2DIY4

Application that allows users to find restaurants that serve their favorite recipes, or find recipes of their favorite restaurants' dishes

Meghana Denduluri *meghdend@seas.upenn.edu*
Anna Orosz *aorosz@sas.upenn.edu*
Matthew Scharf *mschar@seas.upenn.edu*
Shreyas Sonbarse *sonbarse@seas.upenn.edu*

December 16, 2020
CIS 550 - Database and Information Systems
University of Pennsylvania
School of Engineering and Applied Sciences

INTRODUCTION AND PROJECT GOALS

Upon closing restaurants amid the Coronavirus lockdown in 2020, a lot of people have experienced an unexpected loss: millions of people lost the comfort of food, the joy of communal eating and gathering with friends and loved ones. To counteract that, and to alleviate some of the stress felt by many since March, people have turned to the joy of cooking; specifically to try and recreate some of the flavors and dishes that they remember and associate with easier and more joyful times.

The goal of our project was two-factored:

- Retrieve a variety of recipes of any dish from a chosen restaurant
- Find a list of restaurants serving the user's chosen recipe/dish

To build our database, we mainly utilized two datasets, namely the Yelp Open Dataset (for the restaurant dataset) and the Food.com Recipes and Interactions (for corresponding recipes). Furthermore, an extensive amount of web scraping was also needed to link restaurants and recipes, which will be further detailed in the report.

TECHNOLOGIES USED

- Python - Standard libraries (NumPy and Pandas) for data pre-processing
- Py_stringmatching and py_strings_im_join Python packages for entity resolution
- Requests, lxml, re, time, random, and json Python packages for web scraping
- Google colab notebook
- PySpark
- MySQL
- React.js, Express.js and Node.js for web development
- Amazon Web Services- RDS cloud database service

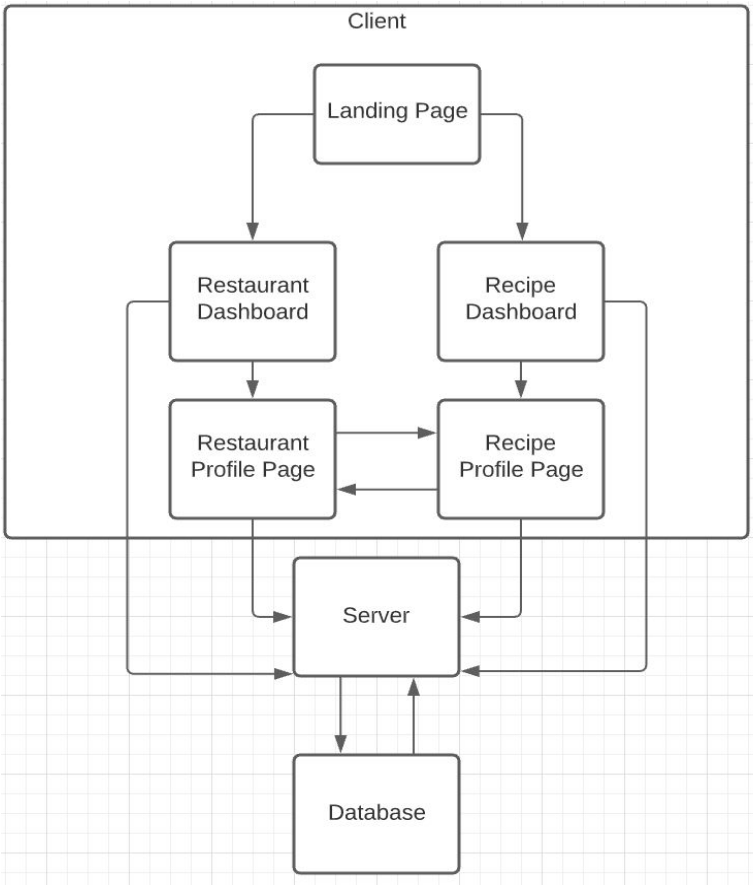
DESCRIPTION OF SYSTEM ARCHITECTURE

The client was built with React.js and the server was built with Node.js and Express.js. The database is a MySQL database hosted on AWS RDS.

The landing page leads you to either a restaurant dashboard or a recipes dashboard. Each dashboard allows you to either explore items by filtering on useful attributes (like tags, location or ingredients) or directly search the database for a specific

name. Both recipe and restaurant search results lead you to an item-specific profile page which contains far more detailed information about the item.

