manual of queries above also contains -- SELECT * FROM workoutdb.exercise; in case the user chooses to simply view the exercises currently available in the database.



```
-- INSERT EXERCISE
CALL InsertExercise('Spider Curls', 'Elbows in-front, curling exercise', 'Arms');

-- INSERT WORKOUTEXERCISE

-- JOHN ADAMS
-- CALL InsertWorkoutExercise('John.adams', '2023-11-28', 'Bench Press', 3, 12, '135 lbs', '15 min');
   CALL InsertWorkoutExercise('John.adams', '2023-11-28', 'Shoulder Press', 3, 10, '120 lbs', '15 min');
```

ExerciseID	ExerciseName	BodyPartID	Description
1	Bench Press	2	Upper body pushing exercise
2	Lat Pull Down	1	Downward cable pulling exercise
3	Bent BB Over Row	1	Bent over pulling exercise
6	Low Cable Row	1	Seated low cable pulling exercise
7	High Cable Fly	2	Cable pushing exercise, aiming up
8	Pull-up	1	Using back to pull from bar
9	Push-Up	2	Ground-level pushing exercise
10	BB Squat	6	Weighted squat, with barbell
11	Standing Calf Raise	6	BB or DB exercise for lower legs
12	Bicep Curl	4	Pulling exercise for bicep
13	Tricep Pushdown	4	Pushing exercise for tricep
14	Preacher Curl	4	Bicep curl with elbows in front of ...
16	Split Squat	6	Lunging squat exercise
17	Deadlift	6	Hip hinging exercise
18	Shoulder Press	3	Upward pressing exercise
19	Spider Curls	4	Elbows in-front, curling exercise