CS564 - Summer 2019

Report for Final Project

Group #2

August 15, 2019

Title of the Project: Healthism or Hedonism

Authors:

Paul Cary – rcary2@wisc.edu

Shunyi Huang – shuang347@wisc.edu

Leixiong Zheng – lzheng69@wisc.edu

Table of Contents:

1. Introduction:
   1. Problem Statement and Importance
   2. Motivation
   3. Application Description
   4. Author Contributions
2. Implementation
   1. System Architecture
   2. Dataset
   3. ER diagram
   4. Relational model
   5. Implementation
   6. Evaluation
3. Conclusion
4. References

I. Introduction

I.1 Problem statement and Importance:

The authors chose to address the problem of discovering new recipes, to facilitate novelty-seeking in individuals who want variety in their meals. The problem of deciding what to eat for any given meal is common enough to have a substantial industry dedicated to solving that problem[1-2], where between 10 and 15% of ‘Top Grossing Food and Drink’ applications each month in the Apple or Google ‘stores’ recommend, track, or store recipes for users. In recommending these recipes, advertisers are trying to tap into the roughly 5% of disposable income that Americans spend on buying food for dining-in[3].

I.2 Motivation:

The authors chose to create a database application in the food and food-recipes domain given its broad applicability (everybody eats), and because tackling a large semi-structured user-generated dataset seemed like an interesting challenge. The approach of the authors is to study one small aspect of that problem by trying to provide recipes for meals, given a number of search criteria, and search criteria combinations.

I.3 Application Description:

The application is written in Java, using JavaFX to provide a GUI interface. The database is implemented using MySQL. The six functional elements of the application that will be referred to in the rest of this document are as follows:

1. Users can sign up with an account that can be used to keep track of the recipes they like and do not like.
2. Users can take one of two ‘paths’ by selecting either “healthism” (low calorie) or “hedonism” (high calorie). Each path has multiple, corresponding, different calorie level presented as search defaults.
3. Users can then search for recipes based on the calories per recipe, tagged categories they want, categories they don’t want, and matches based on Recipe Titles.
4. Recipes can be searched for by partial matches to title, exact matches to category, and by calorie threshold (e.g. searching below a given value). Recipes can be excluded from search by category, or by adding a recipe to a ‘disliked’ list.
5. When users get a recipe the like, they can add that recipe into their list of favorites.
6. Separately, users can look up detailed USDA nutritional information about ingredients in the recipes they find, or any ingredients they are curious about.

I.4 Author Contributions:

**Paul Cary**: Paul Cary contributed to Relational Model design, and wrote the initial schemas. He vetted all datasets, gained dataset expertise, cleaned and tested data characteristics and provided the transformed data in a load-able format. He wrote initial draft of SQL queries, and updated schemas to optimize query performance. He provided very minor design strategies for parsing the database output and implementing the search interface.

**Shunyi Huang and Leixiong Zheng**: Shunyi and Leixiong contributed to the design of the user interface and coding the application. They tested and implemented the database schema. They first drew the GUI control graph on paper and checked if all the scenes can be accomplished, and started to code by using JavaFX. Furthermore, they wrote 20 queries as part of the GUI code to filter all the search conditions based on the user’s requirement. Finally, they decorate the scene of the application and optimize the performance of the whole application.

II. Implementation

II.1 System Architecture

The authors chose to use a MySQL database instance, with Java chosen as the primary development language, and the JavaFX framework for the GUI. All database tests and development of the JavaFX GUI were done on personal laptops. The initial data cleaning, and data organization was done on UW-Madison Linux machines with a combination of shell scripts (checking character sets) and the python pandas library (to check for null values, sort, and join two source tables on selected attributes).

The database schema was designed with a limited number of general entities, and a few relationships between the main entities. The main entities are Users, Recipes, Categories, and Ingredients. The relative sizes of these entity and relationship sets are detailed in the next section.

All tables in the MySQL database use integer keys, giving each record a unique value, and ensuring conformance to the Third Normal Form (3NF). The application was designed such that the preponderance of joins would be between the Recipes entity and the recipe-categories relationship table. After that one join, results would be filtered out based on attribute matching or an attribute range threshold (recipe calories).

II.2 Dataset

There are two datasets which comprise the inputs to our database, one which specifies 20,000+ recipes and which among nearly 700 categories users tagged for those recipes, and one dataset that specifies all of the nutritional information in a set of 8000+ ‘ingredients’.