Each restaurant profile page lists the dishes served at that restaurant and recipes similar to those dishes. Those recipes link to the corresponding recipe profile page. The recipe profile lists the restaurants at which a dish similar to the recipe is served. Those restaurants link to the corresponding restaurant profile page. The result is an interconnected web of recipes and restaurants that can be explored.



DATA SOURCES

Yelp Open Dataset

The Yelp dataset is a subset of our businesses, reviews, and user data for use in personal, educational, and academic purposes. Available as JSON files, use it to teach students about databases, to learn NLP, or for sample production data while you learn how to make mobile apps.

We only used businesses that are restaurants.

business_id	name	neighborhood	address	city	state	postal_code	latitude	lo
FYWN1wneV18bWNgQj...	"Dental by Design"	null	"4855 E Warner Rd...	Ahwatukee	AZ	85044	33.3306902	-111.
He-G7vWjzVUysIKrf...	"Stephen Szabo Sa...	null	"3101 Washington Rd"	McMurray	PA	15317	40.2916853	-80.
KQPW8lFf1y5BT2Mxi...	"Western Motor Ve...	null	"6025 N 27th Ave,...	Phoenix	AZ	85017	33.5249025	-112.
8DShNS-LuFqpEWIp0...	"Sports Authority"	null	"5000 Arizona Mil...	Tempe	AZ	85282	33.3831468	-111.
PfOCPjBrlQAnz__NX...	"Brick House Tave...	null	"581 Howe Ave"	Cuyahoga Falls	OH	44221	41.1195346	-81.
o9eMRCWt5PkpLDE0g...	"Messina"	null	"Richterstr. 11"	Stuttgart	BW	70567	48.7272	
kCoE3jvEtg6UVz5SO...	"BDJ Realty"	Summerlin	"2620 Regatta Dr,...	Las Vegas	NV	89128	36.20743	-11
OD2hnuuTJI9uotcKy...	"Soccer Zone"	null	"7240 W Lake Mead...	Las Vegas	NV	89128	36.1974844	-115.
EsMcGiZaQuG1OOvL9...	"Any Given Sundae"	null	"2612 Brandt Scho...	Wexford	PA	15090	40.6151022445	-80.091
TGWhGNusxyMaA4kQV...	"Detailing Gone M...	null	""	Henderson	NV	89014	36.0558252127	-115.0
XOSRcvtaKc_Q5H1SA...	"East Coast Coffee"	null	"737 West Pike St"	Houston	PA	15342	40.2415480142	-80.212
Y0eMNa5C-YU1RQOZf...	"CubeSmart Self S...	null	"2414 South Gilbe...	Chandler	AZ	85286	33.2717201	-111.
xcgFnd-MwkZeO5G2H...	"T & T Bakery and...	Markham Village	"35 Main Street N"	Markham	ON	L3P 1X3	43.8751774	-79.
NmZtoE3v8RdSJEczY...	"Complete Dental ...	null	"107 Whitaker Str"	Homestead	PA	15120	40.4014882	-79.
fNMVV_ZX7CJSDWQGd...	"Showmars Governm...	Uptown	"600 E 4th St"	Charlotte	NC	28202	35.2216474	-80.
l09JfMeQ6ynYs5MCJ...	"Alize Catering"	Yonge and Eglinton	"2459 Yonge St"	Toronto	ON	M4P 2H6	43.7113993	-79.
IQSlT5jGE6CCDhSG0...	"T & Y Nail Spa"	null	"8411 W Thunderbi...	Peoria	AZ	85381	33.6086538	-112.
b2I2DXtZVnpUMCXp1...	"Meineke Car Care...	null	"2518 Ironwood Dr"	Sun Prairie	WI	53590	43.18508	-89
0FMKDOU8TJT1x87OK...	"Senior's Barber ...	null	"13375 W McDowell"	Goodyear	AZ	85395	33.463629	-112
Gu-xs3NIQTj3Mj2xY...	"Maxim Bakery & R...	null	"9665 Bayview Ave...	Richmond Hill	ON	L4C 9V4	43.8675648	-79.

Food.com Recipes and Interactions

This dataset consists of 180K+ recipes and 700K+ recipe reviews covering 18 years of user interactions and uploads on Food.com (formerly GeniusKitchen).

