GitHub – https://github.com/watsonkh/CMSC608-AdvancedDatabaseProject

CMSC 608
Advanced Database
Semester Project

Project Members
Paula Gearon
Luke Unterman
Kyle Watson

Recipe Database

Problem Description

Recipes are a very common form of data, with people creating and writing recipes, since the earliest known examples from Mesopotamia around 1700 BCE, to the present day. The modern form of recipes with ingredients followed by a method first appeared in the 19th century, and this style has been maintained across most modern cooking books, private recipe collections, and online recipes.

With such a long history, the quantity of available data is enormous, and constantly expanding. Many people maintain their own collections, and these can be useful when they include just a few dishes, all the way through to systems containing more meal descriptions than could be sampled in a lifetime. Systems that can support recipe collections at various scales provide significant utility as they can be used for purposes ranging from an individual cook keeping their family recipes, through to providing a central source for large communities to access.

Modern recipes are expected to follow a specific format, which connects various types of data in specific ways. Ingredients must be presented in the order in which they are used, and must use appropriate units of measurement. The method will be presented as a series of steps, which must be in the correct order. Metadata is also expected, including obvious elements like title, the number of servings, the original source, and more.

At the same time, a recipe is not a simple set of text, but describes real-world items and processes. Different uses of a recipe may require modifications when being applied, such as substituted ingredients, conversions of measurements, or more. Storing each element of the recipe in a semantically useful way, allows such modifications to be made automatically for a user.

Consequently, a useful recipe database will implement a structure that can represent all the elements of a recipe, with sufficiently detailed structure to provide the flexibility to grow with a user's needs. It should provide an interface that displays recipes in the expected format, while also enabling easy searching. While it should be easy for users to add to and modify the collection, it should also come with a set of recipes to encourage a user to explore and experiment with in their kitchen, giving them a place to start and grow with.

Implementation

The system has been built as a web application run from a web server, connected to an external relational database. Given that a recipe database is likely to be run by individual users, they are likely to be close to the web server, meaning that the application can be run within the server, allowing the user interface to be minimal.

Database

To allow for advanced searching features, the database needs to handle vector indexes, or to integrate with an external vector database. Selecting PostgreSQL allows the use of the *pgvector* module, which integrates this into a standard installation. Systems based on PostgreSQL/pgvector are very common, and are available as a Docker image, making installation and maintenance fast and easy.

Web Server

Flask was selected for the web server for a series of reasons:

- Web pages can be built independently in HTML/CSS/JS
- Pages can be built as templates, and expanded by database access
- Microservices are easy to build, so pages can update themselves using live data
- Simple setup, and uses Python, which is familiar to the team
- Simple deployment with the Gunicorn web server

Text Search

Text search was implemented with both an exact search and a semantic search. The exact search directly uses a SQL query to match recipe names, descriptions, steps, and ingredients. The semantic search is a little more complicated. A pre-trained sentence transformer, “all-mpnet-base-v2”, was used to create an embedding from each recipe’s name, description, steps, and ingredients. These embeddings are stored in a table of their own, with a foreign key that links to a recipe id. When a user searches, their query is transformed into an embedding that is compared with the pre-computed recipe embeddings. The 10 most similar recipes are displayed to the user.

Image Search

Image search was implemented in a similar manner. Each recipe’s image is transformed into a vector with a pre-trained model from OpenAI entitled “clip-vit-base-patch32”. When a user searches for a similar image, the image they submit is transformed into a vector of the same number of dimensions, and the images that are the most similar are presented to the user. Note that instead of performing an exact nearest neighbor search to find the most similar image embeddings, an HNSW index is used to more quickly query approximate nearest neighbors.

Design

The design diagrams for this project were built using Draw.io. This is a free utility, based on SVG web standards.

Execution

While the application can be configured manually from its elements, the ideal user experience is to run it in Docker.

