Food Recipe Management System

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.

Table of Contents

Database Schema Overview

In designing the database schema for the Food Recipe Management System, we aimed for normalization to minimize redundancy and ensure data integrity. The key entities identified were `recipes`, `ingredients`, `cooking hardware`, and `categories`.

Design Choices

- **Normalization**: By separating entities into their own tables and using junction tables for many-to-many relationships, we ensure that the database is normalized, reducing redundancy and minimizing the risk of data inconsistencies.
- **Flexibility**: The chosen design allows for flexibility in managing recipes, ingredients, cooking hardware, and categories. New recipes can be easily added, and existing recipes can be updated or deleted without affecting other parts of the system.
- **Data Integrity**: By enforcing foreign key constraints and using junction tables to represent complex relationships, we maintain data integrity within the database. This ensures that only valid and consistent data is stored.
- **Scalability**: The modular design of the database schema allows for scalability as the system grows. Additional features and entities can be added without major modifications to the existing structure.
- **Ease of Maintenance**: Separating entities into their own tables and using meaningful foreign keys improves the maintainability of the database. It becomes easier to troubleshoot issues, update data, and add new functionality.

Creation of Tables

The following SQL Query was used to create the table:

```sql
CREATE TABLE categories (
    id SERIAL PRIMARY KEY,
    name TEXT NOT NULL
);

CREATE TABLE recipes (
    id SERIAL PRIMARY KEY,
    title TEXT NOT NULL,
    cooking_time TEXT NOT NULL,
    instructions TEXT NOT NULL,
    category_id INTEGER NOT NULL REFERENCES categories(id)
);
```

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

CREATE TABLE cooking_hardware (
    id SERIAL PRIMARY KEY,
    name TEXT NOT NULL
);

CREATE TABLE recipe_ingredients (
    recipe_id INTEGER NOT NULL REFERENCES recipes(id),
    ingredient_id INTEGER NOT NULL REFERENCES ingredients(id),
    PRIMARY KEY (recipe_id, ingredient_id)
);

CREATE TABLE recipe_hardware (
    recipe_id INTEGER NOT NULL REFERENCES recipes(id),
    hardware_id INTEGER NOT NULL REFERENCES cooking_hardware(id),
    PRIMARY KEY (recipe_id, hardware_id)
);
```

Explanation of Columns

Categories Table

- `id`: An auto-incrementing integer serving as the primary key for the table, uniquely identifying each category.
- `name`: A text field storing the name of the category. This field is marked as `NOT NULL` to ensure that every category has a name associated with it.

Recipes Table

- `id`: Similar to the categories table, this is an auto-incrementing integer serving as the primary key for the table, uniquely identifying each recipe.
- `title`: A text field storing the title or name of the recipe. This field is marked as `NOT NULL` to ensure that every recipe has a title.
- `cooking_time`: A text field storing the cooking time required for the recipe.
- `instructions`: A text field storing the instructions or steps to prepare the recipe.

- `category_id`: An integer field serving as a foreign key referencing the `id` column of the categories table. This establishes a relationship between recipes and categories, indicating the category to which each recipe belongs.

Ingredients Table

- `id`: An auto-incrementing integer serving as the primary key for the table, uniquely identifying each ingredient.
- `name`: A text field storing the name of the ingredient. This field is marked as `NOT NULL` to ensure that every ingredient has a name.
- `allergens`: A text field storing any allergens associated with the ingredient. This field can contain information about allergens to help users with dietary restrictions or allergies.

Cooking Hardware Table

- `id`: An auto-incrementing integer serving as the primary key for the table, uniquely identifying each piece of cooking hardware.
- `name`: A text field storing the name of the cooking hardware (e.g., oven, stove). This field is marked as `NOT NULL` to ensure that every piece of cooking hardware has a name.

Recipe Ingredients Table

- `recipe_id`: An integer field serving as a foreign key referencing the `id` column of the recipes table. This establishes a relationship between recipes and ingredients, indicating which ingredients are used in each recipe.
- `ingredient_id`: An integer field serving as a foreign key referencing the `id` column of the ingredients table. This establishes a relationship between recipes and ingredients, indicating which ingredients are used in each recipe.
- `PRIMARY KEY (recipe_id, ingredient_id)`: This constraint ensures that each combination of `recipe_id` and `ingredient_id` is unique, preventing duplicate entries and maintaining data integrity.

Recipe Hardware Table

- `recipe_id`: An integer field serving as a foreign key referencing the `id` column of the recipes table. This establishes a relationship between recipes and cooking hardware, indicating which cooking hardware is used in each recipe.
- `hardware_id`: An integer field serving as a foreign key referencing the `id` column of the cooking hardware table. This establishes a relationship between recipes and cooking hardware, indicating which cooking hardware is used in each recipe.
- `PRIMARY KEY (recipe_id, hardware_id)`: This constraint ensures that each combination of `recipe_id` and `hardware_id` is unique, preventing duplicate entries and maintaining data

integrity.

Inserting Data

Here is a sample code of how we can insert data into tables.

