Meal Master

A Simple Meal Scheduling Application

Using :

Developed By:

Alfredo Renteria

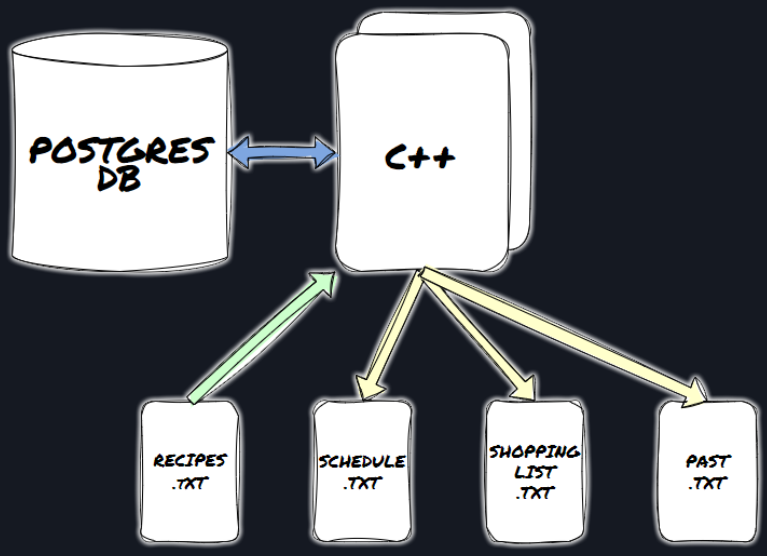
Table of Contents

- ABOUT
  - Story
- General design
  - Overview
- Bash
  - Script file
- C++
  - mealmaster class
  - Main
- PostgreSQL
  - Database design
  - Stored functions
- Files
  - Input - read file
  - Output - write files

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

Reads and updates
- `'Past.txt'`
  - Historical schedules

PostgreSQL

Database for the recipes, ingredients, and mealTypes

Files

Read File
- `'Recipes.txt'`

Write File
- `'Schedule.txt'`
- `'ShoppingList.txt'`

Read/Write File
- `'Past.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. The script may involve a simple test of the ‘Past.txt’, to ensure functionality.

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
```

Quality control new schedule; must differ

```
$ diff oldPast.txt Past.txt
```

Clean

```
$ make clean
```

MEALMASTER CLASS

General:

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*. Querying the database to generate the meal schedule and shopping list for ‘`Schedule.txt`’ and ‘`ShoppingList.txt`’ respectively. Also for updating the schedule history in ‘`Past.txt`’

```
class MealMaster {
public:
    //Default constructor
    MealMaster();
    //Destructor
    ~MealMaster();
    //Builds the MealMaster database from recipes in Recipes.txt
    bool buildDatabase(const std::string& fileName);
    //Generates a meal schedule to Schedule.txt
    bool produceSchedule(const std::string& fileName);
    //Generates a shopping list for the current meal schedule to ShoppingList.txt
    bool produceShoppingList(const std::string& fileName);

private:
    //Adds current meal schedule to historical schedules data Past.txt
    bool updateHistory(const std::string& fileName);
    //Connection handle to the PostgreSQL database
    PGconn *conn;
};
```

MAIN

General:

Main driver code. Utilizes the Recipe class

Goals:

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) NO NULL

STORED FUNCTIONS

General:

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 ID from Recipes table given recipeName

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>
… **…** **…**
INST
<INSTRUCTIONS>
…

Samples for ‘Recipes.txt’:

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

Teriyaki Chicken
P
2 lbs Chicken Breast
1.5 cups Teriyaki Sauce
1 cup White Rice
INST
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
INST
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>

PAST.TXT FILE – OUTPUT – WRITE FILE

Format for ‘Past.txt’:

<SCHEDULE COUNT>

Meal 1

Meal 2

Meal 3

Meal 4

Meal 5

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’`