- Download the project with:
  `git clone git@github.com:watsonkh/CMSC608-AdvancedDatabaseProject.git`
- Change directory into the project directory:
  `cd CMSC608-AdvancedDatabaseProject`
- Build and start the Docker container:
  `docker-compose up --build`
- Navigate to the application:
  http://localhost:5000/

The container can be stopped with [`Ctrl-C`], or from another terminal with the command:

```
docker-compose down
```

To close the container and clear the database entirely, use:

```
docker-compose down -v
```

Entity-Relationship Design

The principal independent entities are **Recipe**, **Ingredient**, and **Unit**.

Recipe

Each **Recipe** has multiple **Recipe_Ingredient**s which describe an **Ingredient**, and a quantity of a **Unit**, along with the user interface information of which order the ingredient should appear in.

Recipes also contain multiple **Step**s, which contain a description of what the cook should do, an order that the step should appear in, and an optional URL for an image associated with the step.

Recipes may also have optional **Image**s, which contain a URL for an appropriate picture of the dish, and a vector encoding to make the images searchable.

Ingredient

Ingredients contain their description, and an optional URL to link to external information about the ingredient, such as a Wikipedia page.

Ingredients also reference the primary **Unit** they are measured in.

Finally, an **Ingredient** may possibly have **Substitution**s, which describe a connection to other ingredients, along with notes about how substitutions may be made.

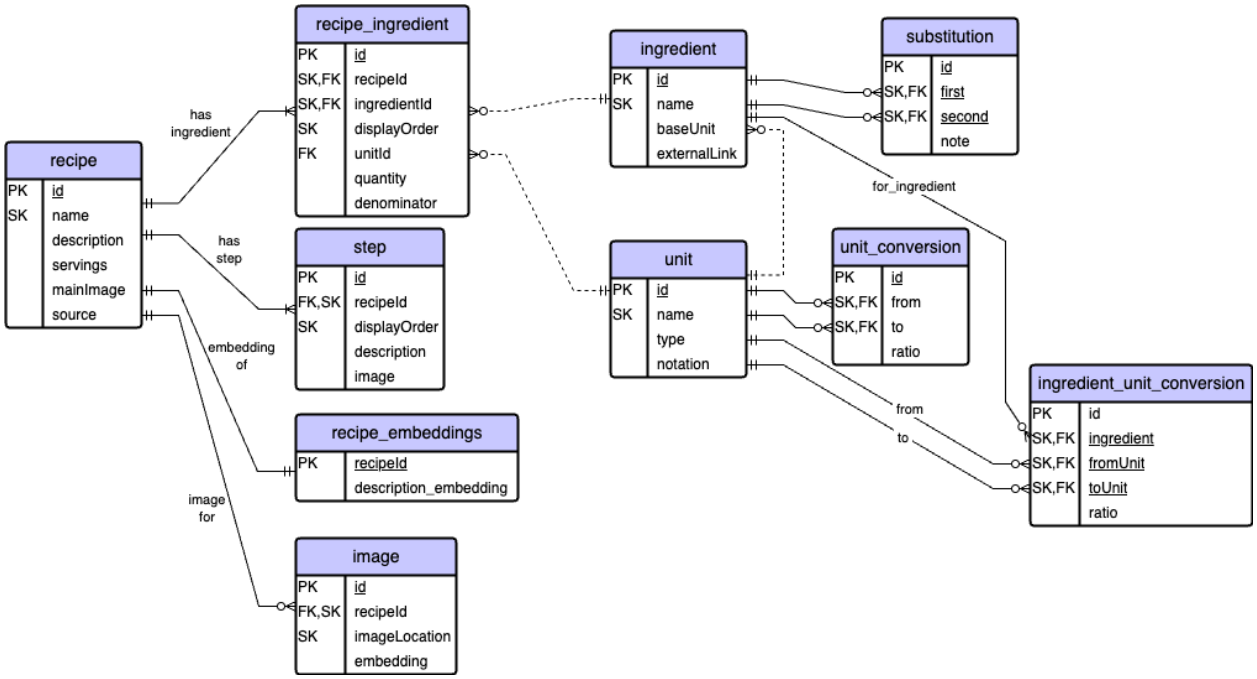
Unit

A **Unit** represents a unit of measurement by name, the type of unit (e.g. mass, volume, count), and the common notation for the unit. **Unit**s may also have multiple **Unit_Converstion**s which describes how to convert from one **Unit** to another by a given ratio. (e.g. 1tsp = 5ml).