II.2.a The first dataset, is a set of recipes scraped from epicurious.com, that comes in two files[4]. The first file is a set of recipes, in json format. Each recipe has a title, an average user rating, calories, fat, protein, and sodium values, a full set of directions, and an ingredients list. The second file of the first dataset is a csv where each recipe title is given on a row, and the epicurious.com defined category tags are defined by the columns. If a recipe is tagged with a given category, there is a ‘1’ in the corresponding column for that recipe. If there is no relationship, the column is marked '0'. Since there was no mapping (e.g. a recipe\_ID) between the json and csv files, the first step was to assign a recipe\_ID to the larger dataset (the json set), remove duplicates from the full json dataset, and join that set to the category-tagged dataset on ‘recipe\_title’, ‘rating’, and ‘calories’. The join needs to be on all three attributes, since title and calories did not guarantee a unique recipe in this dataset (it turns out different recipes can have the same title and rating). The python ‘pandas’ library ‘merge’ function was used to accomplish this join as a left join using the json data as the left table. The resulting data contained 16,000+ rows.

II.2.b With the data joined on a recipe\_ID, we were then able to create the recipe\_category relationship set by looping through all recipe-category pairs, and writing existing relationships to a recipe\_category relationship file.

II.2.c The second dataset consists of a large csv with 8000+ rows of USDA ingredients, also pulled from kaggle.com[5] and detailed information regarding the nutritional values of 75 different attributes of each ingredient per 100g. For more details on this dataset visit the USDA Food Composition database at https://ndb.nal.usda.gov, and see the documentation[6].

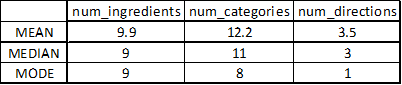


Table 1. Recipes Data - Descriptive Statistics

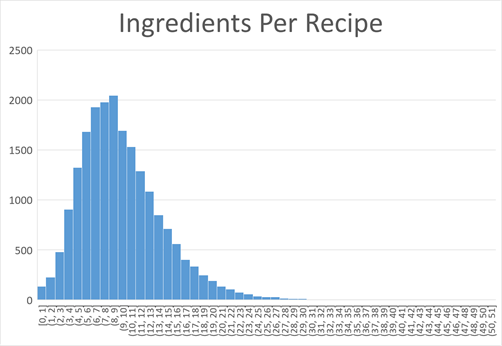


Figure 1. Recipes Data - Histogram of the Number of Ingredients Per Recipe. One can see that the majority of recipes have between 3 and 15 ingredients, with vanishingly few recipes containing 25-50 ingredients.

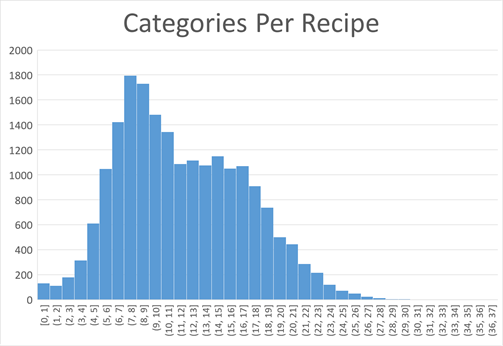


Figure 2. Recipes Data - Histogram of Categories per Recipe

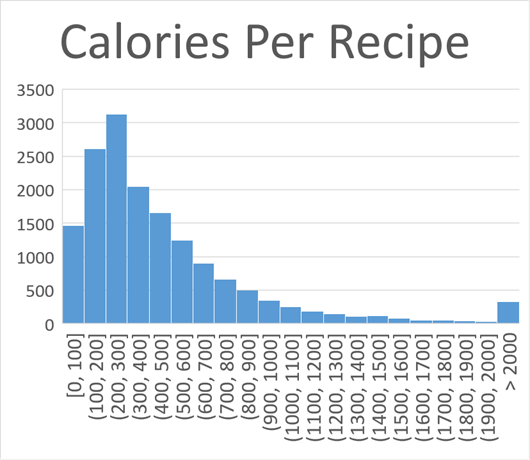


Figure 3. Recipes Data - Histogram of Calories per recipe. The expected number of results can be estimated by this histogram, for any given calorie threshold. One can see that roughly half of the dataset has 400 or fewer calories per recipe.

II.3 ER Diagram

The entity relationship diagram was used to plan out the main entities, which attributes the authors would track, and the relationships between those entities. One unusual feature of our ER diagram is the wholly separate Ingredients entity, which is being provided to users as an interactive reference table, rather than an entity set that can be joined or related to the other entities. The initial plan was to provide a relationship between Recipes and their Ingredient nutritional information, but this proved to be a project which exceeded the scope of this assignment (see Conclusion). See Figure 4 for the full diagram.

