Michelin Starred Yelp Restaurants

CMPE 138: Database Systems I

Section 01

Dr. Davic C. Anastasiu

Group 6

Alex Richards | Yu Jung | Xiao Yi

Tuesday, May 9th, 2017

Introduction

Web applications, like Yelp, are the most common path to get information and reviews for restaurants. However, it is not easy to extract high-end restaurants, like Micheline restaurants, from Yelp. Therefore, the purpose of this project is to develop an application which is specific to Micheline restaurants. To improve the user's experience, a dataset of Micheline restaurants cross-referenced from Yelp data retrieved from Yelp's API was used to build a database. Users can easily get information from the web application and connect to the Yelp page for reviews.

In this project, Yelp's data were analyzed and the data were used to design a database schema that allows end-users to retrieve information about Michelin restaurants. The user interface was designed to be very simple and similar to the one provided by Yelp. The web application interface can provide end-users with the ability to search for restaurants in a or state, as well as by name or the category of food that the restaurant serves.

Database Design

1. External Interfaces
    - Yelp API
        - Used Python to access the Yelp Fusion API 3.0 and build Micheline restaurants database relational tables. Business queries were made for cities that were observed in the Michelin dataset, then review queries were made for the buisness IDs in the filtered subset of the business queries.
    - Michelin restaurants data
        - The restaurants' data were collected from viamichelin.com web page using a Python script that scrapes the restaurant name and location from multiple pages of results returned from a search query. These same queries were used also to query the Yelp API.

2. Software Requirements
    - User-Interface
    - Backend model that contains all of the tables: restaurant, review, category, address, and states.
    - Middleware can take user's input as a request to the server and query database for filtered results.
    - Render a view for search results that offer the ability to further filter by either category, city, or state.
    - Render a view for an individual restaurant's data.

3. Database

The dataset, which is sourced from Yelp, was designed for our own database schema for this project. We provide an API that can be utilized to create a customized client-facing interface. The dataset contains the following entity tables:

```
%sql SELECT type, name, sql FROM sqlite_master;
```

Done.

type	name	sql
table	category	CREATE TABLE category (id INT PRIMARY KEY, title VARCHAR(100))
index	sqlite_autoindex_category_1	None
table	restaurant_by_category	CREATE TABLE restaurant_by_category (restaurant_id INT, category_id INT, FOREIGN KEY (restaurant_id) REFERENCES restaurant, FOREIGN KEY (category_id) REFERENCES category)
table	review	CREATE TABLE review (url VARCHAR(300) PRIMARY KEY, restaurant_id INT, rating FLOAT, name VARCHAR(100), time VARCHAR(100), text VARCHAR(300))
index	sqlite_autoindex_review_1	None
table	sqlite_sequence	CREATE TABLE sqlite_sequence(name,seq)
table	address	CREATE TABLE address(addressID INTEGER PRIMARY KEY AUTOINCREMENT, street VARCHAR(100), city VARCAAR(100), zip_code INT)
table	restaurant	CREATE TABLE restaurant(id INTEGER PRIMARY KEY AUTOINCREMENT, name VARCHAR(100) NOT NULL, rating FLOAT, url VARCHAR(300), price VARCHAR(4), review_count INT, phone VARCHAR(20), image_url VARCHAR(300), addressID INT)
table	state	CREATE TABLE state(stateID INT, stateAbbrivate VARCHAR(2), statename VARCHAR(20))
table	belong_to	CREATE TABLE belong_to(zip_code INT, state_id INT, FOREIGN KEY(state_id) REFERENCES state ON DELETE CASCADE)