Food recipes including ingredients, instructions, and nutritional information and interactions include user reviews and ratings.

	name	id	minutes	contributor_id	submitted	tags	nutrition	n_steps	steps	description	ingredients	n_ingredients
0	arriba baked winter squash mexican style	137739	55	47892	2005-09-16	['60-minutes-or-less', 'time-to-make', 'course...	[51.5, 0.0, 13.0, 0.0, 2.0, 0.0, 4.0]	11	['make a choice and proceed with recipe', 'dep...	autumn is my favorite time of year to cook! th...	['winter squash', 'mexican seasoning', 'mixed ...	7
1	a bit different breakfast pizza	31490	30	26278	2002-06-17	['30-minutes-or-less', 'time-to-make', 'course...	[173.4, 18.0, 0.0, 17.0, 22.0, 35.0, 1.0]	9	['preheat oven to 425 degrees f', 'press dough...	this recipe calls for the crust to be prebaked...	['prepared pizza crust', 'sausage patty', 'egg...	6

DATA CLEANING AND INGESTION

We used Jupyter Notebooks and Python to clean our data. Specifically, we used Numpy, Pandas, and Pyspark to pull the data. Many of the attributes including recipe ingredients, steps, and tags in the Food.com dataset and the business tags attribute in the yelp dataset were stored as lists in the dataframe. To convert these into a viable SQL schema, we had to explode the attribute and create a separate table representing the attribute. This caused us to have many tables in our database schema.

We also wanted to represent both the recipe and restaurant reviews using the same table relation and so we transformed them into a common schema and concatenated them. Then we needed to acquire restaurant menu information which was not present in our data, so we built a yelp scraper to gather the names of dishes served at the restaurants in our data. We performed entity resolution between recipes and dishes using a jaccard join on their names to join the two datasets together. Finally we downloaded the data as csv files and uploaded them to our database.

WEB SCRAPING

The Yelp data did not contain menu information which was important for the restaurant profile pages and the entity resolution between dishes and recipes.

So, we built a web scraper which iterated over the restaurants in the yelp dataset, found the link to the corresponding yelp business page, and extracted menu information using python requests, xpath, string manipulation, and parsing of json objects.

The results of this can be found in the **`Dishes`** and **`ServedAt`** relations.

ENTITY RESOLUTION

After scraping the dish information, we performed entity resolution between the dishes and recipes using jaccard similarity between the name strings.