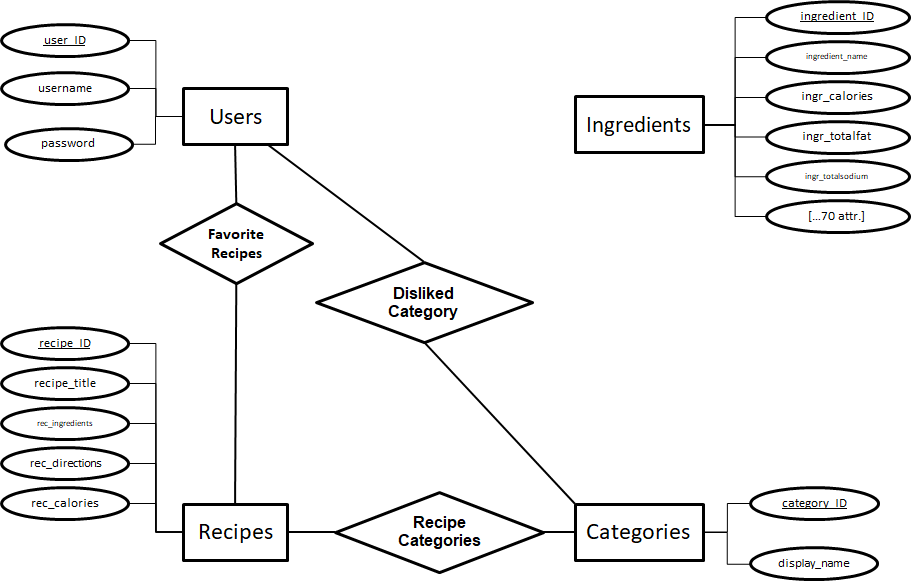


Figure 4. Entity Relationship Diagram. The four main entities (Users, Recipes, Categories, Ingredients) are represented, along with the three relationships between them. Each entity set has a unique ID for every element in that entity set.

II.4 Relational Model

Entities (MySQL data types used):

users(user\_ID: INT, username: VARCHAR, password: VARCHAR)

recipes(recipe\_ID: INT, recipe\_title: VARCHAR, calories: INT, directions: TEXT, ingredients: Text)

categories(category\_ID: INT, display\_name: VARCHAR)

ingredients(ingredient\_ID: INT, name: VARCHAR, serving\_size: VARCHAR, calories: VARCHAR, total\_fat: VARCHAR, plus 72+ additional VARCHAR attributes)

Relationships:

recipe\_category(recipe\_ID: INT, category\_ID: INT)

favorite\_recipes(recipe\_ID: INT, user\_ID: INT)

II.5 Implementation

The first scene of our application is the login/signup page. See Figures 5 and 6.

In our second scene, we have three options for users and they are healthism (the highest default calories in this selection is lower than 700),  hedonism (the highest default calories in this selection is lower than 2000), and the users can choose to get into their favorite list directly. See Figure 7.

After picking either healthism or hedonism, the search interface consists of four main elements. Users can search by (1) calorie threshold, (2) Recipe Title, (3) Included Categories, and (4) Excluded Categories. Users can also navigate to their favorite recipes, the Hedonism search page, or log out using the buttons at the left. See figures 8 and 9.

When searching by calorie threshold, users can type in the specific number of calories they want, and the system default will show them all the recipes for which the calories are lower than their input number. If users know the title, or part of the title of hte recipes they want, they can just type in part of the name and select it from the result set. Filtering by category is separated to two parts, one is the category the users like and the other what they do not like. All of these options can be combined in a single search. Users can search for recipes with 'cookies' in the title, under 300 calories, that are in the 'almond' category, and not in the 'sugar-free' or 'gluten-free' categories.

In the result page, users can still go into their favorite list. They can simply click the recipes they like and click the like button on the upper right corner to add to their favorite list. When users click on any returned recipes (or ingredients), information about that recipe will show up in the window at the right side. See Figure 10.

图片包含 屏幕截图

描述已自动生成

Figure 5. Login/Signup page. This is the first thing the user sees when starting the application.

图片包含 屏幕截图

描述已自动生成

Figure 6. If a user wished to register, the buttons change to reflect that choice, and will insert any user not already in the Users table of the databse.

图片包含 屏幕截图

描述已自动生成

Figure7. Healthism or Hedonism. Users can choose whether they want to see high calorie or low calorie recipes by default by clicking the 'Healthism' or 'Hedonism' buttons respectively.

图片包含 屏幕截图

描述已自动生成

Figure 8. Healthism Search Page.

