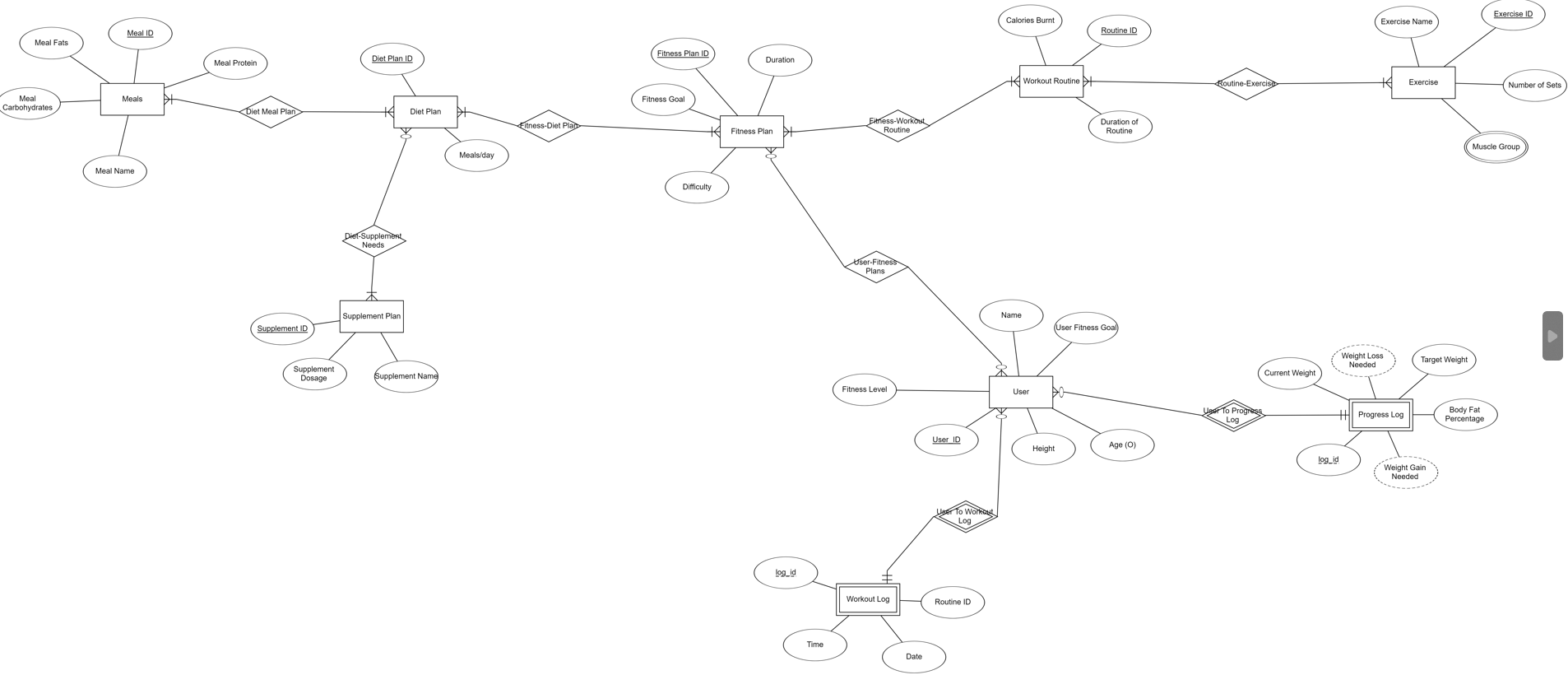
**Documentation**

**Personal Fitness Trainer App**



**ERD Explanation**

This entity-relationship diagram explains the working of a database that pertains to an application that helps its users manage their personal fitness. The database consists of a user entity which has personal attributes like name, age, height, user’s fitness goal and his current fitness level. Whenever a user is created, it has two weak entities associated with it namely workout log and progress log. Workout log is a logger used to track the kind of workouts the user is doing and when he/she is doing it. Progress log is to track personal achievements and progress corresponding to weight loss/weight gain/body fat percentage. A user can have many progress log and workout log entries and each entry corresponds to only one and one user.

The user can also select a number of different fitness plans. The fitness plan entity has a many to many relationship with the user. This entity holds information regarding the difficulty, duration, goal of the fitness plan. One thing to note is, a user can only have one fitness plan active at a time. This activated plan’s ID is stored in the relationship connecting both these entities.

