LazyGrocer

Michael Montanaro, Colby Hegarty, Sanay Doshi

I. ReadMe

A. Installation

1. Clone and navigate to repo
2. `pip install -r requirements.txt`
3. Set PASS for MySQL in `src/config.py`
4. Load `init.sql` into MySQL through MySQL Workbench

B. Usage

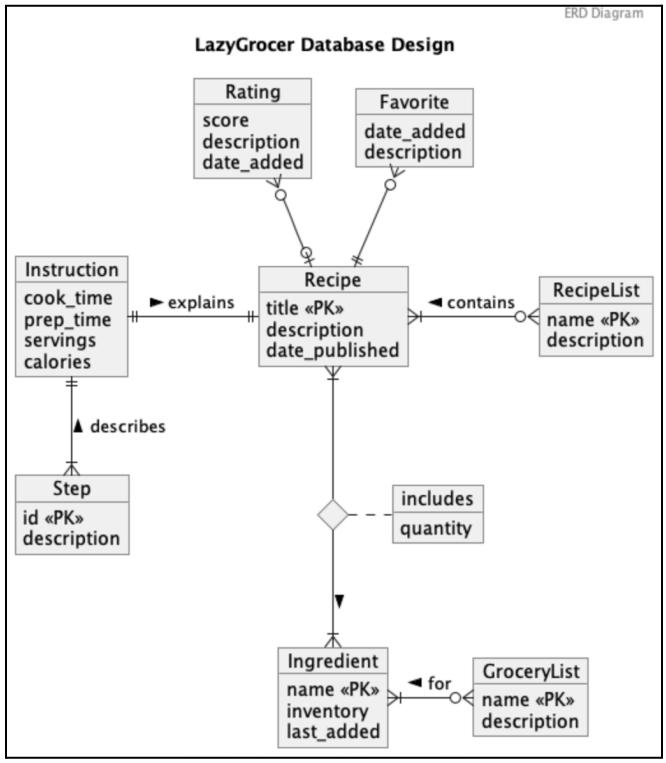
1. While at the project root directory, run `cd src`
2. Run `python main.py`

II. Technical Specifications

- Programming Language
  - *Python*: The project will be developed using Python, leveraging its extensive libraries and compatibility with various systems.
- Libraries and Frameworks
  - *MySQL Connector Python*: Connects the application to a MySQL database to manage data through operations like adding, updating, deleting, and retrieving.
- Database → MySql
  - The backend database will be MySQL, which is perfect with large data volumes. The choice of MySQL stems from our familiarity with it, its compatibility with different operating systems and its robust feature sets, making it ideal for web applications.
- OS → Any
  - The software is designed to operate seamlessly across different computing environments without requiring modification.
- Hardware → Any
  - Since the application is developed using Python, it does not demand high hardware specifications. It can run on general purpose hardware, including older and modern PCs

III. Textual Description of Conceptual Design + UML Diagram

The UML diagram (seen to the right) for the LazyGrocer application's database conceptual design outlines several entities and the relationships between them, which facilitate the application's functionalities as described.