图片包含 屏幕截图

描述已自动生成

Figure 9. Hedonism Search Page.

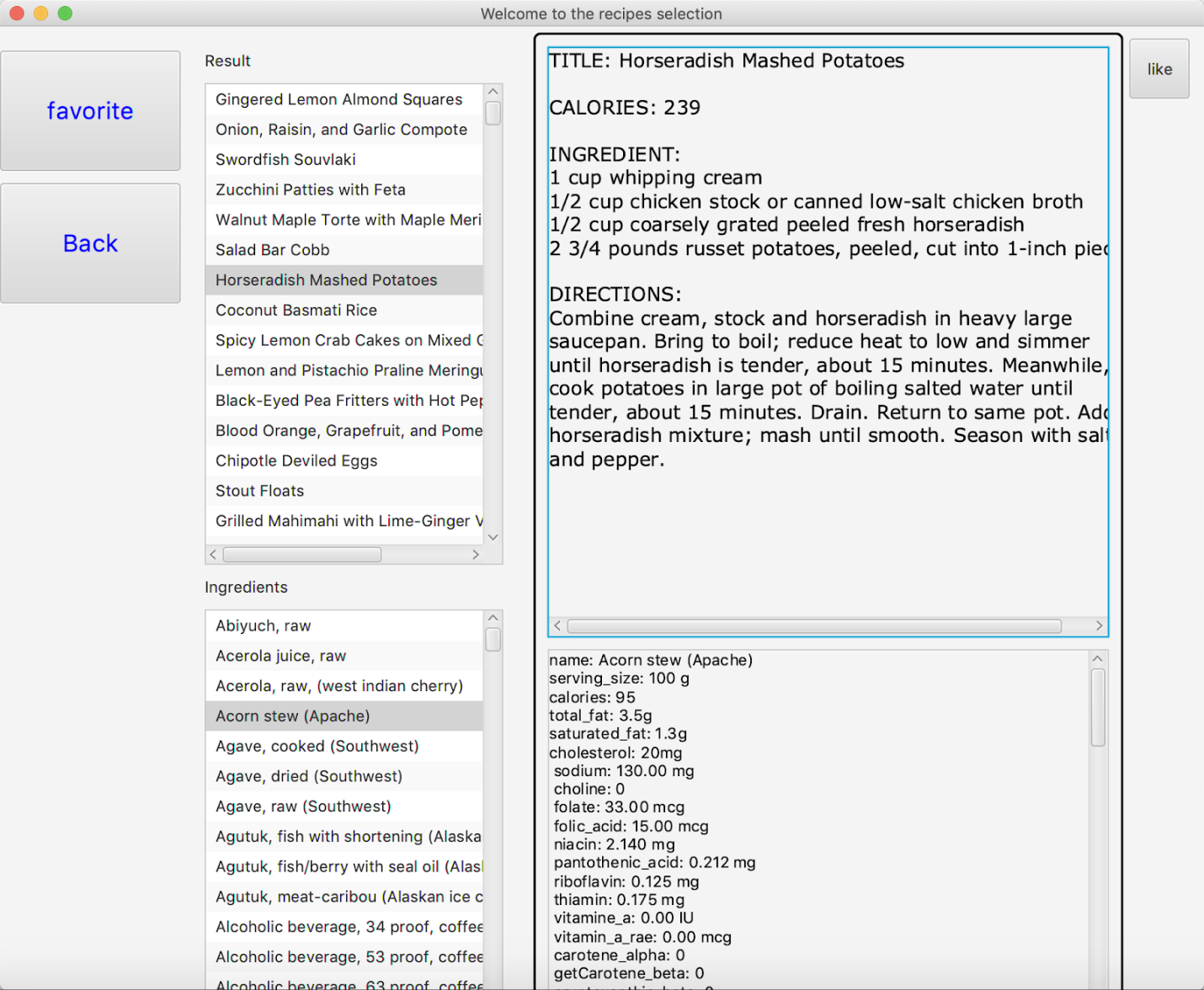


Figure 10. Search Results and Ingredient Lookup.