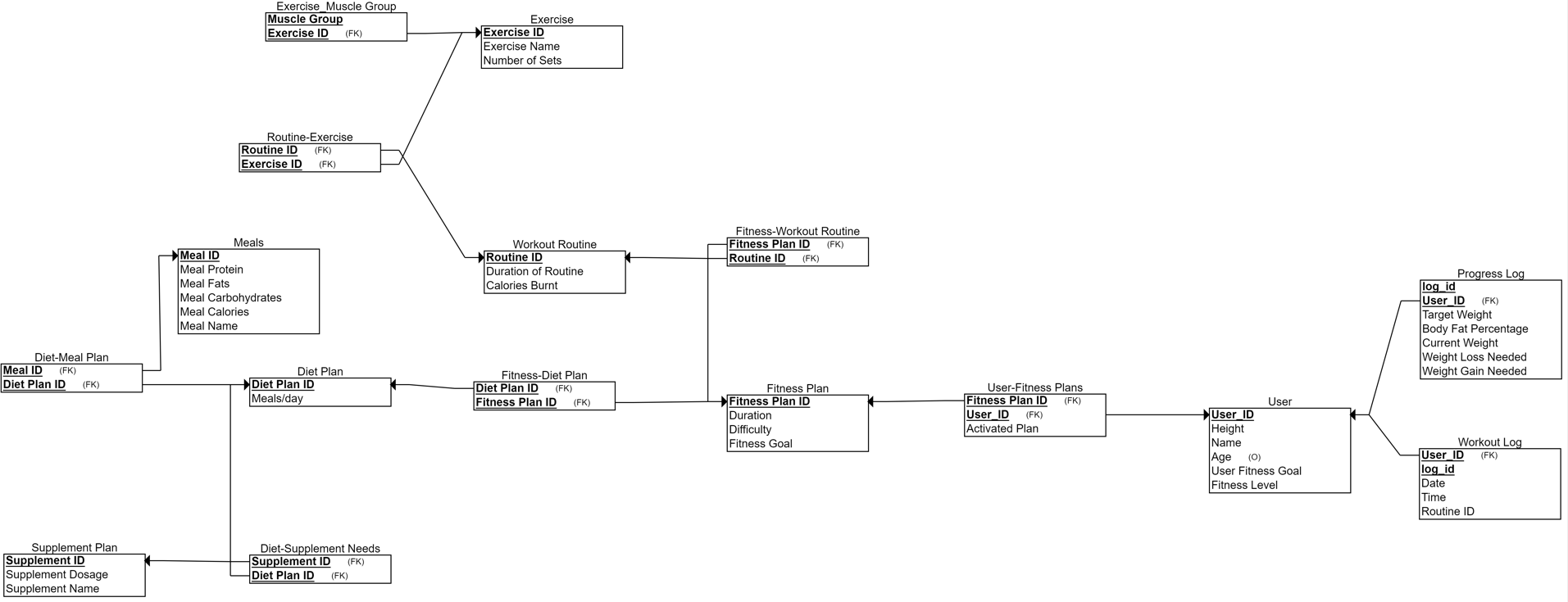
Once you have chosen a particular fitness plan, there are a number of diet plans to choose from. Each diet plan comes with the number of meals the user can eat in a day so that the user selects his/her choice accordingly. Also, once a diet plan has been selected, the user can access a list of meals, supplements recommended by us. The meal entity has a many to many relationship with the entity diet plan and contains the names of the foods, their caloric content (for calorie intake tracking) and their macro nutrient data. The supplement entity has a similar relationship and contains data about names of supplements and dosage needed for the user.

Also, each fitness plan comes with a list of workout routines that the user can choose from as he/she wishes. Each workout routine comes with info regarding its duration and the calories burnt following it.

Once chosen, you can access a list of exercises contained in the workout routine. The exercise entity has a many to many relationship with the workout routine entity and shows information regarding the exercise name, the number of sets one needs to perform and the muscle group(s) it affects. Muscle group is a multivalued attribute and accordingly has a separate table in the database.

In conclusion, using the database, we provide the user the functionality to have complete control over their fitness journey, log every single day of progress and use our expertly curated data to be as efficient as possible .

