The Final Project of Databases: Food Recipe Management System

Created by: Lawunn Khaing, Jesse Sillman, Huy Tran, Tran Truong (ITMI22SP)

Introduction

This report examines the application of SQL in structuring relational databases via PostgreSQL, complemented by a Python-based graphical user interface (GUI) to facilitate user interaction with the database. The application's purpiose is to catalog recipes along with their necessary ingredients, cooking hardware, and categories. This system not only allows for the storage and retrieval of recipe data but also supports features like recipe updating, deletion, and advanced searching capabilities based on ingredients and categories.

Database Schema Overview

The database for the Recipe Management System consists of two main tables: `recipes` and `ingredients`. These tables were created store and manage recipes and their ingredients effectively.

Recipes Table

The `recipes` table serves as the core of the Recipe Management System. The following SQL Query was used to create the table:

```
CREATE TABLE recipes (
    id SERIAL PRIMARY KEY,
    title TEXT NOT NULL,
    cooking_time TEXT NOT NULL,
    ingredients TEXT NOT NULL,
    instructions TEXT NOT NULL,
    cooking_hardware TEXT NOT NULL,
    category TEXT NOT NULL
);
```

As we can see, the table holds the following columns:

- `id`: A unique identifier for each recipe, generated by a sequence for auto-incrementation.
- `title`: The name of the recipe.
- `cooking_time`: The time required to prepare the recipe.
- `ingredients`: A text representation of the ingredients used in the recipe.
- `instructions`: A text based guidance on how to prepare the recipe.
- `cooking_hardware`: Any equipment needed to prepare the recipe.
- `category`: The classification of the recipe (e.g. desssert).

Ingredients Table

The `ingredients` table is linked to the `recipes` table through a foreign key relationship. The following SQL Query was used to create the table:

```
CREATE TABLE ingredients (
    id SERIAL PRIMARY KEY,
    name TEXT NOT NULL,
    allergens TEXT,
    recipe_id INTEGER REFERENCES recipes(id);
);
```

As we can see, the table holds the following columns:

- `id`: A unique identifier for each ingredient, generated by a sequence for auto-incrementation.
- `name`: The name of the ingredient.
- `allergens`: Any allergens that the ingredient may contain.
- `recipe_id`: A reference to the associated recipe in the recipes table.

This table has its own primary key, `id`, and a foreign key. `recipes_id`, which establishes a many-to-one relationship with the `recipes` table since each recipe can have multiple ingredients.

ER-Diagram

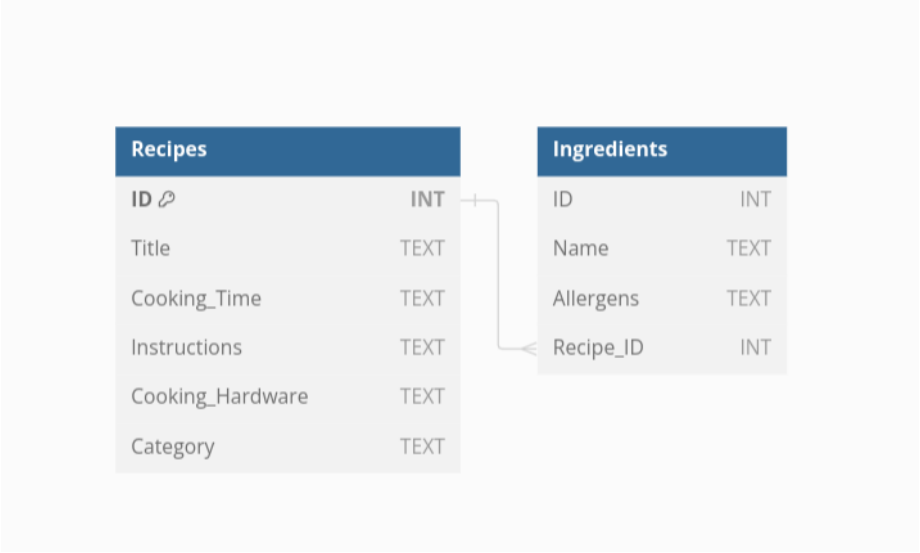

Figure 1: ER Diagram

SQL Commands and Queries with Python

This part covers the SQL commands and queries used within the Pyton code to interact with the PostgreSQL database and manage data within the Recipe Manage System GUI.