```
-- Insert categories
INSERT INTO categories (name) VALUES ('Desserts'), ('Pizza'), ('Pasta');

-- Insert ingredients
INSERT INTO ingredients (name, allergens) VALUES ('Flour', NULL), ('Sugar', NULL), ('Eggs', 'G

-- Insert cooking hardware
INSERT INTO cooking_hardware (name) VALUES ('Oven'), ('Stove'), ('Mixer');

-- Insert recipes
INSERT INTO recipes (title, cooking_time, instructions, category_id)
VALUES ('Chocolate Cake', '60', 'Bake at 350°F for 30 minutes', 1),
       ('Margherita Pizza', '20', 'Bake at 400°F for 15 minutes', 2),
       ('Spaghetti Carbonara', '30', 'Boil pasta, fry bacon, mix with egg sauce', 3);

-- Insert recipe-ingredient relationships
INSERT INTO recipe_ingredients (recipe_id, ingredient_id) VALUES (1, 1), (1, 2), (1, 3);

-- Insert recipe-hardware relationships
INSERT INTO recipe_hardware (recipe_id, hardware_id) VALUES (1, 1), (2, 1), (3, 2);
```

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.

Search by Recipes

The following query retrieves the title of the recipes from `recipes` table.

```
SELECT title FROM recipes WHERE title = %s
```

Searching Recipes by Ingredient

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

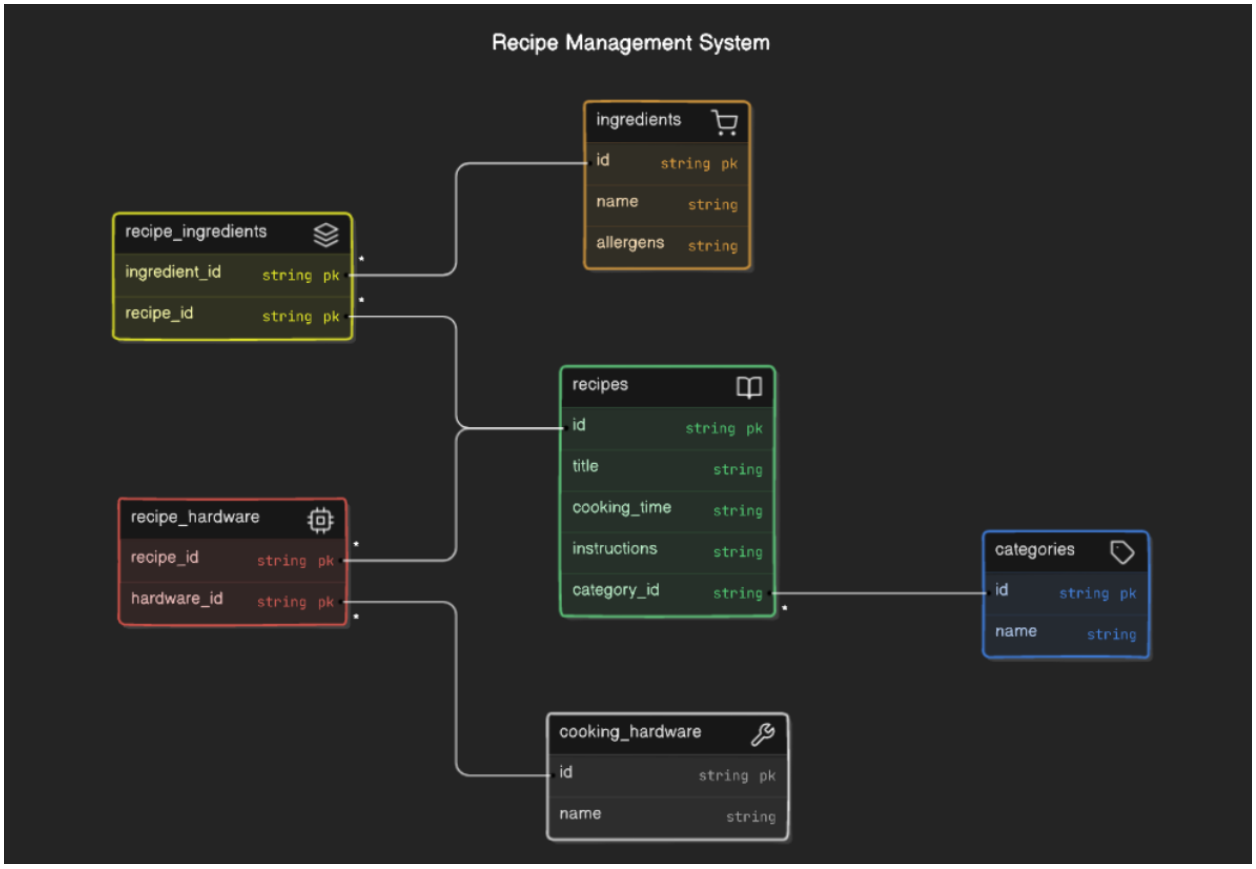

Figure 1: ER diagram

```sql
SELECT DISTINCT r.title
FROM recipes r
INNER JOIN recipe_ingredients ri ON r.id = ri.recipe_id
INNER JOIN ingredients i ON ri.ingredient_id = i.id
WHERE i.name = %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:

```sql
SELECT DISTINCT r.title
FROM recipes r
INNER JOIN categories c ON r.category_id = c.id
WHERE c.name = %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:

```sql
SELECT title FROM recipes
```

Add Allergen to an Ingredient

To add an allergen to an existing ingredient in the `ingredients` table, first we need to fetch data from ingredients

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

After that, if the ingredient exists we can add the allergens: Here is the queries how we add:

