CPSC 304 Project Cover Page

Milestone #: 4

Date: November 29th, 2024

Group Number: 58

Name	Student Number	CS Alias (User ID)	Preferred Email Address
Hediyeh Mahmoudian	15990880	g3i2f	hediemahmoudian@gmail.com
Oreoluwa Akinwunmi	10711489	T8j3b	oreakinwunmi@yahoo.com
Helena Sokolovska	37576162	f3e0f	hesoru@gmail.com

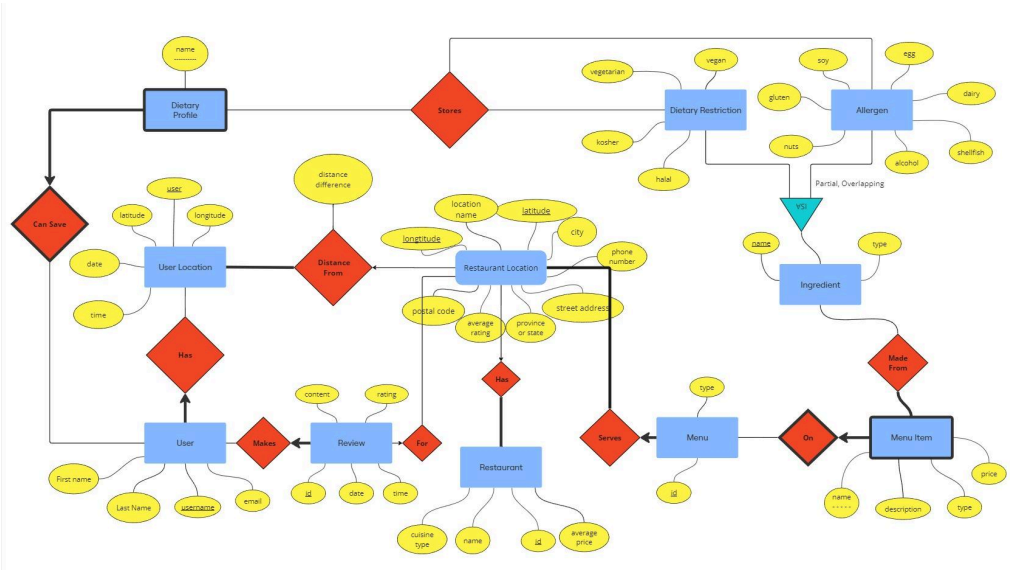
By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above.

In 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.

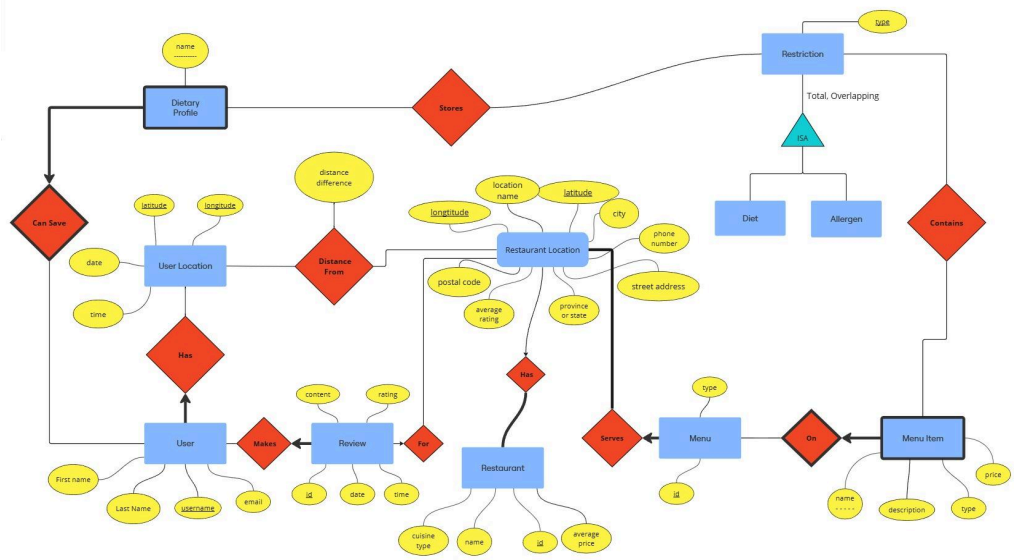
Project Summary: This is a database application catered to UBC students to help them navigate all the different restaurants available on campus. It allows users to search for restaurants nearby based on proximity, menu options, food allergens, dietary restrictions, cuisine type, and affordability.

Schema Differences: Our final schema did differ from what was turned in originally in Milestone 2. Images of the previous ER diagram and what our final project was based on will be attached below, along with specific descriptions of exactly which tables were changed and how:

VERSION 1 - ORIGINAL



VERSION 2 - FINAL



CHANGES

1) User-Location PK Changed

a. We changed the PK from **User** to **longitude** and **latitude** as there was no need to reiterate a user considering the User table had a direct relationship with the User_Location table

2) Distance From Total Participation Removed

a. We changed the relationship of User Location with Restaurant_Location so it is no longer total participation. This is because a User Location need not necessarily be associated with a Restaurant Location at all times. We are instead making it so the “Distance From” relationship is updated ONLY when a user logs into the application and/or clicks on a particular restaurant.