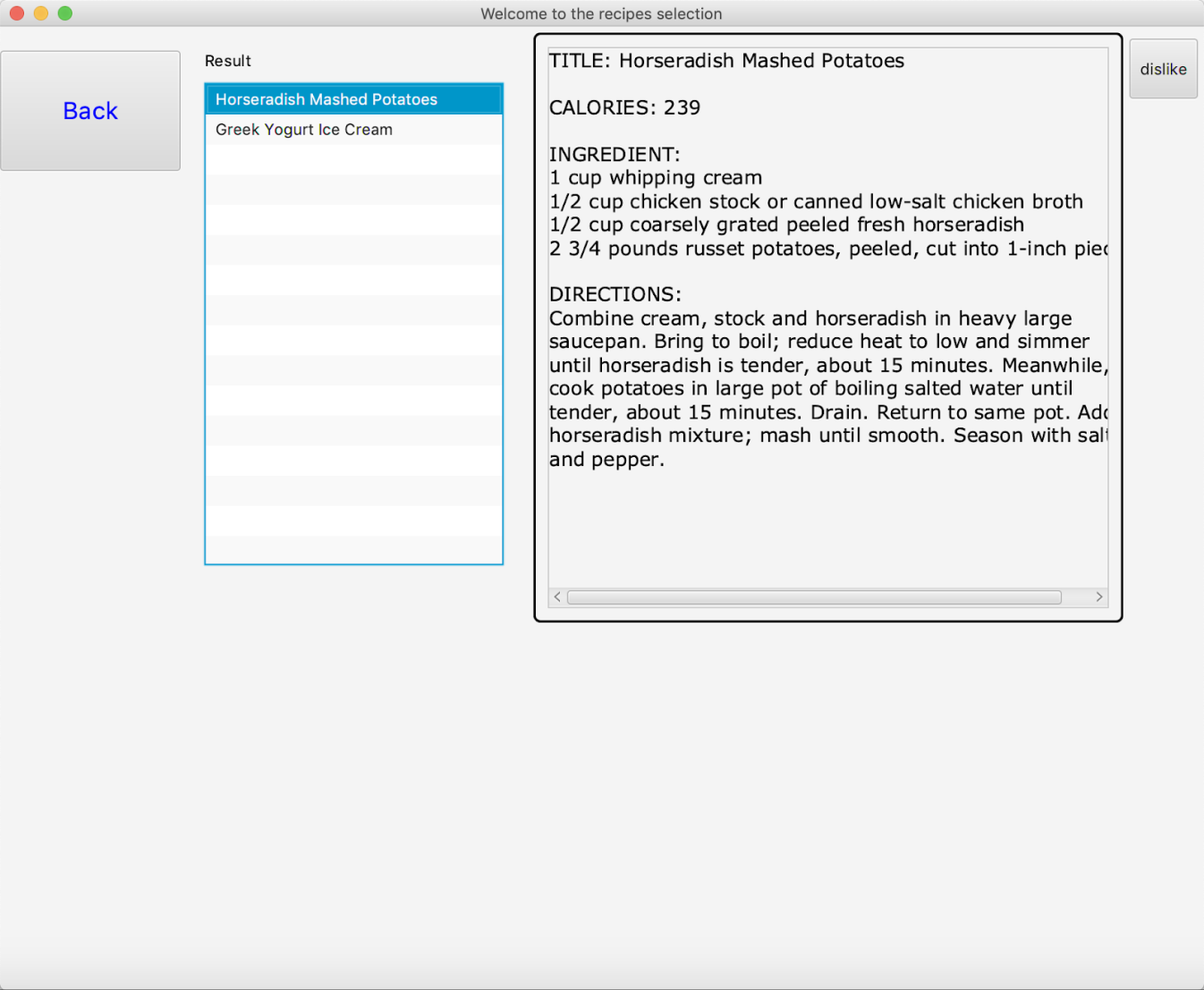


Figure 11. User Favorite Recipes Page.

II.6 Evaluation

The general evaluation plan was to go through each planned functional element (see items 1-6 in I.3 Application Description) and determine whether the prototype fulfilled that element. For each of those elements, the plan was to further break down those elements into the smallest feasible units to be tested. Some selected example general evaluations are listed below for the data cleaning, backend and frontend portions of our application. Following that, tests of the integration between front and backend functions are shown, defined by their success criteria, and with the actual results from those tests.

Data Cleaning Tests:

Is the character encoding consistent, and restricted to a standard (e.g. UTF-8 and unicode). Unsupported character sets are relatively easy to identify programmatically, or visually.

**Results: The data sets are Unicode (character set), encoded by UTF-8 (encoding). This was enforced after writing out the ‘joined’ recipe-to-category data.**

How sparse is the data, and/or do we need to remove elements with missing attributes? Test for null values in required attributes (e.g. title, calories). Is there still enough data (>10,000 rows in one table) after removing rows with missing elements?

**Results: The category data is sparse (<2% of possible relationships), and about one in five of the initial 20,000 recipes did not have calorie information. We ended up with roughly 16,000 recipes after dropping duplicates and removing those without required information.**

Are the primary keys working, and non-key elements conforming to normal forms?