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

SQL queries used in postgres database

Delete Recipe

Deleting Records in Junction Tables in Many-to-Many Relationships

In a many-to-many relationship, such as the one between recipes and ingredients, a junction table is used to associate records from both tables. This junction table typically contains foreign keys that reference the primary keys of the tables involved in the relationship.

In the following schema:

```
recipes       recipe_hardware
--------      -------------------
id (PK)  <--- recipe_id (FK)
title           hardware_id (FK)
            |
            |
            |
```

To delete a recipe, we need to first delete the accociated records from the `recipe_hardware` table that reference the recipe we want to delete. Once those are deleted then we can delete the recipe from the `recipes` table.

Here is the example:

```sql
DELETE FROM recipe_hardware WHERE recipe_id = 3;
DELETE FROM recipes WHERE id = 3;
```

Update Recipe

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

```sql
UPDATE recipes
SET cooking_time = '20', instructions = 'Mix lettuce, croutons, dressing, cheese, and grilled
WHERE title = 'Caesar Salad';
```

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`: Specifies the condition for which rows should be updated. In this case, it updates the rows where the `title` matches the specified value.

Conclusion

In conclusion, this project establishes a robust database schema for managing recipes, ingredients, cooking hardware, and categories. Through the use of junction tables and foreign key constraints, it ensures data integrity and facilitates efficient retrieval of recipe-related information. The provided SQL queries enable seamless insertion, updating, and deletion of data, empowering users to maintain and expand the recipe database with ease.

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