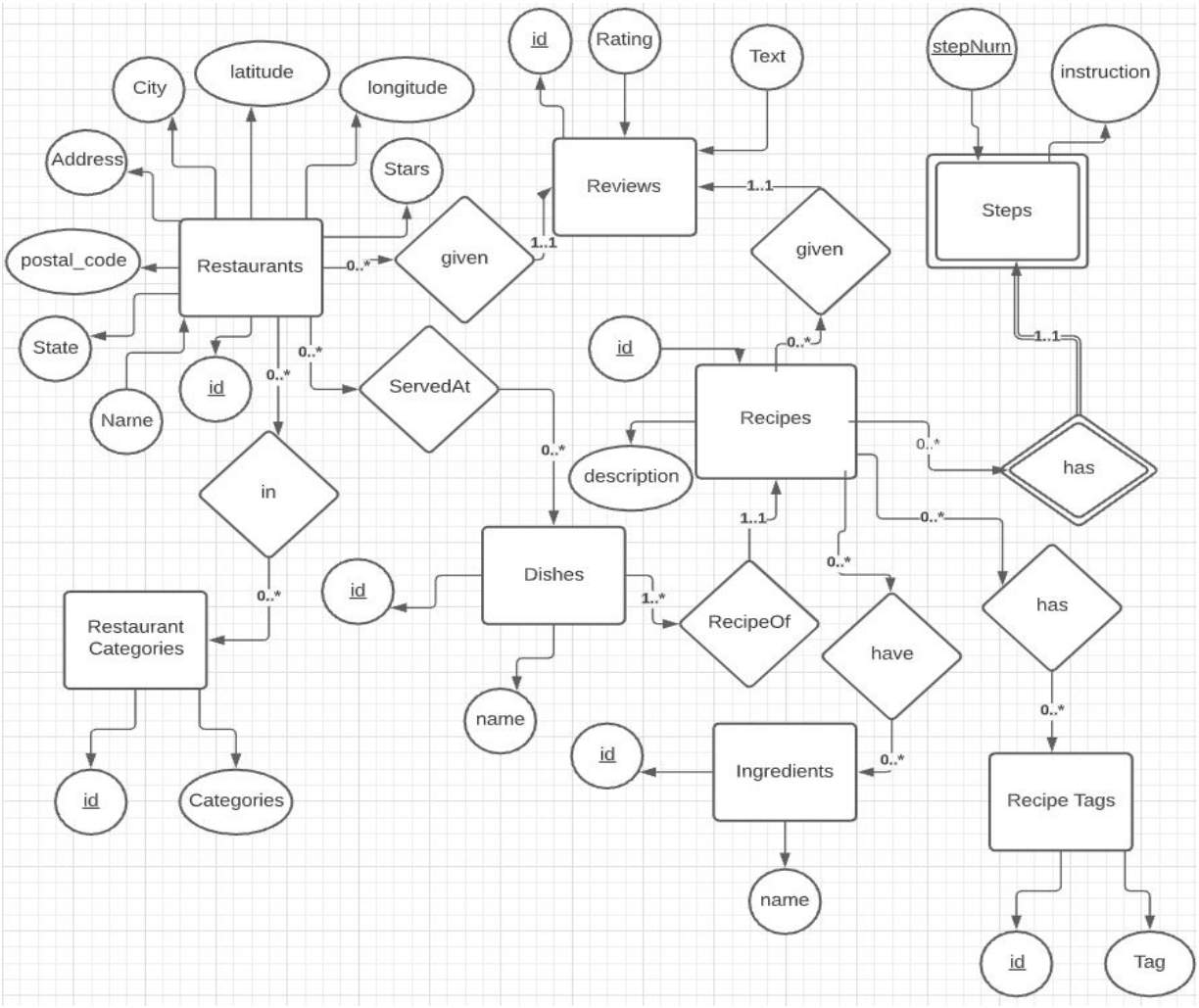
Jaccard similarity is defined as

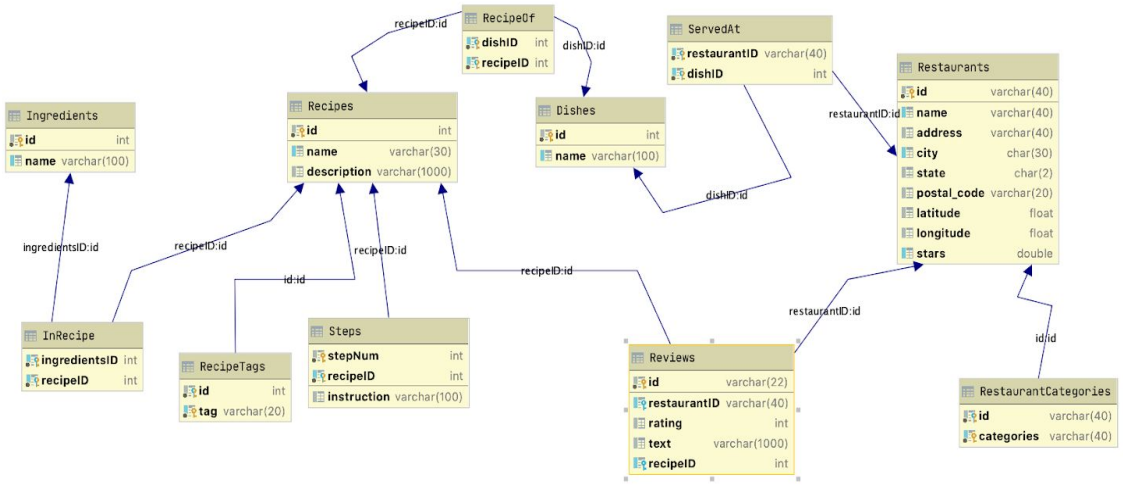
$$\frac{|\ M \cap N\ |}{|\ M \cup N\ |}$$

where M and N are the sets of n-grams for two strings.

We used an n-gram size of 5 with padding and a similarity threshold of .4 and performed a join between recipes and dishes where the similarity exceeded that threshold.
We did this using the py_stringmatching and py_stringsimjoin Python packages. The result of this can be found in the **`RecipeOf`** relation.