**Relational Schema**



The shown relational schema corresponds to the ERD explained above. Tables User-Fitness Plans, Fitness-Diet Plan, Diet-Supplement Needs, Diet-Meal Plan, Routine-Exercise, Fitness-Workout Routine exist to connect the entities with a many to many relationship. The table Exercise-Muscle group is made separately as muscle group is a multivalued attribute of the entity Exercise.

The functional dependencies are as follows :-

**User**

User ID-> Height, Name, Age, Fitness Goal, Fitness Level

**Workout Log**

Log ID, User ID -> Date, Name, Routine ID

**Progress Log**

Log ID, User ID -> Target Weight, Current Weight, Body Fat Percentage, Weight Loss Needed, Weight Gain Needed

**User-Fitness Plan**

Fitness Plan ID, User ID -> Activated Plan

**Fitness Plan**

Fitness Plan ID -> Duration, Difficulty, Fitness Goal

**Fitness-Workout Routine**

Fitness Plan ID, Routine ID -> {No non key attributes}

**Fitness-Diet Plan**

Fitness Plan ID, Diet ID -> {No non key attributes}

**Diet Plan**

Diet Plan ID -> Meals/Day

**Workout Routine**

Routine ID -> Duration, Calories Burnt

**Diet-Meal Plan**

Diet ID, Meal ID -> {No non key attributes}

**Exercise**

Exercise ID -> Exercise Name, Number of sets

**Meals**

Meal ID -> Meal Protein, Meal Carbohydrates, Meal Fats, Meal Name, Meal Calories

**Diet-Supplement Needs**

Diet ID, Supplement ID -> {No non key attributes}

**Supplement Plan**

Supplement ID -> Supplement Name, Supplement Dosage

**Routine-Exercise plan**

Routine ID, Exercise ID -> {No non key attributes}

**Exercise Muscle Group**

Exercise ID, Muscle Group -> {No non key attributes}

**Proving 3NF**

The tables have been normalized to at 3NF. Here are a few examples of normalization:

-- Each table has a primary key, and no non-key attributes depend on only part of the primary key.

-- The "user\_fitness\_goal" attribute in the "user" table is a foreign key that references the "fitness\_goal" attribute in the "fitness\_plan" table, which helps to avoid data redundancy.

-- The "plan\_id" attribute in the "user\_fitness\_plans" table is a foreign key that references the "plan\_id" attribute in the "fitness\_plan" table, which helps to avoid data redundancy.

-- The "exercise\_id" attribute in the "routine\_exercise" table is a foreign key that references the "exercise\_id" attribute in the "exercise" table, which helps to avoid data redundancy.

Also, the following properties hold true for the schema:

1. It is in 2NF

2. There are no transitive dependencies

First, to check if it is in 2NF, we need to make sure that all non-key attributes depend on the whole primary key. In each table all non-key attributes depend on the entire primary key. Therefore, the schema satisfies the condition for 2NF.

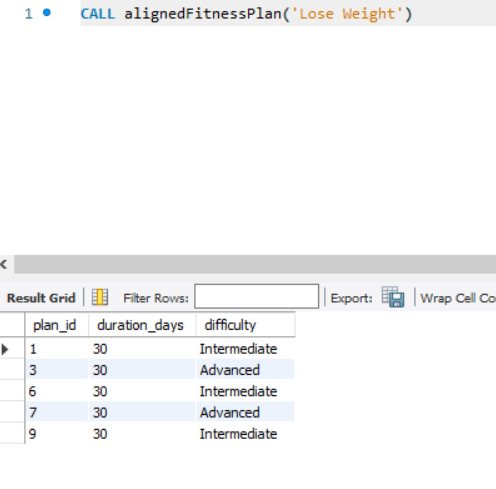
Next, to check if there are any transitive dependencies, we need to check if any non-key attribute depends on another non-key attribute. In this schema, there are no transitive dependencies because each non-key attribute depends only on the primary key of the table it belongs to. Therefore, the schema satisfies the condition for 3NF.

Therefore, we can conclude that this schema is in 3NF.

