Group 3

Team Name: Undercooked

Team Members: Nicholas Deary, Alex Iacob, Alex Lawrence, Benson Yan

Main Domain: Recipe

Project Description:

For this project, we aim to apply the domain knowledge through JavaFX application. We would like to use PostgresSQL to connect to the database and create Java classes to handle the back-end server manipulation.

Phase 2 Update:

The ER model was created with a total of six entities; User, Recipe, Recipe Category, Recipe Ingredient, Item, and Item_Aisle. The User entity contains four attributes. Of these four attributes, the Username is the key attribute. The Recipe entity contains ten attributes and two weak entities. Of these ten attributes, the Recipe name and Author are the key attributes while the Author is also a foreign key from the User. The Recipe Category and Recipe Ingredient entities are weak entities to Recipe and are dependent on its key attributes. The Item entity contains five attributes and one weak entity. Of the five attributes, Item Name and Purchase Date are key attributes. Item_Aisle is a weak entity dependent on the key attributes from the Item entity. All entities were connected by many to many relationships, and one other attribute, quantity, was put on the "Made Of" relationship between Item and Recipe.

The reduction to tables was done in two steps. First, we took the model entities with their attributes and created tables from them. Then we took the relationships with their own attributes and the key attributes of the entities they were linking to make tables from them.

Phase 3 Update:

During the beginning of phase 3, a few changes were made to the ER model. The ER model is now finalized with a total of five entities; Chefs, Recipe, Recipe Category, Item, and Item_Aisle. User entity is replaced by Chef to avoid confusion with SQL keywords, otherwise, everything stays the same. The attribute Domain was removed from the Recipe entity and Recipe now has a total of nine attributes. Recipe Name is no longer a key attribute, so the Recipe entity has narrowed it down to one key attribute and it is called Recipe ID. Chefs and Recipes are put on "Creates" relationships. The Recipe Category entity contains one key attribute which is the category name. Recipe and Recipe Category are put on "Part of" relationships. Item has two attributes which are Item ID and Item Name. Of these two attributes, Item ID is the key attribute. Item is also connected with Recipe with the relationship of "Made of" and one other attribute, quantity. Similarly, Item has another "Owns" relationship with Chefs. Through this relationship, it has attributes: Purchase Date, Expiration Date, Quantity Bought, and Current Quantity. Lastly, Item_Aisle has two attributes: Aisle ID and Aisle Name. Of these two attributes, Aisle ID is the key attribute. Item_Aisle has the relationship of "Stored in" with Item.

The reduction to tables was done in the same way as the previous phase. Tables are created for each model entity followed by their own attributes. Tables for relationships between entities are also linked by their key attributes.

Samples of SQL statements used to create tables:

1. create table chefs

```
(
        username            varchar(32)     not null
                    Constraint user_pk
                            primary key,
        password            varchar(32)     not null
        creation_date       date            not null
        creation_time       time            not null
        last_access_date    date            not null
        last_access_time    time            not null
);
```

2. create table item

```
(
        Item_id         integer         not null
        Item_name       varchar(64)     not null
);
```

3. create table owns

```
(
        purchase_date       date            not null
        expiration_date     date
        quantity_bought     integer         not null
        current_quantity    integer
        username            varchar(32)   not null
            constraint username
                references chefs
                on update cascade on delete cascade
        Item_id             integer       not null
            Constraint item_id
                references item
                On update cascade on delete cascade
    );
4.  create table recipe
    (
        recipe_id           integer       not null,
        recipe_name         varchar(64)   not null,
        description         text,
        servings            integer,
        cook_time           integer,
        date_made           date          not null,
        time_made           time          not null,
        difficulty      varchar(32) default ‘Medium’ :: character varying    not null,
        rating              integer,
        steps               text          not null
    );
```

Samples of SQL statements used to populate tables:

1. Populating the made_of table:

   ```
   INSERT INTO made_of

   VALUES

   (100000, 137739, 7),

   (100001, 137739, 4),

   (100002, 137739, 1),

   ...
   ```

2. Populating the item table:

   ```
   INSERT INTO item

   VALUES

   (100000, 'winter squash'),

   (100001, 'mexican seasoning'),

   (100002, 'mixed spice'),

   …
   ```

3. Populating the recipe_category table:

   ```
   INSERT INTO recipe_category

   VALUES

   ('60-minutes-or-less'),

   ('time-to-make'),

   ('course'),

   …
   ```

Samples of queries for obtaining data:

1. SELECT * FROM item

Item_id	Item_name
100000	Winter squash
100001	Mexican seasoning
100002	Mixed spice
100003	Honey
100004	Butter

2. SELECT * FROM item_aisle

Aisle_id	Aisle_name
100000	A-C
100001	D-F
100002	G-I
100003	J-L
100004	M-O

3. SELECT username, password, creation_date, creation_time, last_access_date,

   last_access_time FROM chefs

Username	Password	Creation date	Creation time	Last_access date	Last_access time
Jennifer	princess	1955-02-04	12:20:00	1955-02-04	12:20:00
Nicholas	123123	1999-11-24	01:05:00	2000-01-10	11:50:00
Alex	abcabc	2000-03-16	12:24:00	2000-03-17	02:32:00
Benson	321321	2000-09-14	10:20:00	2001-02-21	06:45:00
Superman	9145	2015-02-04	12:20:00	2020-02-04	12:20:00