3) Distance From Key Constraint Removed

a. We removed the key constraint on Restaurant Location and replaced it with a many-to-many relationship with User Location. We realized a Restaurant Location will not necessarily be associated with at most 1 user location, rather, it will calculate the distance between MULTIPLE user locations if multiple people are using the app simultaneously. Alternatively, it could also not be associated with any User Locations if a user has not logged into the app or clicked on a particular restaurant.

4) Changed ISA Relationship

a. We altered our ISA so a Dietary profile is now associated with a Restriction that MUST be either (or both) a diet (e.g. kosher, halal, etc.) and/or an allergen (e.g. gluten, soy, etc.). Furthermore, we removed Ingredient and the Made From relationship, in place of Restriction having a direct relationship with Menu Item - i.e. a menu item could contain many restrictions and vice versa. This will make our query processing much easier.
b. This also means we will have a total of **4 tables** for **Diet & Allergen**, as we need 2 tables to present which Menu_Item tuples contain a dietary restriction and allergen respectively. We also need 2 tables to represent which Dietary_Profile has which dietary restriction and allergies, respectively. A Restriction table will not be necessary due to the Total, Overlapping constraint in this ISA.

SQL QUERIES: ALL SQL queries can be found in the **appService.js** file, located in the ROOT directory of the project. Each query and it's relevant *starting* line number will be listed below:

/* NOTE: Due to restrictions of Oracle, we could not use "on delete cascade" OR "on update cascade" */

1. **INSERT** - LINE 95 (addUserProfile). Adds user sign-up data to database (User_Has).

   ```
   INSERT INTO
         User_Has (Username, First_Name, Last_Name, Email,
         User_Longitude, User_Latitude)
   VALUES
         (:username, :first_name, :last_name, :email, :longitude, :latitude);
   ```

2. **UPDATE** - LINE 335 (updateReviewContent). Allows user to update review content (also sets new date/time) for a given review ID.

   ```
   UPDATE
         Review_For_Makes
   SET
         ${columnName} = :newContent,
         Record_Date = SYSDATE,
         Record_Time = SYSTIMESTAMP
   WHERE
         Id=:reviewID;
   ```

3. **DELETE** - LINE 360 (deleteReviewContent). Deletes review with given review ID from Review_For_Makes.

   ```
   DELETE FROM
         Review_For_Makes
   WHERE
         ID =:reviewID;
   ```

4. **SELECTION** - LINE 443 (fetchMenuProfile). Finds menu items (with restaurant name and price) that meet dynamic diet and allergen conditions.

   ```
   SELECT
   ```

```
        mi.Menu_Name,
        r.Restaurant_Name,
        mi.PRICE
    FROM
        CONTAINS_ALLERGEN ca,
        Contains_Diet cd
            JOIN
        Menu_Item_On mi ON cd.Menu_Item_Name = mi.Menu_Name AND
mi.MENU_ID = cd.MENU_ID
            JOIN
        Menu_Serves ms ON mi.Menu_Id = ms.Id
            JOIN
        Restaurant_Location_Has rlh ON ms.Restaurant_Latitude = rlh.Latitude AND
ms.Restaurant_Longitude = rlh.Longitude
            JOIN
        Restaurant r ON r.Id = rlh.Restaurant_Id

    ${whereClause}
    GROUP BY
        mi.Menu_Name, r.Restaurant_Name, mi.PRICE
    ${groupByClause}

    ORDER BY
        mi.Menu_Name;
```

5. **PROJECTION** - LINE 297 (fetchAUserReview). Finds review data (including restaurant location name) for a given review ID.

```
SELECT
    rl.Location_Name,
    rfm.Content AS Review_Content,
    rfm.Rating,
    rfm.Record_Date,
    rfm.Record_Time
FROM
    Review_For_Makes rfm
JOIN
    Restaurant_Location_Has rl
ON
    rfm.Restaurant_Longitude = rl.Longitude
```

```
    AND rfm.Restaurant_Latitude = rl.Latitude
WHERE
    rfm.Id = :reviewID;
```

6. **JOIN** - LINE 500 (fetchRestaurantMenuFromDb). Presents all the menu items for a restaurant location at the given latitude and longitude. Note: aftering joining, duplicates of menu items will exist with different diets and allergens - diets/allergens are consolidated into 1 menu item in the frontend.

```
SELECT
    mi.Menu_Name,
    mi.Description,
    mi.Price,
    d.Diet_Type,
    a.Allergen_Type
FROM
    Menu_Item_On mi
JOIN
    Menu_Serves ms ON mi.Menu_Id = ms.Id
JOIN
    Restaurant_Location_Has rlh ON ms.Restaurant_Latitude = rlh.Latitude AND
ms.Restaurant_Longitude = rlh.Longitude
LEFT JOIN
    Contains_Diet cd ON mi.Menu_Name = cd.Menu_Item_Name AND
mi.Menu_Id = cd.Menu_Id
LEFT JOIN
    Diet d ON cd.Diet_Type = d.Diet_Type
LEFT JOIN
    Contains_Allergen ca ON mi.Menu_Name = ca.Menu_Item_Name AND
mi.Menu_Id = ca.Menu_Id
LEFT JOIN
    Allergen a ON ca.Allergen_Type = a.Allergen_Type
WHERE
    rlh.Latitude = :lat AND rlh.Longitude = :lon;
```