**Procedure Explanation**

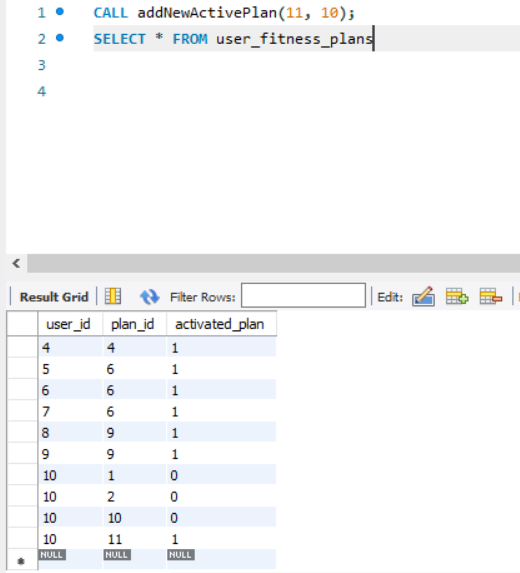
1. **alignedFitnessPlan(IN goal VARCHAR(100))**

This procedure returns the fitness plans associated with the particular goal in parameter. It takes one input parameter "goal" of type VARCHAR with a maximum length of 100 characters. The main purpose of this procedure is to select and retrieve the "plan\_id", "duration\_days", and "difficulty" from the table "fitness\_plan" where the "fitness\_goal" matches the input parameter "goal".



1. **addNewActivePlan(IN fitness\_plan\_id INT, IN id INT)**

This procedure is used to add a new fitness plan to the user's current list of plans and make it his/her active plan. It takes two input parameters "fitness\_plan\_id" and "id" of type INT. The main purpose of this procedure is to update the "activated\_plan" field in the "user\_fitness\_plans" table to 0 for the given "user\_id" (provided as input "id") and then insert a new row into the same table with the provided "fitness\_plan\_id" and "user\_id" values and set the "activated\_plan" field to 1.



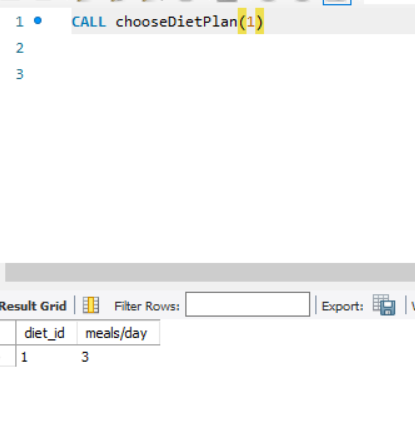
1. **chooseDietPlan(IN userID INT)**

This procedure shows the diet\_plan to choose from corresponding to user's current fitness plan. The procedure takes one input parameter "userID" of type INT.

The main purpose of this procedure is to select and retrieve the "diet\_id" and "meals/day" columns from the "diet\_plan" table for a diet plan that matches the activated fitness plan of the provided user ID.

This is achieved by joining the "diet\_plan" table with the "fitness\_diet\_plan" table on their common attribute "plan\_id". Then, a subquery is used to retrieve the "plan\_id" of the activated fitness plan for the provided "userID" from the "user\_fitness\_plans" table.

Finally, the outer query filters the joined result by matching the retrieved "plan\_id" with the "plan\_id" from the "fitness\_diet\_plan" table.



1. **muscleWiseRoutine(IN userID INT ,IN muscle\_group VARCHAR(100))**

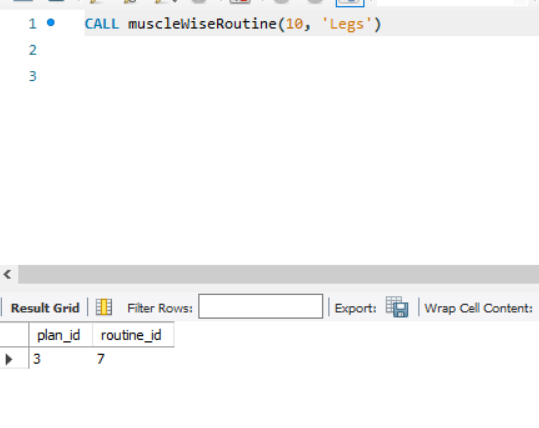
This procedure shows the workout routines available corresponding to the muscle group the user wants to train if available in his own fitness plan.

The query selects the "plan\_id" and "routine\_id" columns from the "user\_fitness\_plans" table, and joins this table with the "fitness\_workout\_routine" table on their common attribute "routine\_id".

The outer query then filters the joined result by selecting only the rows where the "routine\_id" matches any of the values returned by the subquery.

