Food Tracker: Know What You Eat

Yixue (Wendy) Feng, Wendi Kuang, Yuezhan Tao, Xuchen Wang

December 27, 2019

1 Introduction

According to a study conducted by the National Center of Health Statistics at the Center for Disease Control and Prevention in 2017, approximately 40 % of US adults aged 20 and above were obese [1]. Since obesity may cause high blood pressure, stroke, heart disease, stress incontinence, type-2 diabetes, and many other serious diseases [2], it is essential to keep track of daily intakes. However, despite the availability of numerous food applications, very few can actually give detailed and comprehensive information on the nutrients for the major foods available on the market.

In order to resolve this problem, we came up with the idea of developing an user interface that uses a food database to provide detailed nutrition information for popular food on the market. For this project, we used a set of data provided by the National Agricultural Library to create a food database and developed a web application that can retrieve the useful food information. This web application provides three main functionalities. First of all, users can retrieve the nutrition information of a specific food they eat or wish to eat. Secondly, by entering multiple food that a user has eaten throughout the day, our application can calculate how much nutrition he/she has consumed. Moreover, food suggestions can also be made based on users' needs and goals. We wish, by providing the detailed nutritional information for the popular foods people usually eat, users can be aware of what they eat and make diet/work-out plans accordingly.

2 Data Source & Technologies

2.1 Data Description

The data used in this project is downloaded from the official website of Food Data Central, managed by the Agricultural Research Service and hosted by the National Agricultural Library. It is an integrated data system that provides nutrient profile data come from a variety of sources. The data sets can be found at: `https://fdc.nal.usda.gov/download-datasets.html`.

The food table includes all the foods used in our database. This table is the primary table that links all the others together. The branded-food tables include a subset of food that have brands. The food-category table splits the food into 28 different categories, like vegetable and pork products. The nutrient table and the food-nutrient table indicate the nutrient names and the amount each

type of food contains. Lastly, the nutrient-conversion-factor table and the calorie-conversion-factor table indicate the calorie values of protein, fat, and carbohydrate for each food.

2.2 Data Pre-processing

1. The downloaded data are in 33 *CSV* files, and each one is a unique table with multiple attributes. We selected a subset of *CSV* files and filtered out the attributes are not needed in our application.

2. We dealt with the missing data by dropping the *NA* values in the files and preserved the necessary ones for later calculations.

3. We cleaned the data by changing all the characters to lowercase, connecting strings using underscores, and adjusting the data type for some attributes with **R** (e.g., converted all IDs to type int).

4. For food with null food_category_id, we created another row in food_category table and assigned these foods to the category "*Unknown*."

5. We pre-processed food descriptions containing useless information like "broccoli, raw, 2 bunches (in1) - nfy0905dd" and removed the meaningless last part.

6. We split descriptions in the food table into two attributes: descMajor and descMinor, to allow for a broad search using descMajor and narrow search using descMinor.

2.3 Summary statistics

The following are the summary statistics of the table after pre-processing.

Dataset Name	File Size	Number of Rows	Number of Attributes
food.csv	13MB	292060	4
branded_food.csv	80.5MB	260365	5
food_nutrient.csv	75.4MB	4800957	3
nutrient_conversion_factor.csv	142KB	10919	2
calorie_conversion_factor	94KB	4767	4
nutrient	6KB	227	3
food_category	765B	29	2

Table 1: All important datasets used in this project

2.4 Data Ingestion

2.4.1 Local MySQL database

The data was first populated into the local database through the terminal using the procedure and command listed below.

1. We created a database called "food_tracker" in MySQL database in our local machine
2. We used DDL statements in "data_processing/create_table.sql" to create tables in our local machines. The detailed DDL statements can be found in Appendix A.
3. We loaded data into database with statements in "data_processing/load_data.sql"

```
LOAD DATA INFILE "file-path"
INTO TABLE food_category
FIELDS TERMINATED BY ‘,’
ENCLOSED BY ‘"’
LINES TERMINATED BY ‘\n’
IGNORE 1 ROWS;
```

It is worth noting that all tables must be created and loaded in the following order to meet the foreign key constraints, and all code can be found in Appendix A.

