CPSC 304 Project Cover Page

Milestone #: 4

Date: November 29th 2024

Group Number: 38

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Alex Luo	17603341	k7i1t	alexluo602@gmail.com
Jerrold Huang	26238998	k5l2e	zhonghan.huang@outlook.com
Jason Liao	67122887	V3d9u	jasonliao999@gmail.com

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above. (In the case of Project Milestone 0, the main purpose of this page is for you to let us know your e-mail address, and then let us assign you to a TA for your project supervisor.)

In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia

2. Brief Summary:

We completed the implementation of our app, a personal health management app. Functionalities include

- Daily macro intake logging and monitoring
- Tracking different goals based on the user's situation (height, weight, etc.)
- Guidance on recipes based on a user's health situation
- Simple analytics on user's exercises
- Insight into other user's stats

3. Schema Description:

The schema is the same other than the fact that we have de-normalized it. We did this because the normalized version of the database made it so that it's less user-friendly - so we decided to go with the original unnormalized schema proposed in milestone 3.

3. SQL Queries:

2.1.1 Insert: Add a new Ingredient row

INSERT INTO Ingredient (name, brand, taste, food_type, macro_id) VALUES (:name, :brand, :taste, :food_type, :macro_id)

- Macro_id is a foreign key that is also unique
- Can be found in /project/rest/Ingredient/IngredientService.js
- Line: 43

2.1.2 UPDATE: Update a field chosen by the user in userInfo

```
`UPDATE UserInfo SET ${valClause} where user_id=:user_id`,
[...values, user_id],
```

- Username is unique
- Can be found in /project/rest/user/userService.js
- Line: 65

2.1.3 DELETE: Deleting a recipe also automatically deletes all the steps referencing it

DELETE FROM Recipe WHERE recipe_id=:recipe_id

- Can be found in /project/rest/recipe/recipeService.js
- Line: 69

2.1.4: Selection: Select Ingredients given user input

- Can be found in /project/rest/Ingredient/IngredientService.js
- Line 101: Where the query is executed
- Lines: 98-104: responsible for executing the query
- Lines: 107-171: helper function to build the query by parsing user input

2.1.5: Projection: Project Ingredient fields:

- Can be found in /project/rest/Ingredient/IngredientService.js
- Line 177: where the query can be found
- Lines: 173-191: responsible for executing the query to get the selected fields

2.1.6 Join: Find all of the meals that a given user has eaten

```
SELECT mr.meal_record_id, mr.meal_record_date, mr.recipe_id, r.name AS
      recipe_name
FROM MealRecord mr
JOIN Recipe r ON mr.recipe_id = r.recipe_id
WHERE mr.user_id = :userId
```

- Can be found in /project/rest/meals/mealRecordService.js
- Lines: 92-95

2.1.7 Aggregation with Group by: Calculate the average calories burned in each type of exercise

```
SELECT type, AVG(calories_burned) AS avg_calories
      FROM ExerciseRecord
      GROUP BY type
```

- Can be found in /project/rest/exercise/exerciseService.js
- Lines: 40-42

2.1.8 Aggregation with Having: Finds users who have more than 3 meals

```
SELECT ui.user_id, ui.username, COUNT(mr.meal_record_id) AS meal_count
      FROM UserInfo ui
      JOIN MealRecord mr ON ui.user_id = mr.user_id
      GROUP BY ui.user_id, ui.username
      HAVING COUNT(mr.meal_record_id) > 3
      ORDER BY meal_count DESC
```

- Can be found in /project/rest/user/userService.js
- Lines: 106-111

2.1.9 Nested aggregation with GROUP BY: This query retrieves the types of exercises (from the ExerciseRecord table) that burn more calories on average than the average total calories burned by all exercise types.

```
SELECT E.type, SUM(E.calories_burned) AS total_calories_burned
FROM ExerciseRecord E
GROUP BY E.type
```

```
HAVING SUM(E.calories_burned) > (
   SELECT AVG(total_calories)
   FROM (
      SELECT SUM(E2.calories_burned) AS total_calories
      FROM ExerciseRecord E2
      GROUP BY E2.type
   ) sub_totals
)
```

- Can be found in /project/rest/exercise/exerciseService.js
- Lines: 107-117

2.1.10 Nested aggregation with GROUP BY: Identify user_ids who have done all of the exercises.

```
SELECT er.user_id
FROM ExerciseRecord er
GROUP BY er.user_id
HAVING COUNT(DISTINCT er.type) = (
   SELECT COUNT(DISTINCT type) FROM ExerciseRecord
)
```

- Can be found in /project/rest/user/userService.js
- Lines: 129-134