The description of how the data was loaded into the database:

Instead of plugging in the data individually into the database, we decided to use a program to generate data that we will be importing into the database. We built a program using Python which reads into a file and splits all the data into a dictionary. This eases up on identifying which attributes are we filling up. For example, for the chef's table, the database stores Username, Password, Creation date, Creation time, Last access date, and Last access time. We decided to use "rockyou.txt" which is a text file filled with passwords. Since we do not restrict input for username and password for Chefs, we split the passwords from "rockyou.txt" and use one split for username and the other for password. As for the date and time, we used Python built-in function (random) to create a randomized date and time for Creation and Last Access. As for the Recipe and Item data, we used a similar approach and read in the data from the given link in the Recipe Domain write-up.

Phase 4 Update:

To perform our data analysis, we used Excel to gather information into a spreadsheet where we could scan for patterns and figure out some common attributes about our users, such as the fact that a majority of recipes created and used by them are in the hard category compared to recipes that fall between the easy and medium categories, which means our users enjoy challenging themselves while cooking. We also used SQL statements to pull helpful data out of the database such as "SELECT COUNT(item_id), recipe_id FROM made_of GROUP BY recipe_id ORDER BY COUNT(item_id)" which gave us the number of ingredients used in each one of our recipes.

When we started to create our application, we noticed that even though everything was being shown, the page was taking a decent amount of time to load and render everything while also making the page laggy. To improve this performance, we decided to display this information on separate pages and have buttons that would bring you to these pages. Another place that can be improved is with the back-end classes. Currently, each class has to individually make a connection to the database before doing any updating/querying. This can be improved by having a parent class that already has a connection established then have each child class make the changes required.

Appendix of SQL Statements:

Log in:

- SELECT username, password FROM chefs WHERE username = ‘username’

Register:

- INSERT INTO chefs(username, password, creation_date, creation_time, last_access_date, last_access_time) VALUES (username, password, dateAndTime, dateAndTime)

Search Recipes by Name:

- SELECT * FROM recipe WHERE recipe_name LIKE '%search%'

Search Recipes by Category:

- SELECT * FROM recipe WHERE recipe_id IN (SELECT recipe_id FROM part_of WHERE category_name=’search’)

Search Recipe by Ingredients:

- SELECT * FROM recipe WHERE recipe_id NOT IN (SELECT r.recipe_id FROM made_of AS r INNER JOIN (SELECT * FROM made_of EXCEPT (SELECT * FROM made_of WHERE item_id IN (SELECT item_id FROM owns WHERE username=’username’))) AS s ON (r.recipe_id = s.recipe_id)) ORDER BY rating DESC

Search Recipes that the User Instantiated:

- SELECT * FROM recipe WHERE recipe_id IN (SELECT recipe_id FROM creates WHERE username=’search’)

Make Category:

- INSERT INTO recipe_category(category_name) VALUES (‘categoryName’)

Make Recipe:

- INSERT INTO recipe(recipe_id, recipe_name, description, servings, cook_time, date_made, time_made, difficulty, rating, steps) VALUES (‘recipeID’, ‘recipeName’, ‘description’, ‘servings’, ‘cookTime’, ‘dateAndTime’, ‘difficulty’, ‘0’, ‘steps’)

Cook Recipe:

- SELECT item_id, quantity FROM made_of WHERE recipe_id=’recipeID’
- SELECT current_quantity FROM owns WHERE username=’username’ AND item_id=’itemID’
- UPDATE owns SET current_quantity= ((SELECT current_quantity FROM owns WHERE username= ‘username’ AND item_id=’item_id’) - item_quantity) WHERE username= ‘username’ AND item_id= ‘item_id’
- SELECT * FROM has_cooked WHERE username= ‘username’ AND recipe_id= ‘recipeID’

- INSERT INTO has_cooked(username, recipe_id) VALUES (‘username’ , ‘ recipeID’)

Delete Recipe:

- DELETE FROM recipe WHERE recipe_id= recipeID

Add Recipe to Category:

- INSERT INTO part_of(recipe_id, category_name) VALUES (‘recipeID’, ‘categoryName’)

Edit Recipe Name:

- UPDATE recipe SET recipe_name = ‘recipeName’ WHERE recipe_id= recipeID

Get Most Recent Recipes:

- SELECT * FROM recipe ORDER BY date_made DESC, time_name DESC LIMIT ‘topNRecent’

Get Highest Rated:

- SELECT * FROM recipe ORDER BY rating DESC LIMIT ‘topNRated’

Add Recipe Rating:

- UPDATE recipe SET rating= ‘newRating’ WHERE recipe_id=’recipe_id’
- UPDATE recipe SET number_of_ratings= (numRating + 1) WHERE recipe_id= ‘recipe_id’

Analytics:

- SELECT COUNT(item_id), recipe_id FROM made_of GROUP BY recipe_id ORDER BY COUNT(item_id)
- SELECT recipe_name FROM recipe WHERE recipe_id IN(Select recipe_id FROM has_cooked)