**Results: Yes. By defining an integer primary key for every entity set, we are guaranteed to have the data following the 3rd Normal Form (since all attributes are defined by a unique value). A potential issue is that this allows duplicates, so all recipe data is checked for duplicates before being loaded into the database.**

Backend:

Will the DBMS load the data (keys loaded, data conforms to schema, etc.)?

**Results: This was tested for every table, and invariably there were issues with syntax (‘FIELDS OPTIONALLY ENCLOSED BY ‘“‘ TERMINATED BY ‘,’, etc.), mismatches between expected column ordering, or a truncated text value because we underestimated the maximum size of a text field. We did eventually pass this test for all tables.**

Can we send a query from the GUI to the database?

**Results: Yes. We tested all the queries we used by comparing the data we collected with the data returned by DBMS. Both are the same.**

Can we collect and parse the output of a query?

**Results: Yes. We tested all the queries at the mysql terminal first and when they worked out, we put them into Java string format.**

Frontend:

Do all design elements appear (e.g. does the recipe display correctly)? Success would be a properly displayed recipe.

**Results: Yes. At first, only the directions were displayed, but we were able to properly pull out and format the full recipes.**

Are all design elements functional (i.e. do all buttons work)?

**Results: We found that that the initial calorie threshold mechanism was not working, due to an improper ordering of the input data. After fixing the data source and re-importing, the calorie threshold search works.**

Example test cases:

**Case 1**

Test: Insert user in users; As during ‘registration’

Reasoning: Check that database connection is working, and INSERT works.

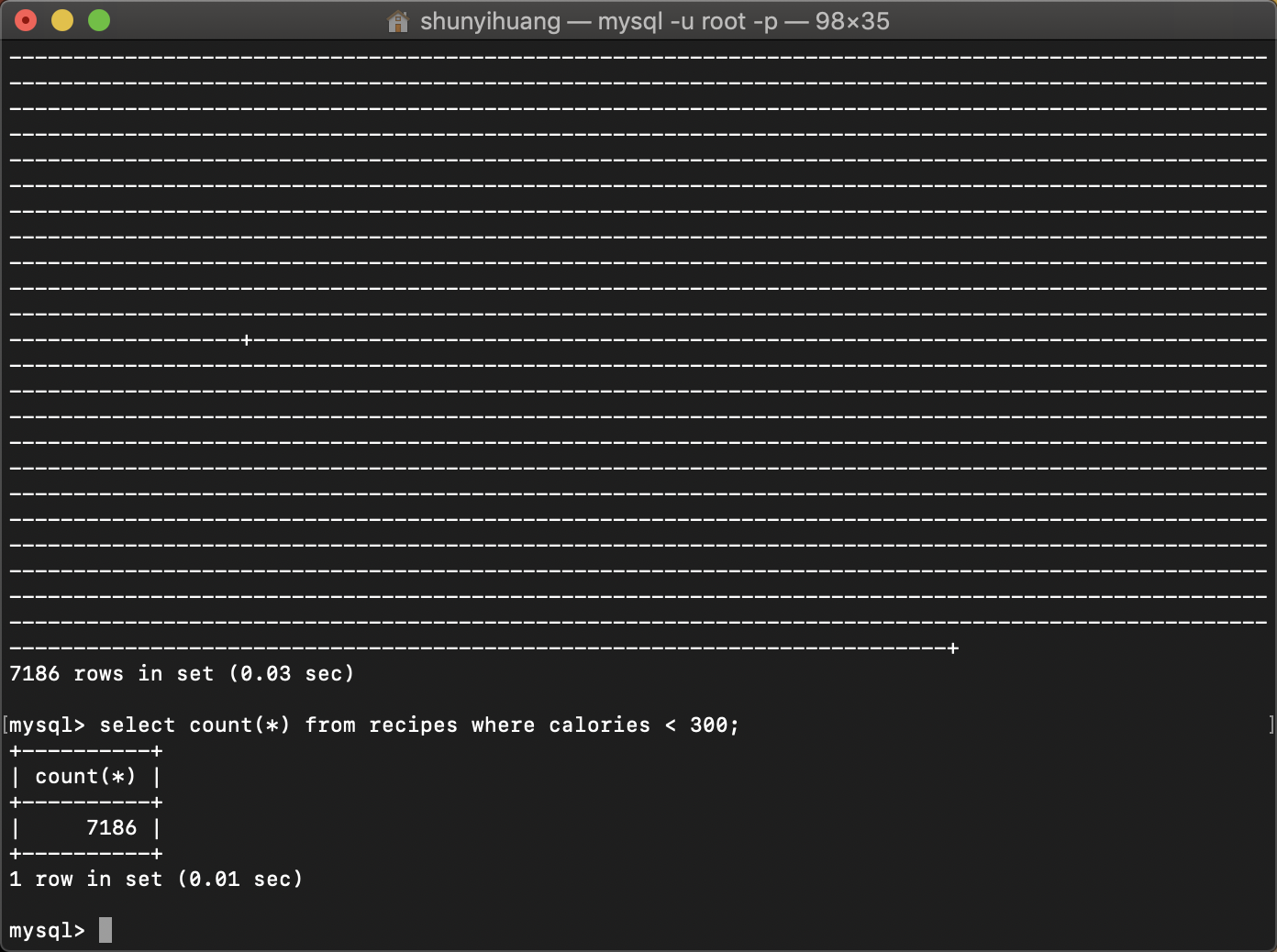
Criterion for success -- select \* from users returns inserted user