ENTITY-RELATIONSHIP DIAGRAM





RELATIONAL SCHEMA

Number of instances in each table:

- Dishes: 27,206
- Ingredients: 14,917
- InRecipe: 1,964,109
- RecipeOf: 78,659
- Recipes: 231,623
- RecipeTags: 4,036,485
- RestaurantCategories: 111,016
- Restaurants: 40,386
- Reviews: 2,340,957
- ServedAt: 56,734
- Steps: 2,203,703

NORMAL FORM AND JUSTIFICATION

The database was designed to reduce redundancy and so that each relation contains only keys and attributes that directly describe its key. The database is in BCNF with the exception of the location attributes in the Restaurants relation-technically, all the location information can be inferred from just the latitude and longitude. However, we made a decision to keep all the location information in the Restaurants table because we conceptualized all the location-related columns as one “location” attribute.

QUERIES

1. The following query obtains the id and name for dishes served at a given restaurant. We use this to display the dishes in the restaurant profile page.

```
select d.name dishName, d.id dishId from Restaurants r
 join ServedAt sa on r.id = sa.restaurantID
 join Dishes d on d.id = sa.dishID
 where r.id = '${restId}';
```

2. This query gets the id and name of the top ten restaurants that serve a given recipe. We use this query to generate links to relevant restaurants from the recipe

profile page.

```
SELECT DISTINCT s.restaurantID, res.name
FROM RecipeOf r JOIN ServedAt s ON r.dishID = s.dishID
INNER JOIN Restaurants res on res.id = s.restaurantID
WHERE r.recipeID = '${recipeId}'
ORDER BY res.stars DESC, RAND ()
LIMIT 10;
```

3. This query retrieves information for recipes relevant to a given dish. This allows us to provide recipe information and links underneath restaurant menu options.

```
select d.id dishId, r.id recipeId, r.name recipeName, r.description
recipeDescription from Dishes d
     join RecipeOf ro on ro.dishID = d.id
     join Recipes r on r.id  = ro.recipeID
     where d.id in (${dishidsstr})
```

4. This query produces the 1000 recipes which are most commonly found in recipes, in decreasing order of prevalence. This is used to create the ingredient scroll-down bar in the recipe dashboard page.

```
select i.name from
 Ingredients i join InRecipe r on r.ingredientsID=i.id
 group by i.name
 order by count(recipeID) desc
 limit 1000;
```

5. This query finds all the recipes that contain all of the ingredients in a list. It also filters for recipes that have associated dishes to link to using the “WHERE id in (select recipeID from RecipeOf)” clause. This query is used to generate search results from the ingredient list in the recipe dashboard page.

```
WITH ids AS (
     SELECT r.id
     FROM Recipes r
     JOIN InRecipe i ON i.recipeID = r.id
     JOIN Ingredients n ON i.ingredientsID = n.id
     WHERE n.name in ${ingredients}
     GROUP BY r.id
     HAVING COUNT(n.id) = ${ingredientsArray.length}
   )
```

```
   SELECT *
   FROM Recipes NATURAL JOIN ids
   WHERE id in (select recipeID from RecipeOf)
   order by RAND () limit 20;
```

6. This query returns restaurants (in decreasing order of rating) that are in a specified city and contain a specified tag. It also filters for restaurants that have a menu by using the “WHERE id in (select restaurantID from ServedAt)” clause. This query was used to generate filtered results in the restaurant dashboard page.

```
 SELECT Restaurants.id, name, city, stars
 FROM Restaurants
 JOIN RestaurantCategories ON Restaurants.id = RestaurantCategories.id
 WHERE Restaurants.city = '${city}'
 and RestaurantCategories.categories= '${tag}'
and Restaurants.id in (select restaurantID from ServedAt)
order by stars desc, RAND () limit 20;
```

7. This query finds all the recipes that contain a specified tag. It also filters for recipes that have associated dishes to link to using the “WHERE id in (select recipeID from RecipeOf)” clause. This query is used to generate search results from the tag drop-down in the recipe dashboard page.

```
SELECT id, name, description
   FROM Recipes
   NATURAL JOIN RecipeTags
   where tag = '${tag}'
and id in (select recipeID from RecipeOf)
order by RAND () limit 20;
```

PERFORMANCE EVALUATION

We will speed up our databases by adding indices to columns which are used for ordering, matching, and joining. By adding indices (which are B+ Trees in MySQL), the database maintains an ordered tree which can be parsed in O(log n) time instead of a full scan requiring O(n). Because this application only reads from the database and does not write, indices provide great speedups without the additional overhead of increased insertion times. Also, in considering which queries to add, it is important to note that MySQL automatically adds indices to primary and foreign keys and so we did not have to

consider these cases.

