MEAL MASTER

A Simple Meal Scheduling Application

Using :

Developed By:

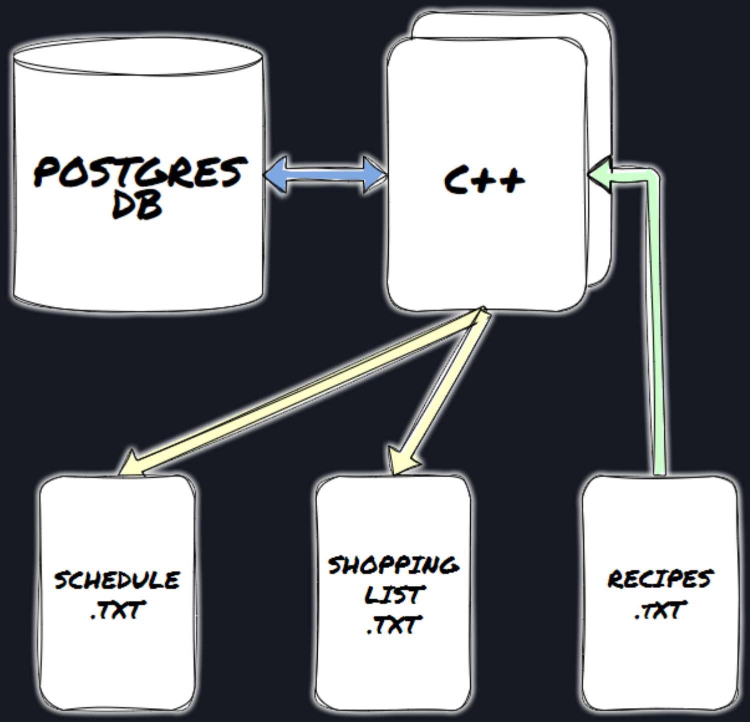
Alfredo Renteria

Table of Contents

ABOUT

Story:

GENERAL DESIGN



C++

Fills PostgreSQL database from

- `'Recipes.txt'`
    - File containing recipes

Generates from database

- `'Schedule.txt'`
    - Weekly dinner plan
- `'ShoppingList.txt'`
    - Shopping list for plan

PostgreSQL

Database for the recipes, ingredients, mealTypes, and history

Files

Read File

- `'Recipes.txt'`

Write File

- `'Schedule.txt'`
- `'ShoppingList.txt'`

SCRIPT FILE

General:

The Bash script file, ‘`ProduceSchcedule.sh`’ will manage the C++ client and PostgreSQL database. It will be responsible for running .sql files and commands prior to running the C++ client.

Ensure that the search_path is set to the correct schema

```
SET SEARCH_PATH to MealMaster;
```

Run the .sql files

```
$ psql -U <user> -d <database> -f <filename>.sql
```

Make the Meal Master C++ application

```
$ make
```

Run Meal Master to generate a schedule file

```
$ ./mealMaster
```

Clean

```
$ make clean
```

MEALMASTER CLASS

A class that is responsible for managing connections with the *PostgreSQL* database. These connections involve reading recipes from ‘`Recipes.txt`‘ to build the database in *PostgreSQL*. As well as querying the database to generate the meal schedule and shopping list for ‘`Schedule.txt`’ and ‘`ShoppingList.txt`’ respectively.

```
class MealMaster {
public:
    //Constructor
    MealMaster() {
        /* Make connection with db */
        /* Query stored function to insert mealTypes into db */
    }

    //Destructor
    ~MealMaster() {
        /* Close connection with db */
    }

    //Builds database from Recipes.txt
    bool buildDatabase() {
        /* Open Recipes.txt file */
        /* Error handle */
        /* Read until EOF */
        /* Insert recipes into db */
        /* Close file */
    }

    //Makes weekly meal schedule in Schedule.txt
    bool buildSchedule() {
        /* Create Schedule.txt file */
        /* Error handle */
        /* Based on meal type variance for schedule, query db for recipes */
        /* Write week schedule */
        /* Insert schedule into db */
        /* Close file */
    }
    //Makes list for recent meal schedule in ShoppingList.txt
    bool buildShoppingList() {
        /* Create ShoppingList.txt file */
        /* Error handle */
        /* Query for recent meal schedule */
        /* Query for compilations of ingredients */
        /* Write shopping list */
        /* Close file */
    }

private:
    //Retrieves mealTypeID from mealTypes table
    //NOTE: All ID getters for db operate similarly
    int getMealTypeID(std::string&) {
        /* Build query with input string */
        /* Execute query with db connection */
        /* Error handle */
        /* Convert result to integer */
        /* Clear pointer */
        /* Return result */
    }

    //Retrieves ingredientID from Ingredients table
    int getIngredientID(std::string&);
    //Retrieves recipeID from Recipes table
    int getRecipeID(std::string&);

    //Adds ingredient to Ingredients table
    //NOTE: All db insertions operate similarly
    void addIngredient(std::string&) {
        /* Build query with input string */
        /* Execute query with db connection */
        /* Error handle */
        /* Clear pointer */
    }

    //Adds recipe to Recipes table
    void addRecipe(std::string&, int, std::string&);
    //Adds qty and unit info for ingredient in a recipe to RecipeIngredients table
    void addRecipeIngredients(int, int, float, std::string&);
    //Generates SQL queries
    std::string queryGen(std::string, std::string) {
        /* Build query with stored function string */
        /* If input string, add to query */
        /* Return query */
    }

    //Connection handle to the PostgreSQL database
    PGconn* conn;

    //Struct for holding recipe information
    struct Recipe {
        int recipeID;
        int mealTypeID;
        std::string name;
        //Holds ingredientID, qty, and unit
        std::vector<std::pair<int, std::pair<float, std::string>>> ingredients;
    };
};
```