The subquery retrieves the "routine\_id" values by joining the "exercise\_muscle\_group" table with a temporary table that is created by joining the "exercise" and "routine\_exercise" tables. This is done by using the "NATURAL JOIN" operator, which joins the tables based on their common attributes.

The subquery then filters the result by selecting only the rows where the "muscle\_name" from the "exercise\_muscle\_group" table matches the "muscle\_group" provided by the caller.

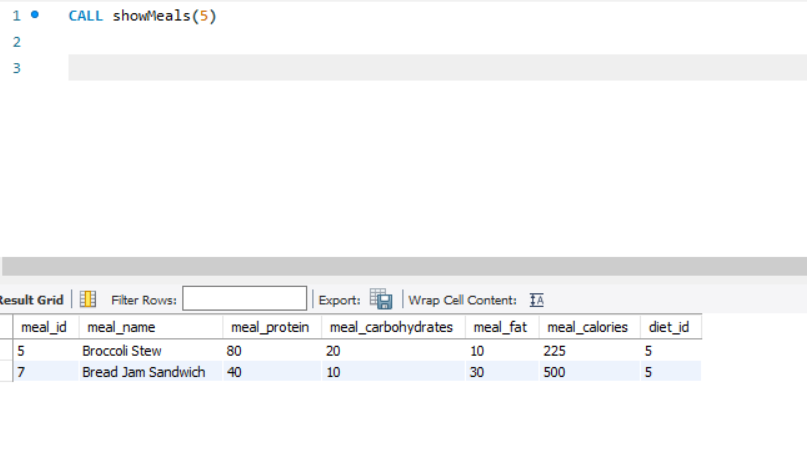


1. **showMeals(IN dietID INT)**

This procedure is used to meals corresponding to a diet plan.

The query selects all columns from the "meals" table and joins it with the "diet\_meal\_plan" table on their common attribute "meal\_id". This is done by using the "NATURAL JOIN" operator, which joins the tables based on their common attributes.

The outer query then filters the joined result by selecting only the rows where the "diet\_id" from the "diet\_meal\_plan" table matches the "dietID" provided by the caller.



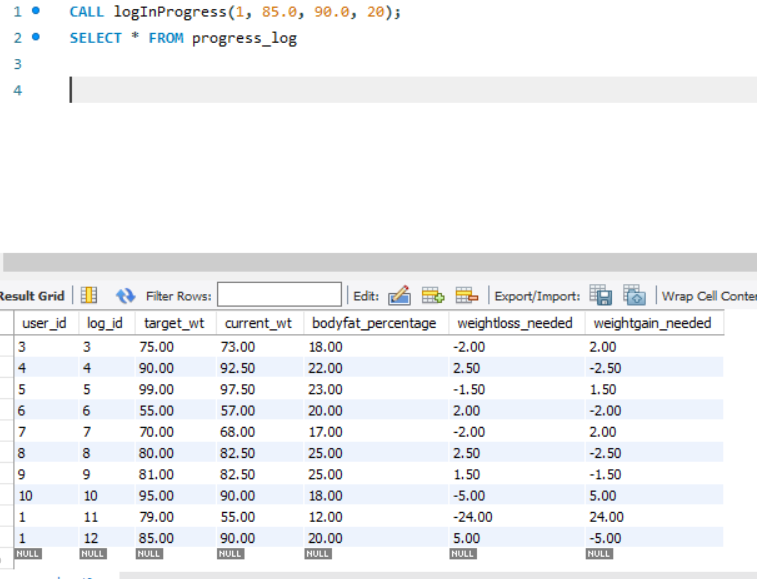
1. **logInProgress(IN userID INT, IN target DECIMAL(4,2), IN currentwt DECIMAL(4,2), IN bodyfat DECIMAL(4,2))**

This procedure is used to log user progress in the progress\_log table of the database.

The query inserts a new row into the table with values provided by the caller. The values are inserted into the "user\_id", "target\_wt", "current\_wt", and "bodyfat\_percentage" columns, respectively.

The values provided by the caller are referenced using input parameters "userID", "target", "currentwt", and "bodyfat", respectively.

Overall, this query inserts a new row into the "progress\_log" table with the values provided by the caller. The query assumes that the table "progress\_log" already exists and has the columns "user\_id", "target\_wt", "current\_wt", and "bodyfat\_percentage".