Furthermore, we added a materialized view to query 4 because (1) after adding indices to the database it was the slowest query (with a time of close to a full second) and (2) the query is frequently used and not dependent on any values so a materialized view would help a lot. This did in fact greatly improve retrieval time (by about a factor of 5) even after putting in the index..

Query 2:

Before adding indices: 222 ms (execution: 110 ms, fetching: 112 ms)

We can add an index on Restaurant.name to speed up duplicate elimination and add an index on Restaurant.stars to speed up reordering.

After adding index:
6 rows retrieved starting from 1 in 125 ms (execution: 53 ms, fetching: 72 ms)

Query 4:

Before adding indices: 402 ms (execution: 1 s 315 ms, fetching: 87 ms)

We can add an index to `Ingredient.name` to speed up the grouping.

After adding indices: 813 ms (execution: 742 ms, fetching: 71 ms)

We will now add a materialized view called 'MaterializedViewIngredientsDropdown' in the database (which is just a table in MySQL) which is simply the result of the query saved as a table:

```
CREATE TABLE MaterializedViewIngredientsDropdown AS
SELECT i.name
FROM Ingredients i JOIN InRecipe r ON r.ingredientsID=i.id
GROUP BY i.name
ORDER BY count(recipeID) DESC
LIMIT 1000;
```

Then query 4 just becomes:

```
SELECT name FROM MaterializedViewIngredientsDropdown;
```

After adding materialized view: 122 ms (execution: 66 ms, fetching: 56 ms)

Query 5:

Before adding indices: 8 s 780 ms (execution: 8 s 670 ms, fetching: 110 ms)
This query will benefit from the `Ingredient.name` index implemented for query 4.

After adding indices: 570 ms (execution: 331 ms, fetching: 239 ms)

Query 6:

Before adding indices: 204 ms (execution: 83 ms, fetching: 121 ms)

We will add an index on `Restaurants.city` and `RestaurantCategories.categories` to speed up matching. Furthermore, the query will benefit from the index added to `Restaurants.stars` for query 2.

After adding indices: 137 ms (execution: 58 ms, fetching: 79 ms)

Query 7:

Before adding indices: 287 ms (execution: 219 ms, fetching: 68 ms)

We will add a hash index to `RecipeTags.tag` to speed up the matching.

After adding indices: 210 ms (execution: 85 ms, fetching: 125 ms)

TECHNICAL CHALLENGES AND HOW THEY WERE OVERCOME

We faced three significant technical challenges:

1. **Size:** The datasets were quite large-the reviews alone were several gigabytes. Because of this, the data did not fit on ram and so had to be manipulated through spark and kept crashing python, the jupyter notebook, and our machines. Furthermore, uploading the data to the database was difficult and had many errors. Because of this, we had to think hard about good design both for our database and for our data processing algorithms to make them computationally feasible (e.g. vectorized functions, multiple stages in a single pass, etc).

2. **Web Scraping:** Web scraping ended up being significantly harder than

expected. We thought that we would be able to find a usable web api or easy way to scrape all the menu data that was a core part of our initial website designs. However, web menu api's were prohibitively expensive and websites that collated menu data were often disorganized and inconsistent. We ended up scraping menu data from Yelp using python requests and xpath to navigate their html. However, we found that their menu data was often irregular, missing, or inconsistent. Furthermore, they would often block our IP addresses and so we had to rotate proxies and user-agents which took a lot of time. Then, after the web scraper was working for a few days, they changed their html format and we had to rebuild the parsing component almost from scratch. Even after all that difficulty, yelp only had about 25% of the 50,000 restaurant menus we attempted to scrape and for those, yelp only displayed 4-5 menu items.