7. **AGGREGATION (GROUP BY)** - LINE 179 (fetchAllRestraurantsFromDB). Finds information about all restaurant locations, grouped by details of each location and restaurant attributes.

```
SELECT
    rl.Location_Name,
    rl.STREET_ADDRESS,
    rl.CITY,
    rl.PROVINCE_OR_STATE,
    rl.POSTAL_CODE,
    rl.PHONE_NUMBER,
    r.Cuisine_Type,
    ROUND(r.Average_Price, 0) AS Average_Price,
    rl.AVERAGE_RATING,
    ROUND(rl.Latitude, 6) AS Latitude,
    ROUND(rl.Longitude, 6) AS Longitude,
    COUNT(*) AS Total_Rows
FROM
    Restaurant_Location_Has rl
JOIN
    Restaurant r ON rl.Restaurant_Id = r.Id
GROUP BY
    rl.Location_Name,
    rl.STREET_ADDRESS,
    rl.CITY,
    rl.PROVINCE_OR_STATE,
    rl.POSTAL_CODE,
    rl.PHONE_NUMBER,
    r.Cuisine_Type,
    ROUND(r.Average_Price, 0),
    rl.AVERAGE_RATING,
    ROUND(rl.Latitude, 6),
    ROUND(rl.Longitude, 6);
```

8. **AGGREGATION (HAVING)** - LINE 149 (fetchFoodFromDescription). This query retrieves a list of menu items whose descriptions or menu item name has the word inputted by the user in the search bar.

```
SELECT MENU_NAME, r.RESTAURANT_NAME, PRICE, DESCRIPTION,
MENU_TYPE, MENU_ID
 FROM MENU_ITEM_ON mi
        JOIN
    Menu_Serves ms ON mi.Menu_Id = ms.Id
        JOIN
```

```
        Restaurant_Location_Has rlh ON ms.Restaurant_Latitude = rlh.Latitude
          AND ms.Restaurant_Longitude = rlh.Longitude
          JOIN
        Restaurant r ON r.Id = rlh.Restaurant_Id
    GROUP BY MENU_TYPE, MENU_ID, MENU_NAME, DESCRIPTION, PRICE,
r.RESTAURANT_NAME
    HAVING (LOWER(DESCRIPTION) LIKE '%' || LOWER(:description) || '%') OR
LOWER(MENU_NAME) LIKE ( '%' || LOWER(:description) || '%')
    ORDER BY MENU_TYPE;
```

9. **NESTED AGGREGATION (GROUP BY**) - LINE 222 (fetchTopRatedByCuisineFromDb). Finds the highest average user-rated restaurant (with cuisine type and average rating) in each cuisine type.

```
SELECT
  R.Cuisine_Type,
  R.Restaurant_Name,
  AVG(RF.Rating) AS Average_Rating
FROM
  Restaurant R
JOIN
  Restaurant_Location_Has RL ON R.Id = RL.Restaurant_Id
JOIN
  Review_For_Makes RF ON RL.Longitude = RF.Restaurant_Longitude AND
RL.Latitude = RF.Restaurant_Latitude
GROUP BY
  R.Cuisine_Type, R.Restaurant_Name
HAVING
  AVG(RF.Rating) = (
    SELECT
      MAX(AVG(RF2.Rating))
    FROM
      Restaurant R2
    JOIN
      Restaurant_Location_Has RL2 ON R2.Id = RL2.Restaurant_Id
    JOIN
      Review_For_Makes RF2 ON RL2.Longitude =
RF2.Restaurant_Longitude AND RL2.Latitude = RF2.Restaurant_Latitude
    WHERE
      R2.Cuisine_Type = R.Cuisine_Type
```

```
        GROUP BY
            R2.Cuisine_Type
    )
ORDER BY
    R.Cuisine_Type, Average_Rating DESC;
```

10. **DIVISION** - LINE 266 (fetchRestaurantsServingAllDietsFromDb). This query gives us the restaurants that serve all dietary preferences, i.e. restaurants offering all dietary types listed in the diet table.

```
SELECT
    R.Restaurant_Name,
    R.Cuisine_Type
FROM
    Restaurant R
JOIN
    Restaurant_Location_Has RL ON R.Id = RL.Restaurant_Id
JOIN
    Menu_Serves MS ON RL.Longitude = MS.Restaurant_Longitude AND
RL.Latitude = MS.Restaurant_Latitude
JOIN
    Contains_Diet CD ON MS.Id = CD.Menu_Id
JOIN
    Diet D ON CD.Diet_Type = D.Diet_Type
GROUP BY
    R.Restaurant_Name, R.Cuisine_Type
HAVING
    COUNT(DISTINCT D.Diet_Type) = (SELECT COUNT(*) FROM Diet);
```