1. Food Category
2. Food
3. Branded Food
4. Nutrient
5. Food Nutrient
6. Food Calorie Conversion Factor
7. Food Nutrient Conversion Factor

2.4.2 AWS MySQL database

The database was migrated to the AWS by dumping the local database out and loading them into the cloud database. The local dump file was created by the following command:

```
sudo mysqldump \
    --databases food_tracker \
    --master-data=2  \
    --single-transaction \
    --order-by-primary \
    -r backup.sql \
    -u root \
    -p
```

After this process, we can log into the cloud database and then use the "source backup.sql" command to load the dump data into the cloud-hosted MySQL database. The database configuration file was adjusted to connect our application to this cloud-hosted database.

2.5 Relational Schema and ER diagram

2.5.1 Relational Schema

The following is the relational schema used in our database:
food(fdc_id, descMajor, descMinor, food_category_id)
–food_category_id REFERENCES food_category(food_category_id)
branded_food(fdc_id, brand_owner, ingredients, serving_size, serving_size_unit)
– fdc_id REFERENCES food(fdc_id)
food_category(food_category_id, description)
food_nutrient(fdc_id,nutrient_id, amount)
– fdc_id REFERENCES food(fdc_id)
– nutrient_id REFERENCES nutrient(nutrient_id)
nutrient(nutrient_id, name, unit_name)
food_nutrient_conversion_factor(nutrient_conversion_id,fdc_id)
– fdc_id REFERENCES food(fdc_id)
– nutrient_conversion_id REFERENCES food_calorie_conversion_factor(nutrient_conversion_id)
food_calorie_conversion_factor(nutrient_conversion_id, protein_value, fat_value, carbohydrate_value)

2.5.2 ER Diagram

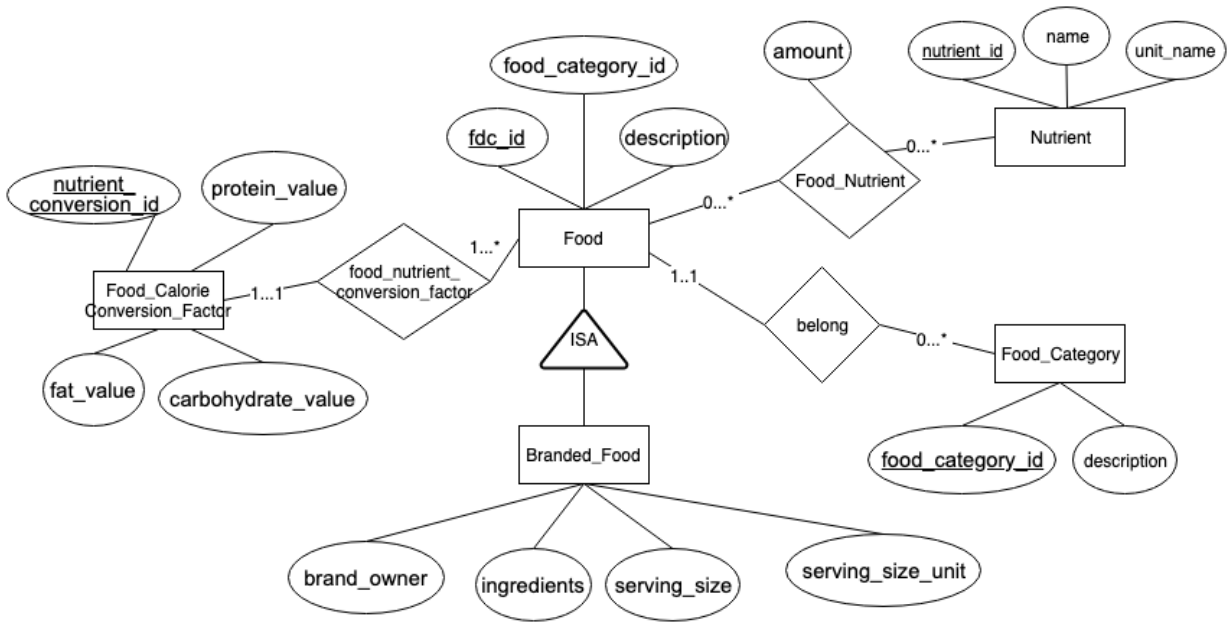

Figure 1: ER diagram.