MAIN

Main driver code. Utilizes the Recipe class

DATABASE DESIGN

MealTypes		
PK	mealTypeID	SERIAL
UQ	mealTypeName	VARCHAR(2) NOT NULL

Ingredients		
PK	ingredientID	SERIAL
	ingredientName	VARCHAR(100) NOT NULL

Recipes		
PK	recipeID	SERIAL
	recipeName	VARCHAR(100) NOT NULL
FK	mealTypeID	INT REF
	instructions	TEXT NOT NULL

RecipeIngredients		
PK	recipeIngredientsID	SERIAL
FK	recipeID	INT REF
FK	ingredientID	INT REF
	totalQty	NUMERIC NOT NULL
	qtyType	VARCHAR(20) NOT NULL

History		
PK	historyID	SERIAL
	monID	INT REF Recipes(recipeID)
	tueID	INT REF Recipes(recipeID)
	wedID	INT REF Recipes(recipeID)
	thuID	INT REF Recipes(recipeID)
	friID	INT REF Recipes(recipeID)

STORED FUNCTIONS

Database stored functions will be included in ‘`storedFunctions.pgsql`’. The stored functions will help prepare the database, and also facilitate access to the C++ client interface.

fillMealTypes()	Input: none Return: none
	Fills MealTypes table with all possible mealTypesNames: 'G' | 'P' | 'R' | 'S'
getMealTypeID(...)	Input: mealTypeName VARCHAR Return: mealTypeID INT
	Retrieves mealTypeID from MealTypes table given mealTypeName
addIngredient(...)	Input: ingredientName VARCHAR Return: none
	Inserts ingredient into the Ingredients table given ingredientName
getIngredientID(...)	Input: ingredientName VARCHAR Return: ingredientID INT
	Retrieves ingredientID from Ingredients table given ingredientName
addRecipe(...)	Input: recipeName VARCHAR, mealTypeID INT, instructions TEXT Return: none
	Inserts recipe into the Recipes table given recipeName, mealtTypeID, and instructions
getRecipeID(...)	Input: recipeName VARCHAR Return: recipeID INT
	Retrieves recipeID from Recipes table given recipeName
addRecipeIngredients(...)	Input: recipeID INT, ingredientID INT, totalQty NUMERIC, qtyType VARCHAR Return: none
	Inserts qty and qtyType info for ingredient in a recipe

getRecipesByMealType(...)	Input: mealTypeID INT Return: TABLE with recipeID INT, recipeName VARCHAR
	Retrieves recipes from Recipes table that match given mealTypeID
getRecentSchedules()	Input: none Return: TABLE with historyID INT, monID INT, tueID INT, wedID INT, thuID INT, friID INT
	Retrieves last 3 schedules from History table

RECIPES.TXT FILE – INPUT – READ FILE

Formatting for ‘`Recipes.txt`’:

Recipes in the ‘`Recipes.txt`’ file should be formatted as shown here.
When adding multiple recipes to the file, apply a new line in between individual recipes.

See *Samples* section for brief examples of ‘`Recipes.txt`’ formatting.

<RECIPE NAME>
<MEAL TYPE>

- **<OPTIONS>:**
  - **‘G’**: Green
  - **‘P’**: Poultry
  - **‘R’**: Red Meat
  - **‘S’**: Seafood

[INGREDIENTS]
<QTY> <QTY TYPE> <INGREDIENT NAME>
… … …
#
<INSTRUCTIONS>
…

Samples for ‘`Recipes.txt`’:

```
Italian Pasta Salad
G
1 head Broccoli
1 - Cucumber
1 cup Italian Dressing
1 pack Pasta Noodles
#
1. …
2. …
3. …
```

```
Teriyaki Chicken
P
2 lbs Chicken Breast
1.5 cups Teriyaki Sauce
1 cup White Rice
#
1. …
2. …
3. …
4. …
```

```
Tacos de Carne Asada
R
2 lbs Beef Chuck Steak Boneless
0.5 bunch Cilantro
1 bunch Green Onion
1 cup Green Salsa
12 - Tortillas
0.5 - Yellow Onion
#
1. …
```

SCHEDULE.TXT FILE – OUTPUT – WRITE FILE

Formatting for ‘`Schedule.txt`’:

```
Monday     Tuesday    Wednesday  Thursday   Friday
Meal 1     Meal 2     Meal 3     Meal 4     Meal 5
```

SHOPPINGLIST.TXT FILE – OUTPUT – WRITE FILE

Formatting for ‘`ShoppingList.txt`’:

<QTY> <QTY TYPE> <INGREDIENT NAME>

OTHER TOOLS USED IN THIS PROJECT

- Draw.io for diagrams/sketches
- Photoshop for logo/image editing
- DALL-E for generating logo design ideas
- Google Docs for ‘`DESIGN.pdf`’