**PASSED**

**Case 2**

Test: Search for Recipes where Calories less than 300

Reasoning: Check that simple conditional Queries return results



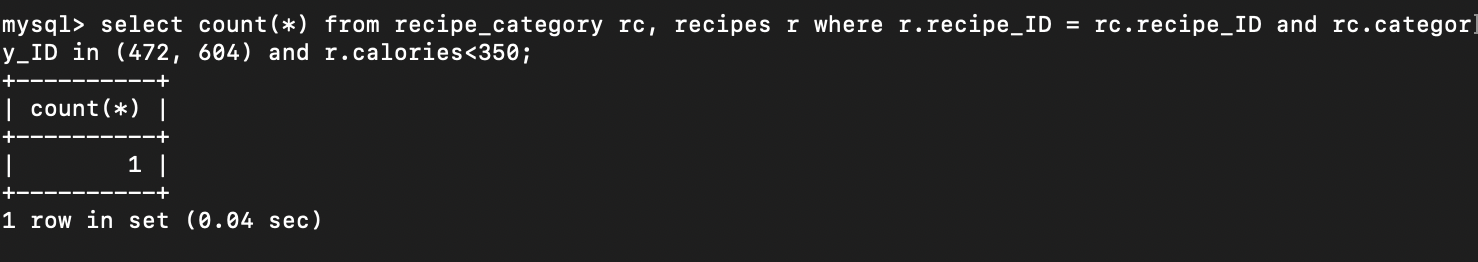
Criterion for success -- match the number of returned results

**PASSED**

**Case3**

Test: Search for Recipes where Category in [Pizza, Taco] and Calories < 350

Reasoning: Check that joins on Category, and filtering on a recipe attribute work



--Criterion for success: The returned recipe\_ID should be the same number and match results returned using MS EXCEL.

VIA EXCEL:

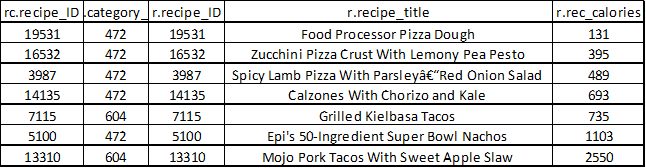
recipes.csv and relationship[...].csv--

categories (taco = 604, pizza = 472)

returned -- 7 recipes

filter on calories less than 350

returns -- 1 recipes

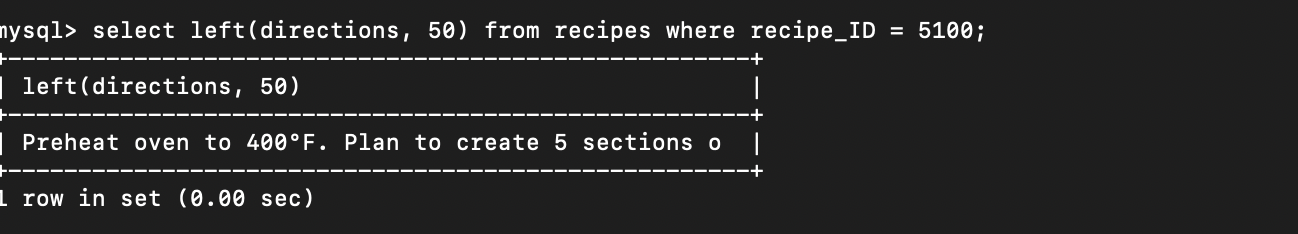
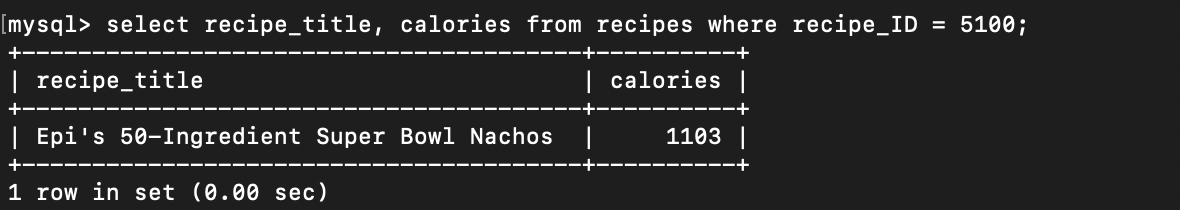