2.6 Normal Form

All our relations are in BCNF, as, in each table, all functional dependencies are implied by the keys:

food: fdc_id $\rightarrow descMajor, descMinor, food\_category\_id$

branded_food: fdc_id $\rightarrow brand\_owner, ingredients, serving\_size, serving\_size\_unit$

food_category: food_category_id $\rightarrow description$

food_nutrient: fdc_id, nutrient_id $\rightarrow amount$

nutrient: nutrient_id $\rightarrow name, unit\_name$

food_nutrient_conversion_factor: nutrient_conversion_id, fdc_id $\rightarrow nutrient\_conversion\_id, fdc\_id$

food_calorie_conversion_factor: nutrient_conversion_id $\rightarrow protein\_value, fat\_value, carbohydrate\_value$

2.7 Technology Used

- Data pre-processing: Python (pandas, numpy), R.
- Web development: "MEAN" stack (MySQL, Express, AngularJS, Node), HTML, CSS.
- Cloud technology: AWS relational database service (RDS)-MySQL

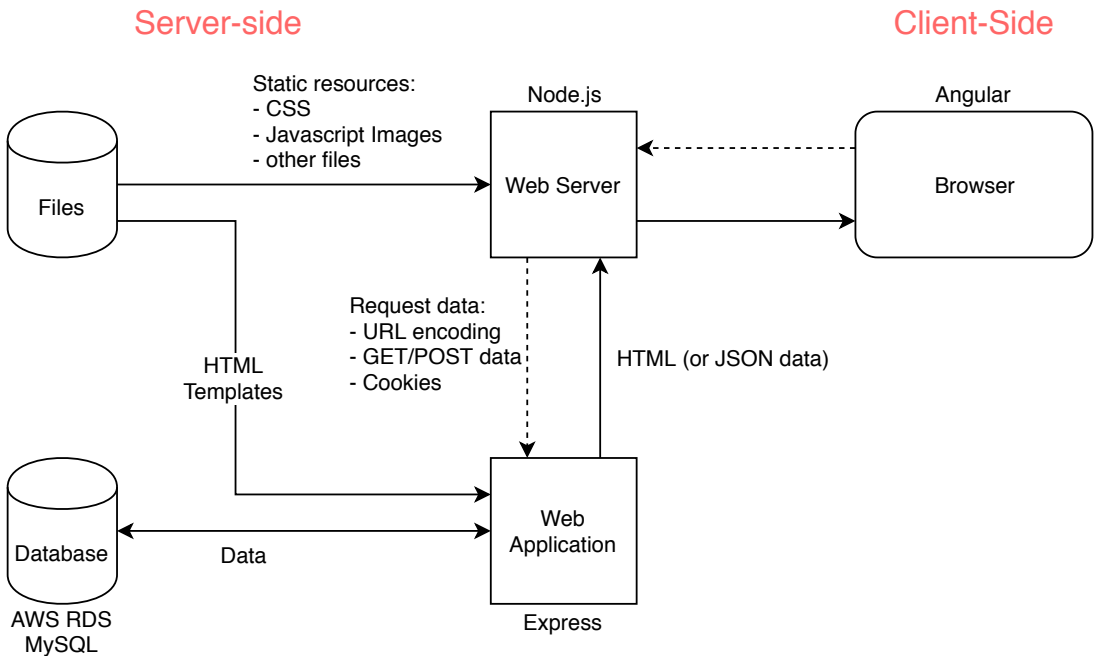

Figure 2: Dynamic web app architecture

3 System Architecture

Below are the descriptions of our web pages and what each page accomplishes:

- ***Home***
  The home page provides a brief introduction to the main functionalities of this application. We also implemented a flip-card function at the bottom of our website, where the user could preview the food in the common food categories. The food preview is dynamically generated by connecting to the back-end every time the home page is refreshed.

- ***Search Food***
  The search page allows users to search for their favorite food and returns the nutrition information. We expect users to type in a generic name for the food they wish to look into, such as eggs, yogurt. To make their searches easier and smoother, users can choose food (e.g., brown organic eggs or Chobani blueberry Greek yogurt) from a drop-down list, and the information regarding that food will be returned on the right-hand side of the website with a serving size of 100 grams.