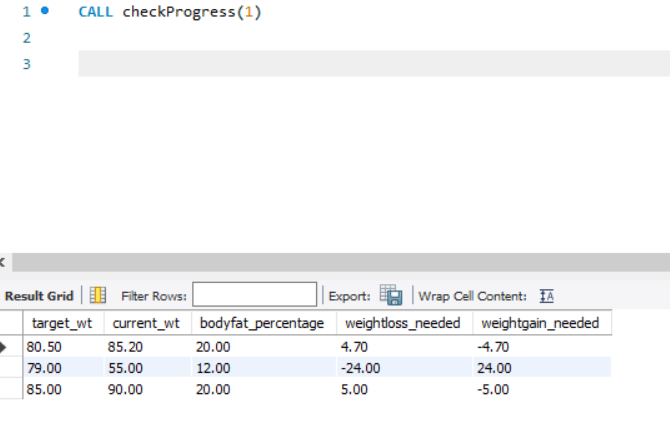
****

1. **checkProgress(IN userID INT)**

This procedure is used to check the user's progress log. The query selects the "target\_wt", "current\_wt", "bodyfat\_percentage", "weightloss\_needed", and "weightgain\_needed" columns from the "progress\_log" table. It joins this table with a subquery that selects the maximum "log\_id" for each "user\_id" in the "progress\_log" table. The subquery is performed by selecting the maximum "log\_id" and grouping the results by "user\_id". The subquery also filters the results by selecting only the rows where the "user\_id" matches the "userID" provided by the caller.

The outer query then joins the "progress\_log" table with the results of the subquery on their common attributes "user\_id" and the "MAX(log\_id)" returned by the subquery.

Overall, this query retrieves the latest progress data (i.e., the highest "log\_id") for a specific user, where the user is identified by the "userID" provided by the caller. The query also calculates the "weightloss\_needed" and "weightgain\_needed" by subtracting the "current\_wt" from the "target\_wt".

****

1. **showHealthyMealsCorrespondingToUser(IN userID INT)**

This procedure shows the meals that the current user can take acc to his currenly active fitness plans and the diet plans associated it.

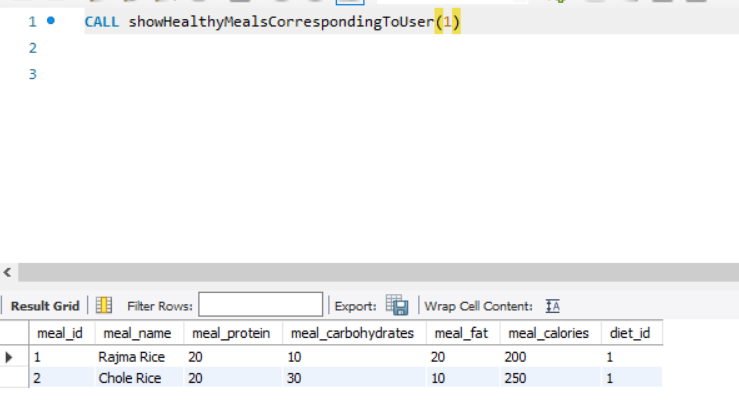
The query selects all columns from the "meals" table and joins it with the "diet\_meal\_plan" table on their common attribute "meal\_id". This is done by using the "NATURAL JOIN" operator, which joins the tables based on their common attributes.

The query then filters the joined result by selecting only the rows where the "diet\_id" from the "diet\_meal\_plan" table matches the "diet\_id" returned by a subquery.

The subquery selects the "diet\_id" from the "fitness\_diet\_plan" table where the "plan\_id" matches the "plan\_id" returned by another subquery.

The innermost subquery selects the "plan\_id" from the "user\_fitness\_plans" table where the "user\_id" matches the "userID" provided by the caller and "activated\_plan" is equal to 1.

Overall, this query retrieves all the meals for the currently activated diet plan of a specific user, where the user is identified by the "userID" provided by the caller. The query assumes that the tables "meals", "diet\_meal\_plan", "fitness\_diet\_plan", and "user\_fitness\_plans" already exist and have the appropriate columns.



1. **showFitnessLevelExercises(IN userID INT)**

This procedure shows the exercises that the current user can train with his current level of fitness. The query selects the "name" and "sets" columns from the "exercise" table and joins it with a set of subqueries using the "JOIN" operator.