**PASSED**

**Case 4**

Test: Sanity Check -- Do the returned Recipe Title, Ingredients, and Directions Match?

Check for recipe\_ID == 5100



FROM EXCEL --

recipe\_ID: 5100

recipe\_title: Epi's 50-Ingredient Super Bowl Nachos

rec\_calories: 1103

rec\_directions: Preheat oven to 400°F. Plan to create 5 sections of nacho flavors[...]

rec\_ingredients: 6 (9-ounce) bags restaurant-style tortilla chips|2 cups store-bought|[...]

Criterion for Success: Title, calories, directions and ingredients should all match.

**Result: PASSED**

III. Conclusion

The authors would like to conclude with some remarks about some unexpected challenges that arose over the course of this assignment, as well as the relevant lessons learned.

The largest challenge that was not resolved, was that we initially planned to match each ingredient from the recipes to the USDA list of ingredients, if such a match could be made. This turns out to a be a non-trivial exercise in natural language processing. Unfortunately, the processing required to make that connection exceeded the scope and time allowances of this assignment. We then decided to manually match categories with an 'ingredient' type, to a specific USDA ingredient as a starting point to assist the users in finding nutrition information. It turns out that this too is a non-trivial problem, given the number of USDA values for a category like 'beef'. There are over 960 'kinds' of beef listed in the USDA nutritional values table (comprising roughly 11% of all entries). Which one do you pick? The USDA data is also hierarchical, such that 'squash', with or without salt, accounts for 30 separate records, which further specify, whether it was cooked or baked, drained or not drained, etc. and which continues fractally downward to a specific preparation of squash. Gaining some expertise with this data was the first challenge when setting the scope of this application.

From a database design perspective, the primary join of the application is between the Recipes entity set and the recipe-categories table. The performance implications of this prompted the authors to move away from the initial plan of using recipe titles (a potentially large varchar) as part of a super key, and toward using integer keys for all tables. Added care had to be taken to ensure that records were truly unique before loading them into the database, but since the Recipe entity set was not designed to take user input or change size, the extra cleaning of the data up-front was deemed worth the improved performance.

When going to implement the GUI, it turns out that hours can be spent developing a seemingly simple bit of functionality (e.g. getting the database to connect to the application, or parsing database output). While there were great ambitions early on in the project, and multiple layers of searches and filtering, or recommending based on some 'hedonic value' metric, the final determinant of what the core functionality would be was how long it would take to implement any given feature in the GUI.

Lastly, the importance of testing every point in the data pipeline was made abundantly clear when we stumbled on an error that seemed to be impossible. We learned the lesson that any time a table is truncated or dropped to be re-loaded, that it should be checked again in the console to make sure that exactly the same procedure was followed, and that all of the columns are in the correct location. One should check the schema against the actual columns in your imports, and do not assume they are the same.

In all, there were too many lessons learned to list here, and the authors appreciate the exercise of creating a database application with so few restrictions.

IV. References

1. Google Play Store, Food and Drink; Source: https://play.google.com/store/apps/category/FOOD\_AND\_DRINK?hl=en\_US
2. Apple App Store, Food and Drink; Source: https://apps.apple.com/us/genre/ios-food-drink/id6023
3. USDA.gov, Americans’ budget share for total food changed little during the last 20 years; Source:<https://www.ers.usda.gov/data-products/chart-gallery/gallery/chart-detail/?chartId=76967>
4. Recipes and Recipe Categories; Source:h[ttps://www.kaggle.com/hugodarwood/epirecipes](https://www.kaggle.com/hugodarwood/epirecipes)

5. Nutrition Information: <https://www.kaggle.com/trolukovich/nutritional-values-for-common-foods-and-products>

6. USDA Food Composition Database; <https://ndb.nal.usda.gov/ndb/doc/index>