- ***Daily Nutrition***
  The daily nutrition page assists the user in calculating the total nutritional contents, namely protein, carbohydrates, and fat, for any food items. The users first search for a specific food, add it to a running list of food items, and type in the amount they have consumed for each item. The total amount of macros will be calculated and displayed on the page in units of grams.

- ***Recommendation***
  The recommendation page includes two major parts, including the recommendations of foods made based on the users' needs and goals and a few live Twitter feeds from food channels. In the first part, users can specify the percentage of carbohydrate, fat, or protein they wish to take from food, and this application will return the items that meet this criterion. The second part contains three Twitter feeds, including two cooking channels and a Philadelphia restaurant magazine.

4 Queries

In total, we implemented 11 queries in this project to achieve the above functionalities. To better illustrate these queries, five of them are selected and discussed below.

- Query 1 returns all the food in the Dairy and Egg product category.

- Query 2 retrieves the major and minor descriptions of top twenty foods that contain a certain fizzy description term entered by the user.

- Query 3 returns the name of the nutrient and its amount for a specific food.

- Query 4 computes the protein, fat, and carbohydrate values based on the food and the amount the user inputs.

- Query 5 is the most complicated one used on our website. It is used to make a recommendation of foods based on the range of either protein, fat, or carbohydrate values entered by our users.

5 Performance Evaluation

We performed query optimization on the two most complicated queries deployed on our website.

The first one calculates the amount of three nutrients (i.e., protein, fat, and carbohydrate) for a specific food that the user provided in the Daily Nutrient page. We joined the food, nutrient, and food-nutrient tables three times for the three nutrients and then combined the three temporary tables together. The food_nutrient table represents the many-many relationship between the food table and the nutrient table, containing 4,800,957 rows and is approximately 75.4 MB in size. The original query took roughly 0.05 sec because we used cross join over the three tables and did full table scans. This was fast but not optimal, so we used inner join later and sped up the query to around 0.01 sec. Since 0.01 sec is already fast enough, we did not use any indices or caching for this query.

The second query we optimized is the one used in the Recommendation page. This query calculates the percentage of the calories each nutrient (protein, fat, and carbohydrate) has in each food and returns the ones that satisfy the users' requirements. This one is the most complex query on our website because it joins five tables (food, nutrient, food-nutrient, food-nutrient-conversion-factor, and food-calorie-conversion-factor) three times to calculate the calorie. Besides the food-nutrient and food tables, which have 4,800,957 rows and 292,060 rows, respectively, the food-nutrient-conversion-factor table also has 10,919 rows, matching the food table with the food-calorie-conversion-factor table. The original query took roughly 1.70 sec to return the result using the cross join. After switching to inner join, we improved the query performance by approximately 0.14 sec. We then experimented with different indices. Adding indices on the food-nutrient table would speed up the running time to approximately 1.40 sec, and adding an indices on the food-nutrient-conversion-factor table would speed up the running time to approximately 1.50 sec. This is probably because that the food-nutrient has significantly more rows than the food-nutrient-conversion-factor. Therefore, choosing an appropriate joining method, as well as indexing the food-nutrient table, improved the performance.

6 Technical Challenges

The first challenge that we faced was cleaning the datasets. A significant portion of the data has meaningless numbers in the end, which makes the data extremely difficult to read.

The second challenge is how to load the data file into a local machine efficiently. If loading the data with MySQL Workbench, the loading time becomes exceptionally long. It will take hours to load a single table into the database. Our final solution is modifying the configuration file for MySQL and loading the data from the command line; then it only takes seconds to minutes to load a table.

The third challenge that we faced during the development is understanding and using the description of the food in the food table. Many descriptions are very similar to each other and are hard to read. In order to resolve this problem, we used Python to split the description into major and minor

descriptions. Major descriptions include the main food type, and the minor descriptions indicate the slight difference between the food. By doing so, not only the presentation of data becomes more precious, but also the results can be further used as the inputs to another query.

The fourth challenge is tuning the front-end of the website. Our application is very dynamic and there are usually multiple Angular components in one controller. Often times, these components are connected to each other and can take many forms such as strings or arrays. Connect these components to the back-end was a challenge for us as well.