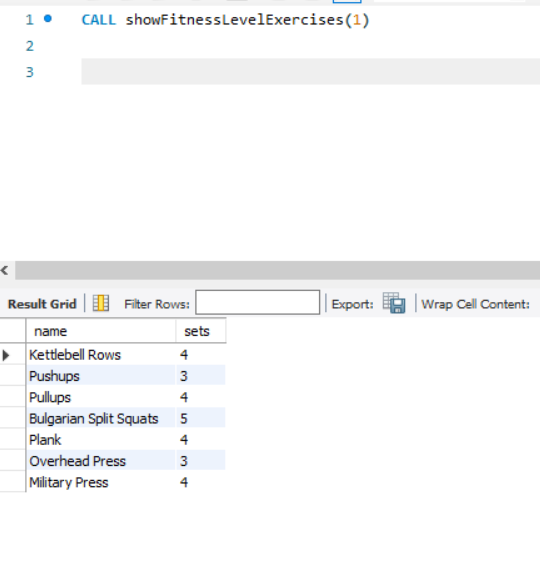
The innermost subquery selects all columns from the "user" table and joins it with the "fitness\_plan" table on their common attribute "difficulty". The subquery then filters the joined result by selecting only the rows where the "user\_id" matches the "userID" provided by the caller.

The next subquery joins the results of the innermost subquery with the "fitness\_workout\_routine" table on their common attribute "plan\_id". The subquery then selects all columns from the joined result.

The third subquery joins the results of the second subquery with the "routine\_exercise" table on their common attribute "routine\_id". The subquery then selects all columns from the joined result.

The outermost query joins the results of the third subquery with the "exercise" table on their common attribute "exercise\_id". The query then selects the "name" and "sets" columns from the joined result and eliminates any duplicate rows using the "DISTINCT" keyword.

Overall, this query retrieves the names and sets of all exercises in the currently activated workout routine of a specific user, where the user is identified by the "userID" provided by the caller. The query assumes that the tables "exercise", "routine\_exercise", "fitness\_workout\_routine", "fitness\_plan", and "user" already exist and have the appropriate columns.



1. **addNewWorkoutRoutine(IN userID INT, IN fplan\_id INT, IN routineID INT)**

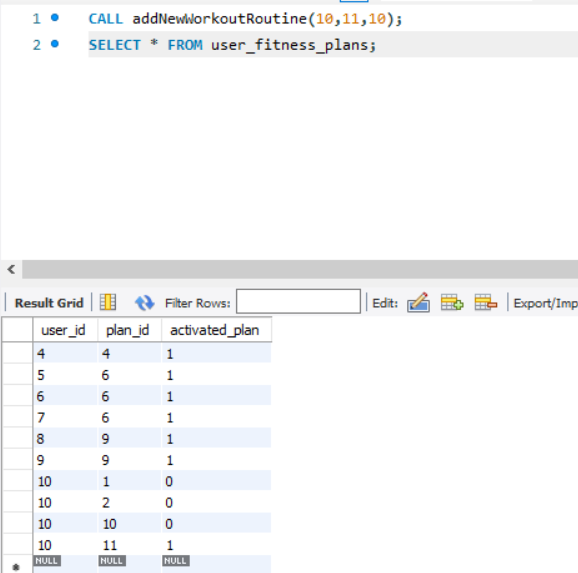
This procedure is used to add a new workout routine int one of the fitness plans that our user has.

The query uses the "INSERT INTO" statement to insert a new row into the "fitness\_workout\_routine" table. The "VALUES" keyword specifies the values to be inserted into the columns of the table.

The first value specified is "routine\_id", which is assumed to be a variable that holds the ID of the new workout routine to be added.

The second value specified is a subquery that retrieves the "plan\_id" of the currently activated fitness plan of a specific user, where the user is identified by the "userID" provided by the caller. The subquery uses the "SELECT" statement to retrieve the "plan\_id" column from the "user\_fitness\_plans" table and filters the result by selecting only the rows where the "user\_id" matches the "userID" provided by the caller.

Overall, this query adds a new workout routine with a specific ID to the currently activated fitness plan of a specific user, where the user is identified by the "userID" provided by the caller. The query assumes that the "fitness\_workout\_routine" and "user\_fitness\_plans" tables already exist and have the appropriate columns.