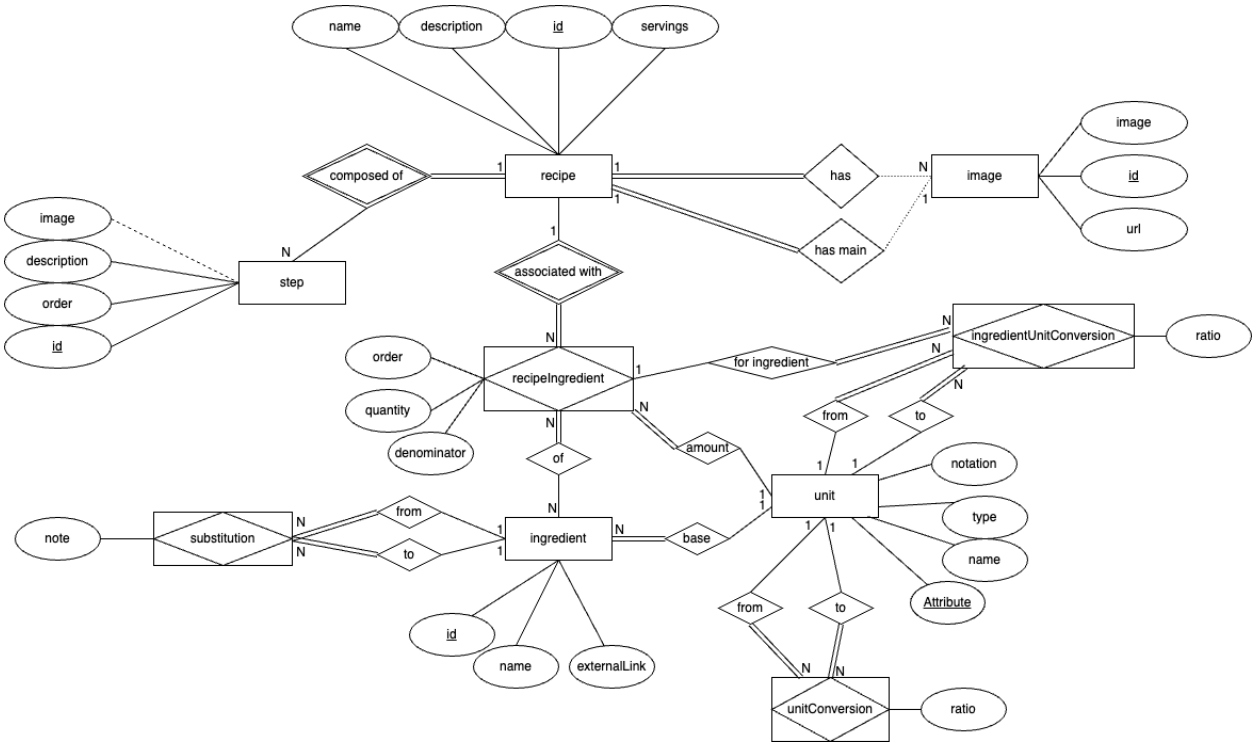
There is also the possibility of **Ingredient_Unit_Conversion**s which provide a ratio of one unit to another for a given ingredient (e.g. 1 cup flour = 120g flour).

ER Diagrams

The Crow's Foot ER Diagram for this is shown here



This has an associated Chen ER diagram:



Business Rules

In order to formally define the various constraints and use cases associated with the proposed RDBMS, we have created the following list of business rules:

- Users should be able to modify recipes retrieved by the recipe database to:
    1. Change the units of measurement used by a **Recipe_Ingredient**
    2. Change the types of units of measurement for some ingredients, going between mass and volume for ingredients where this makes sense (e.g. 1 cup flour = 125g flour)
    3. Swap ingredients with potential substitutes
- Users should be able to filter recipes by:
    1. Their adherence to different dietary restrictions (e.g., vegan, gluten-free)
    2. Estimated cooking duration
    3. Serving size
- Users should be able to upload an **Image** of a particular dish and retrieve **Recipe**s with similar image thumbnails
- Users should be able to query the proposed RDBMS for **Recipe**s with descriptions semantically similar to their “craving”/user input, ranked by their similar score in descending order
- Each **Recipe** should consist of at least one step and one ingredient
- Each **Step** in a **Recipe** should be listed in ascending order
- **Recipe**s and **Step**s can each optionally have **Image**s, which are stored as URLs
- **Recipe** descriptions should be at most 1,000 characters long, and **Ingredient** names should be at most 100 characters long