The last challenge is making the front-end responsive, such as making navigation bar a toggle bar when shrinking the page, or have components be stacked vertically or horizontally depending on the width of the web page.

7 Extra Credits

- The whole database has been migrated to the AWS cloud.
- Twitter real-time steaming data was attached at the bottom of the "Recommendation" page.
- AngularJS was used as the front-end web framework to write the web

References

[1] C. M. Hales, M. D. Carroll, C. D. Fryar, and C. L. Ogden, "Prevalence of obesity among adults and youth: United states, 2015–2016," 2017.

[2] A. Lang and E. S. Froelicher, "Management of overweight and obesity in adults: behavioral intervention for long-term weight loss and maintenance," *European Journal of Cardiovascular Nursing*, vol. 5, no. 2, pp. 102–114, 2006.

Appendix A: Table creation DDLs

DDL 1: Food

```
CREATE TABLE food(
fdc_id INT NOT NULL,
description VARCHAR(40),
food_category_id INT,
PRIMARY KEY(fdc_id),
FOREIGN KEY(food_category_id)
REFERENCES food_category(food_category_id));
```

DDL 2: Branded Food

```
CREATE TABLE branded_food(
fdc_id INT NOT NULL,
brand_owner VARCHAR(40),
ingredients VARCHAR(40),
serving_size DECIMAL(10, 3),
serving_size_unit VARCHAR(5),
PRIMARY KEY(fdc_id),
FOREIGN KEY(fdc_id)
REFERENCES food(fdc_id));
```

DDL 3: Food Category

```
CREATE TABLE food_category(
food_category_id INT NOT NULL,
description VARCHAR(40),
PRIMARY KEY(food_category_id));
```

DDL 4: Food Nutrient

```
CREATE TABLE food_nutrient(
fdc_id INT NOT NULL,
nutrient_id INT NOT NULL,
amount DECIMAL(10, 3),
PRIMARY KEY(fdc_id, nutrient_id),
FOREIGN KEY(fdc_id)
REFERENCES food(fdc_id),
FOREIGN KEY(nutrient_id)
REFERENCES nutrient(nutrient_id)
);
```

```
CREATE TABLE food_nutrient(
fdc_id INT NOT NULL,
nutrient_id INT NOT NULL,
amount DECIMAL(10, 3),
PRIMARY KEY(fdc_id, nutrient_id),
FOREIGN KEY(fdc_id)
REFERENCES food(fdc_id),
FOREIGN KEY(nutrient_id)
REFERENCES nutrient(nutrient_id)
);
```

DDL 5: Nutrient

```
CREATE TABLE nutrient(
nutrient_id INT NOT NULL,
name VARCHAR(255),
unit_name VARCHAR(255),
PRIMARY KEY(nutrient_id)
);
```

DDL 6: Food Nutrient Conversion Factor

```
CREATE TABLE food_nutrient_conversion_factor(
nutrient_conversion_id INT NOT NULL,
fdc_id INT NOT NULL,
PRIMARY KEY(nutrient_conversion_id, fdc_id),
FOREIGN KEY(fdc_id)
REFERENCES food(fdc_id),
FOREIGN KEY(nutrient_conversion_id)
REFERENCES
food_calorie_conversion_factor(nutrient_conversion_id)
);
```

DDL 7: Food Calorie Conversion Factor

```
CREATE TABLE food_calorie_conversion_factor(
nutrient_conversion_id INT NOT NULL,
protein_value DECIMAL(10, 3),
fat_value DECIMAL(10, 3),
carbohydrate_value DECIMAL(10, 3),
PRIMARY KEY(nutrient_conversion_id)
);
```

Appendix B: List of Queries deployed in this project

Query 1:

```
SELECT DISTINCT descMajor
FROM food
WHERE food_category_id = 1 AND LENGTH(descMajor)< 30;
```

Query 2:

```
SELECT DISTINCT  F.descMajor, F.descMinor, F.fdc_id
FROM food F
WHERE lower(descMajor) LIKE '%beef%'
LIMIT 20;
```

Query 3:

```
SELECT FN.amount, N.name
FROM food_nutrient FN JOIN food F ON FN.fdc_id = F.fdc_id
JOIN nutrient N ON FN.nutrient_id = N.nutrient_id
WHERE F.fdc_id = `xxxxx` AND N.name NOT LIKE '%:%'
```