1. **showMeSupplements(IN userID INT)**

This procedure shows the list of supplements recommended by the app to our user corresponding to his fitness goal. The query uses the "SELECT" statement to retrieve data from the "supplement\_plan" table, joining it with three other tables using the "NATURAL JOIN" operator.

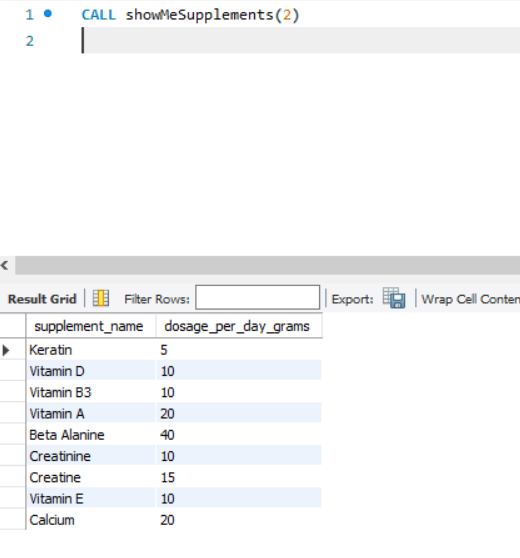
The first subquery used in the "NATURAL JOIN" chain retrieves data from the "diet\_supplement\_needs" table, which contains information about the supplements needed for a specific diet plan.

The second subquery retrieves data from the "user\_fitness\_plans" table, which contains information about the currently activated fitness plan of a specific user. This subquery also joins with the "fitness\_diet\_plan" table, which contains information about the diet plans associated with the fitness plans.

The final "NATURAL JOIN" operator combines the results of the previous subqueries with the "supplement\_plan" table, which contains information about the supplements available and their recommended dosages.

The "DISTINCT" keyword is used to remove any duplicate rows from the result set.

Overall, this query retrieves a list of supplements and their corresponding dosages per day in grams based on the user's currently activated fitness plan and diet supplement needs.



**FAQ**

1. **Can you help me create a new fitness plan that aligns with my goals?**

Yes, just enter your details and select your fitness goal. Then click on show fitness plans to start a fitness plan of your choice and get fit!

1. **I want to change my current fitness plan. How can I do that in the system?**

If you want to change your current fitness plan, just add a new plan after going back to the homepage. This new plan will now serve as your active plan and would suggest your everything you need in your fitness journey.

1. **Can you suggest a workout routine that targets specific muscle groups?**

Yes, you can train specific muscle groups by selecting one of our workout routines. Each workout routine comes with exercises that target specific muscles. You can get a list of such exercises by clicking on display exercises. Otherwise you can also click on the show muscle specific routines, to get the list of workout routines that train the part you want to train

.

1. **I want to track my progress toward my fitness goals. Is there a way to do that in the system?**

Yes, you can keep a track of your progress using the progress log feature. Just log in your data whenever you see some progress. You can check your progress anytime you want using the check progress feature. Moreover, you can even make use of our workout log feature to maintain a record of your workout sessions.

1. **Can you recommend some healthy meals that align with my dietary preferences?**

Yes, whenever you select a fitness plan, you can choose one of our diet plans suited for your body type and your fitness goals. You can click on the display meals button to get a list of healthy meals that align with your preferences.

1. **I want to track my calorie intake. Is there a feature in the system for that?**

Yes, whenever you choose a diet plan and look at the corresponding meals we suggest to you, you can also check the calories they hold before having them.

1. **Can you suggest exercises that are suitable for my fitness level?**

Yes, you can simply add your fitness level (whether it be beginner, intermediate or advanced) while putting in your user details. We will only suggest you fitness plans that match your fitness level and goal from our program.

1. **I want to add a new workout routine to my fitness plan. How can I do that in the system?**

Whenever you select your fitness plan, you will get a bunch of options for the workout routine you can choose. Just select one of them and start working out!

1. **Can you recommend supplements that will help me achieve my fitness goals?**

Sure! Pertaining to your body details and your diet plan, we suggest you the best supplements you should be using. Just click on the recommend supplements button to get a list of these supplements.

1. **I want to track my body measurements, such as my weight and body fat percentage. Is there a way to do that in the system?**

Yes, you can make use of our progress log section to save such details for future references.

1. **Can you suggest exercises that are suitable for my current injuries or limitations?**

Yes, just click on the show exercises button to get a list of exercises suitable for you.