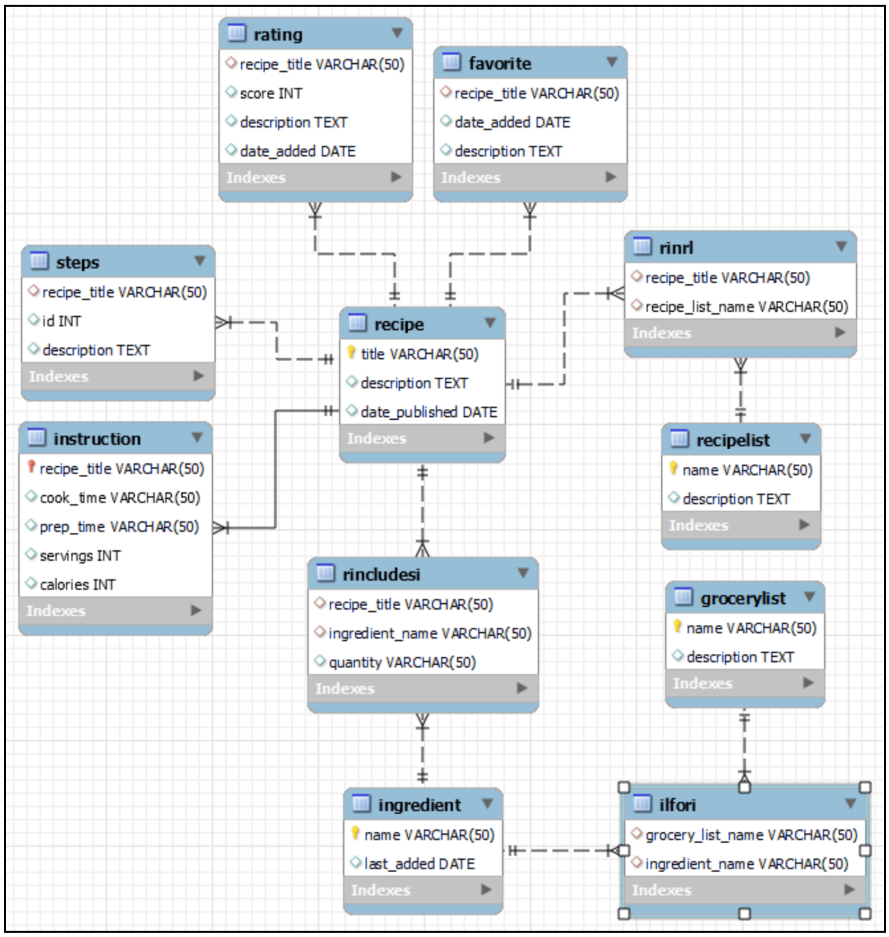
Entity Description

1. **Recipe Entity:** Represents individual recipes with a unique title. It holds a description of the recipe and the date it was published.
2. **Instruction Entity:** Contains detailed preparation instructions for a recipe, including cooking time, preparation time, servings, and calories information.
3. **Step Entity:** Lists the individual steps involved in preparing a recipe, with each step having an ID and a detailed description.
4. **Rating Entity:** Holds rating information for recipes, where each rating includes a score, a description of the rating, and the date it was added.
5. **Favorite Entity:** Marks recipes as favorites, with records that include the date the recipe was favorited and a description.
6. **RecipeList Entity:** Acts as a collection or list of recipes, identified by a name and described by its content.
7. **Ingredient Entity:** Details individual ingredients with a name as the primary key, the current inventory level, and the date the ingredient was last added.
8. **GroceryList Entity:** Compiles lists of ingredients needed for shopping, with a name for the list and a description of its contents.

Relationship Description

1. **Recipe to Instruction Relationship (“explains”):** This one-to-one relationship means that each Recipe has one set of Instructions that 'explains' how to make the dish, including the cooking and prep times, servings, and calories.
2. **Recipe to Step Relationship (“describes”):** The one-to-many relationship shows that a Recipe is 'described' by multiple Steps, where each step provides a portion of the cooking instructions.
3. **Recipe to RecipeList Relationship (“contains”):** This many-to-many relationship allows a Recipe to appear in multiple RecipeLists, and each RecipeList can ‘contain’ various Recipes. It reflects the ability of users to organize recipes into different collections.

IV. Logical Design (Reverse Engineered)



V. Final User Flow of System

Upon opening the LazyGrocer, the user has a variety of options and can interact with the application to complete several tasks. They can choose to view one or all recipes, recipe lists, ingredients, and grocery lists. When viewing a recipe, the instructions and rating associated with the recipe are provided to the user. They can also view the steps associated with a specific recipe or all the recipes associated with a specific recipe list. The user can even view all of the ingredients associated with one or more recipes saved in the application. In addition to being able to view the data stored in the application, the user can add a recipe, recipe list, rating, steps, instructions, ingredients, or grocery list. They can choose to update any of these as well, also including whether or not a recipe is a "favorite". Lastly, the user can delete recipes, recipe lists, ingredients, or grocery lists. All services described above are accessible to the user through a series of prompts, the first of which asks the user to select a CRUD service to be performed on the database. Once the selection is made, the app presents the user with a list of the resulting sub-services related to their desired CRUD operation. The user can then select, using an ID, the corresponding sub-service that goes with their desired action. The sub-service prompts the user for any necessary information and then runs the operation.

VI. No visual adjustments were made to the outputted data. This was due to complications with reading a SQL script (init.sql) from either `pymysql` or `mysql.connector`, which took up quite a bit of our time. The initializing script being read from the Python DBMS was unsuccessful and we had to adjust to running the script from the MySQL Workbench.

```
def delete(self):
print("Select DELETE Service")
for index, service in enumerate(self._delete_services_):
print(f'{index}. {service}')
d_service = input("DELETE ID: ")
_handle_delete_service_(d_service)
def _handle_delete_service_(self, service):
        iService = int(service)
        if iService == 1:
                del_recipe_title = input("Recipe Title: ")
```

```
        deleteRecipeCommand =
DeleteRecipe.DeleteRecipeCommand(del_recipe_title, self.dc)
deleteRecipeCommand.execute()
elif iService == 2:
        list_name = input("Recipe List Name: ")
        deleteRecipeCommand =
DeleteRecipe.DeleteRecipeCommand(list_name, self.dc)
        deleteRecipeCommand.execute()
        elif iService == 3:
        list_name = input("Recipe List Name: ")
        deleteRecipeCommand =
DeleteRecipe.DeleteRecipeCommand(list_name, self.dc)
        deleteRecipeCommand.execute()
elif iService == 4:
        del_recipe_title = input("Recipe Title: ")
        deleteRecipeCommand =
DeleteRecipe.DeleteRecipeCommand(del_recipe_title, self.dc)
deleteRecipeCommand.execute()
```

VII. “Lessons Learned”

A. Database Schema design and development techniques, specifically the use of procedures to simplify the backend python code and reduce the amount of raw SQL.

B. Group meetings should have been held earlier in the semester so that we could break up the project into smaller parts and work on things more independently. The debugging process took up quite a bit more time than was expected as well. This led to reduced attention to the front end of the application.

C. The design approach was reached almost perfectly, we fell a bit short in user-flow. The correction to said issue was through the user-prompt system that can be found in `controller/appController.py`.

VIII. “Future Work”

a. We hope that this project and database will be used to help technically savvy people keep track of their recipe and improve the grocery shopping experience.
b. More support for tracking the ingredients that the user has at their house. Additionally, adding support for a login and multi-user system would be fantastic. The multi-user system would allow for the Rating table to have more of an impact, since it could come from multiple sources.
c. Lastly, if granted even an hour more time, we would have adjusted the code to work similar to the `git` CLI tool. This would allow the user to bypass the first CRUD service selection.