Searching Recipes by Ingredient

The following query retrieves the titles of recipes from `recipes` table where ingredients match a specified ingredient:

```
SELECT r.title
FROM recipes r
INNER JOIN i ON r.id=i.recipe_id
WHERE i.name = %s;
```

Here's the breakdown of the query:

- `SELECT r.title`: Selects the `title` column from the `recipes` table.
- `FROM recipes r`: This specifies that the data is being retrieved from the `recipes` table and assigns it an alias `r`.
- `INNER JOIN ingredients i ON r.id = i.recipe.id`: This clause joins the `recipes` table (`r`) with the `ingredients` (`i`) table based on the condition that the `id` column matches the `recipe_id` column in `ingredients`.
- `WHERE i.name = %s`: This condition filters the joined result set to only include rows where the `name` column in the `ingredients` table matches the specified ingredient name (represented by %s).

Note: In Python programming, `%s` is a placeholder used in SQL queries to represent a value that will be provided later due to prevent SQL injection attacks and to make the code more readable and maintainable.

Searching Recipes by Category

To search recipes based on their category, we utilized a SQL query similar to the one used for ingredient-based searching, as following:

```
SELECT r.title
FROM recipes r
WHERE r.category = %s;
```

This query retrieves the titles from the `recipes` table where the category matches a specified category value.

Refresing Recipes

The following method refreshes the recipe listbox by clearing its contents and fetching all recipe titles from the database:

```
SELECT title FROM recipes
```

Add Recipe

The following SQL query inserts a new recipe into the `recipes` table with the provided details and returns the ID of the inserted recipe:

```
INSERT INTO recipes (title, cooking_time, ingredients, instructions, cooking_hardware, categ
VALUES (%s, %s, %s, %s, %s, %s) RETURNING id;
```

Here's the breakdown of the query:

- `INSERT INTO recipes`: Specifies that we are inserting data into the `recipes` table.
- `(title, cooking_time, ingredients, instructions, cooking_hardware, category)`: Lists the columns into which we are inserting data.
- `VALUES (%s, %s, %s, %s, %s, %s)`: Defines that we are providing values for each of the columns listed above.
- `RETURNING id`: This clause is used to return the ID of the newly inserted row.

Delete Recipe

To delete a recipe, we typically use the DELETE statement:

```
DELETE FROM recipes WHERE title = %s;
```

Here's the breakdown of the query:

- `DELETE FROM recipes`: Specifies that we are deleting data from the `recipes` table.
- `WHERE title = %s`: Inserts a condition that filters the rows to be deleted, targeting the recipe with a specific title.

Update Recipe

To update a recipe, we typically use the UPDATE statement:

```
UPDATE recipes
SET title = %s,
    cooking_time = %s,
    ingredients = %s,
    instructions = %s,
    cooking_hardware = %s,
    category = %s
WHERE title = %s;
```

Here's the breakdown of the query:

- `UPDATE recipes`: Specifies the `recipes` table to be updated.

- `SET`: Specifies the columns the user wants to update.
- `WHERE title = %s`: Specifies the condition for which rows should be updated. In this case, it updates the rows where the `title` matches the specified value.

Add Allergen to an Ingredient

To add an allergen to an existing ingredient in the `ingredients` table, we can use the UPDATE statement similarly to update a recipe:

```
UPDATE ingredients
SET allergens = %s
WHERE name = %s;
```

Displaying Allergens in a Warning

To display allergens in a warning message, we can use the following SQL query:

```
SELECT allergens FROM ingredients WHERE name = %s;
```

Here's the breakdown of the query:

- `SELECT allergens`: Selects the `allergens` column from the `ingredients` table.
- `FROM ingredients`: Specifies the `ingredients` table from which to retrieve data.
- `WHERE name = %s`: Specifies the condition for which rows to select. In this case, it selects the row where the `name` of the igredeient matches the specified value.

For each ingredient in the recipe, the application checks if there are any associated allergens. If an ingredient is found to have allergens, the application appends a warning to the recipe details.

This document was created from the `docs/REPORT.md` from this github repository.