Data Acquisition

Initial data was acquired from the Serious Eats website by searching for recipe URLs listed in its XML sitemap which contained the term “recipe.” Then, for each of these recipe URLs, the Python library BeautifulSoup was used to scrape JSON-LD blocks containing recipe information (ingredients, steps, descriptions) from each Serious Eats webpage. However, because our data model required each ingredient to have a quantity, ingredient name, and unit, we had to further preprocess the scraped recipe data via Named Entity Recognition. More specifically, we prompted the Google Gemini 1.5 Flash LLM to convert each of the scraped recipe ingredient strings from the SeriousEats website into three different entities: “Quantity,” “Ingredient,” and “Unit.” This ultimately generated 100 initial recipes.

Data Preprocessing

This data needed extensive cleaning. Example errors included:

- Some recipes were saved as a series of `null` values.
- Some recipes had formatting that was impossible to parse.
- Some recipes were just random text in the structure.

Once clean, we parsed the file, converting minor problems into correct data, where possible. For instance, a 14oz can of tomatoes could appear as 14oz of canned tomatoes or 1 can of 14oz. Some recipes also described estimations, which we turned into specific values. One example was “a few sprigs of thyme” is converted to 3 units of thyme, where a unit has the name “sprig”.

After automated cleaning, the result was converted into SQL statements, which are all stored in the `init` directory for application initialization. These files are loaded by Docker during instance initialization.

Images associated with the recipes are converted into embedding vectors by the CLIP transformer model, and stored in the database. This uses the *pgvector* extension for PostgreSQL. The embedding is performed by a custom program called `flask/emeddings.py` which Docker executes after the SQL initialization is complete.

Limitations

- Minimalistic website user interface, could use additional “flourishes” (i.e., about section describing developers of the website, defined development roles) to make feel more complete
- Website only lists small subset of recipes scraped from Serious Eats website due to data preprocessing related losses
- Currently no distinction between “user” and “admin” roles – all visitors can access admin page to perform CRUD operations on data tables
- Unit conversions and ingredient substitutions not implemented on front-end

Challenges

While defining the schema for our RDBMS and creating the front-end was relatively straightforward, we encountered significant difficulties when collecting and preprocessing the recipe data. For example, many of the recipes we scraped from the SeriousEats website contained special unicode characters which broke our data loading pipeline, and responses from the Gemini LLM were generally inconsistent and difficult to parse. It was also slightly difficult to set up everyone's environments correctly, as our project relies on multiple frameworks, Postgres extensions, and Python libraries. Some of the non-technical difficulties we experienced include having to coordinate specific times to work on the project between three working adults and managing conflicting Git merges.

Future Work

Several features designed into the data model have minimal implementation due to lack of data.

- Ingredient-based unit conversion
- Ingredient substitutions
- Metric kitchen standards: a metric cup is 250ml, while a US cup is 237ml. A metric tablespoon is 15ml, while an Australian tablespoon is 20ml. These conversions need to be included.

While all of this requires extensive data, some tweaks to the user interface will also be required.

Extensions to the data model could also allow for greater user customization. For example, user notes, last time the recipe was made, recipe ratings, publishing local recipes to external sites, and more.

AI

There are also several new features that can incorporate a greater use of AI:

- Entering recipes is a very manual process. Using AI to parse text into a basic structure could make it easy for users to bring recipes from other sources without significant effort.
- Building on the user's preferences and history to suggest:
    - What to cook next
    - What can be made with available ingredients
    - How ingredients can be combined to create a new experimental recipe

Applications like this one evolve constantly, with new features being proposed and added all the time. These are some of the short term improvements that we have considered, but many more ideas have already been suggested by others.