3. **Entity Resolution:** After scraping the required menu data from yelp, we needed to link recipes to dishes. We had planned to match recipe ingredients to dish ingredients in order to do this. However, due to the difficulty acquiring menu data, we were unable to get the ingredients in each dish - we only had the name of the dish. Because we had to link recipes and dishes only using dish names, we decided to use Jaccard similarity of strings to identify dishes and recipes which were similar. This required learning some new python packages and fiddling with the hyperparameters like the similarity threshold and n-gram size, but we ended up finding a combination which was able to identify relevant similarities.

EXTRA CREDIT FEATURES

Web scraping: we are hoping for extra credit points for the web scraping as it was difficult, time consuming, and used a lot of external tools and knowledge like python requests, xpath, and a more in-depth understanding of http requests including headers, user-agents, and proxies.

Appendix: DDL

```
create table Dishes
(
	id int not null
		primary key,
	name varchar(100) null
);

create index Dishes_name_index
	on Dishes (name);

create table Ingredients
(
	id int not null
		primary key,
	name varchar(100) null
);

create index Ingredients_name_index
	on Ingredients (name);

create table Recipes
(
```

```
	id int not null
		primary key,
	name varchar(30) null,
	description varchar(1000) null
);

create table InRecipe
(
	ingredientsID int not null,
	recipeID int not null,
	primary key (ingredientsID, recipeID),
	constraint ingr_recipe
		foreign key (ingredientsID) references Ingredients (id),
	constraint recipes_id
		foreign key (recipeID) references Recipes (id)
);

create index InRecipe_ingredients_id_index
	on InRecipe (ingredientsID);

create index InRecipe_ingredientsid_index
	on InRecipe (recipeID);

create table RecipeOf
```

```
(
	dishID int not null,
	recipeID int not null,
	primary key (dishID, recipeID),
	constraint RecipeOf_ibfk_1
		foreign key (recipeID) references Recipes (id),
	constraint RecipeOf_ibfk_2
		foreign key (dishID) references Dishes (id)
);

create index RecipeOf_dishID_index
	on RecipeOf (dishID);

create index recipeID
	on RecipeOf (recipeID);

create table RecipeTags
(
	id int not null,
	tag varchar(20) not null,
	primary key (id, tag),
	constraint RecipeTags_ibfk_1
		foreign key (id) references Recipes (id)
);
```

```
create index RecipesTags_tag_index
        on RecipeTags (tag);

create table Restaurants
(
        id varchar(40) not null
                primary key,
        name varchar(40) null,
        address varchar(40) null,
        city char(30) null,
        state char(2) null,
        postal_code varchar(20) null,
        latitude float null,
        longitude float null,
        stars double null
);

create table RestaurantCategories
(
        id varchar(40) not null,
        categories varchar(40) not null,
        primary key (id, categories),
        constraint RestaurantCategories_ibfk_1
```

```
                foreign key (id) references Restaurants (id)
);

create index RestaurantsCat_cat_index
        on RestaurantCategories (categories);

create index RestaurantsCategories_index
        on RestaurantCategories (categories);

create index RestaurantsCat_stars_index
        on Restaurants (stars);

create index Restaurants_Cities_index
        on Restaurants (city);

create index Restaurants_city_index
        on Restaurants (city);

create index Restaurants_name_btree_index
        on Restaurants (name);

create index Restaurants_name_index
        on Restaurants (name);
```

```
create table Reviews
(
	id varchar(22) not null
		primary key,
	restaurantID varchar(40) null,
	rating int null,
	text varchar(1000) null,
	recipeID int null,
	constraint Reviews_ibfk_1
		foreign key (restaurantID) references Restaurants (id),
	constraint Reviews_ibfk_2
		foreign key (recipeID) references Recipes (id)
);

create index Recipes_reviews_index
	on Reviews (recipeID);

create index Restaurants_reviews_index
	on Reviews (restaurantID);

create table ServedAt
(
	restaurantID varchar(40) not null,
	dishID int not null,
```

```
	primary key (restaurantID, dishID),
	constraint ServedAt_ibfk_1
		foreign key (dishID) references Dishes (id),
	constraint rest_servedat
		foreign key (restaurantID) references Restaurants (id)
);

create index ServedAt_restid_index
	on ServedAt (restaurantID);

create index dish_servedat
	on ServedAt (dishID);

create table Steps
(
	stepNum int not null,
	recipeID int not null,
	instruction varchar(100) null,
	primary key (stepNum, recipeID),
	constraint step_recipe
		foreign key (recipeID) references Recipes (id)
);
```