Query 4:

```
WITH temp_protein AS(
SELECT F.fdc_id, FN.amount*(`+amount+`/100) AS protein_value
FROM food F JOIN food_nutrient FN ON F.fdc_id = FN.fdc_id
JOIN nutrient N ON FN.nutrient_id = N.nutrient_id
WHERE N.name = 'protein' AND F.fdc_id=`+id+`
),
temp_fat AS(
SELECT F.fdc_id, FN.amount*(`+amount+`/100) AS fat_value
FROM food F JOIN food_nutrient FN ON F.fdc_id = FN.fdc_id
JOIN nutrient N ON FN.nutrient_id = N.nutrient_id
WHERE N.name = 'carbohydrate, by difference' AND F.fdc_id=`+id+`
),
temp_carb AS(
SELECT F.fdc_id, FN.amount*(`+amount+`/100) AS carbohydrate_value
FROM food F JOIN food_nutrient FN ON F.fdc_id = FN.fdc_id
JOIN nutrient N ON FN.nutrient_id = N.nutrient_id
WHERE N.name = 'total lipid (fat)' AND F.fdc_id=`+id+`
)
SELECT TP.fdc_id, TP.protein_value, TF.fat_value, TC.carbohydrate_value
FROM temp_protein TP JOIN temp_fat TF ON TP.fdc_id = TF.fdc_id
JOIN temp_carb TC ON TP.fdc_id = TC.fdc_id
```

Query 5:

```
WITH temp_protein AS(
SELECT F.fdc_id, FN.amount * CCF.protein_value AS protein_value
FROM food F JOIN food_nutrient FN ON F.fdc_id = FN.fdc_id
JOIN nutrient N ON FN.nutrient_id = N.nutrient_id
JOIN food_nutrient_conversion_factor NCF ON F.fdc_id = NCF.fdc_id
JOIN food_calorie_conversion_factor CCF ON NCF.nutrient_conversion_id = CCF.
    nutrient_conversion_id
WHERE N.name = 'protein'
),
temp_fat AS(
```

```
SELECT F.fdc_id, FN.amount*CCF.fat_value AS fat_value
FROM food F JOIN food_nutrient FN ON F.fdc_id = FN.fdc_id
JOIN nutrient N ON FN.nutrient_id = N.nutrient_id
JOIN food_nutrient_conversion_factor NCF ON F.fdc_id = NCF.fdc_id
JOIN food_calorie_conversion_factor CCF ON NCF.nutrient_conversion_id = CCF.
    nutrient_conversion_id
WHERE N.name = 'carbohydrate, by difference'
),
temp_carb AS(
SELECT F.fdc_id, FN.amount* CCF.carbohydrate_value AS carbohydrate_value
FROM food F JOIN food_nutrient FN ON F.fdc_id = FN.fdc_id
JOIN nutrient N ON FN.nutrient_id = N.nutrient_id
JOIN food_nutrient_conversion_factor NCF ON F.fdc_id = NCF.fdc_id
JOIN food_calorie_conversion_factor CCF ON NCF.nutrient_conversion_id = CCF.
    nutrient_conversion_id
WHERE N.name = 'total lipid (fat)'
),
temp1 AS(
SELECT TP.fdc_id,
TP.protein_value/(TP.protein_value+TF.fat_value+TC.carbohydrate_value) AS
    protein_value,
TF.fat_value/(TP.protein_value+TF.fat_value+TC.carbohydrate_value) AS fat_value,
TC.carbohydrate_value/(TP.protein_value+TF.fat_value+TC.carbohydrate_value) AS
    carbohydrate_value
FROM temp_protein TP JOIN temp_fat TF ON TP.fdc_id = TF.fdc_id
JOIN temp_carb TC ON TP.fdc_id = TC.fdc_id
)
SELECT DISTINCT F.descMajor
FROM food F JOIN temp1 T ON F.fdc_id = T.fdc_id
WHERE T.`+myData_name+`_value >= `+myData_min+` AND T.`+myData_name+`_value <= `+
    myData_max+`
ORDER BY F.descMajor;
```