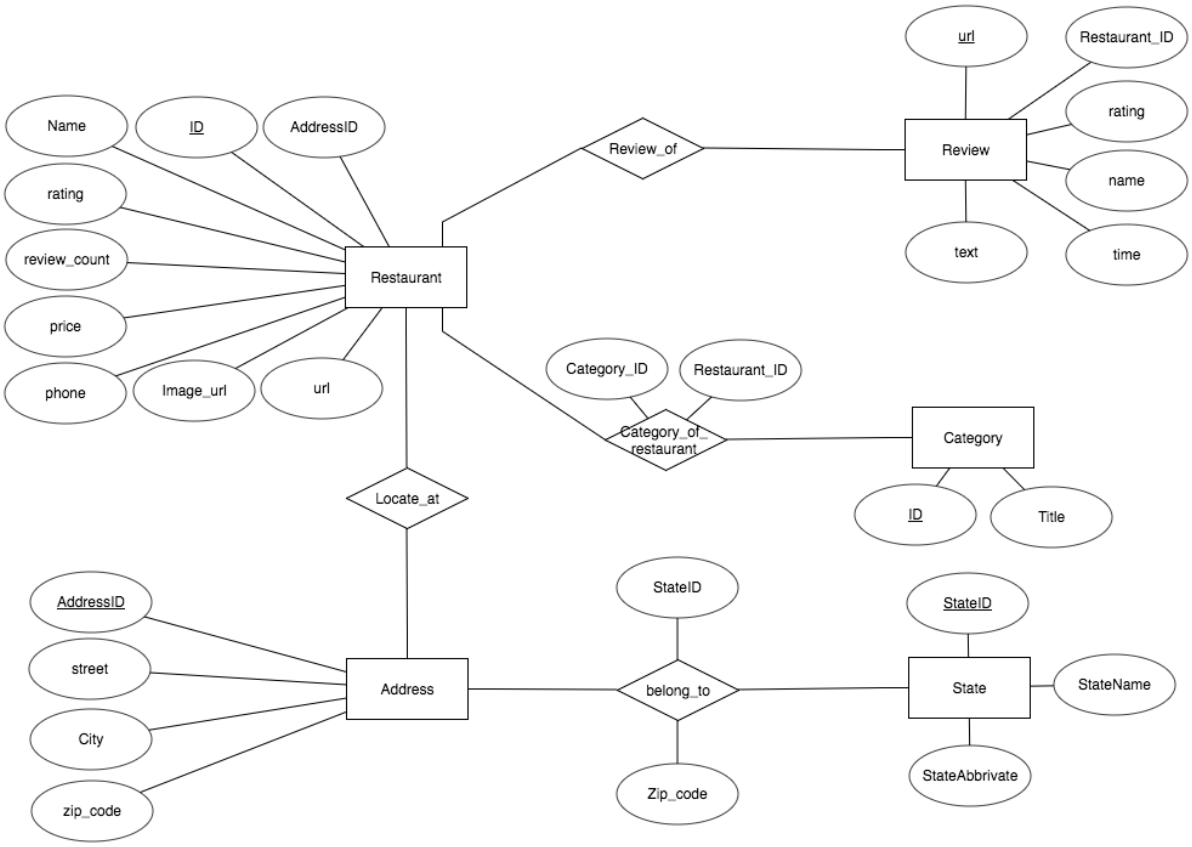
- Restaurant (id : integer, name: string, rating : number, url : string, price: string, review_count : integer, phone : string, image_url: string, addressID: integer)

    - There are 384 Michelin restaurants.
    - The primary key is id and set with auto increment so the id can generate a new id number for each new record insert.
- Address (addressID : integer, street : string, city : string, zip_code: integer)
    - All Michelin restaurants located at 13 cities.
    - The primary key is addressID and set with auto increment so the addressID can generate a new ID number for each new record insert.
- State (stateID : integer, stateAbbrivate : string, statename : string)
    - Only consider Michelin restaurants within three states: Illinois, New York, California.
    - The primary key is stateID.
- Review (url : string, restaurant_id : integer, rating : number, name : string, time: string, text : string)
    - Collect 1152 reviews record from Yelp.
    - The primary key is url
- Category (id : integer, title : string)
    - There are 96 different of categories (types of food).
    - The primary key is id.

Relational table

- Restaurant_by category (restaurant_id : integer, category_id : integer)
    - The foreign keys are restaurant id and category id.
- Belong_to (zip_code : integer, state_id : integer)
    - The foregin key is state_id and set with “On Delete Cascade”.

4. ER-Diagram



Web Application

The web service was built using Python's Django web framework. To operate the web application, care must be taken to install an up-to-date version of Django (our project uses Django 1.10.6) and all of the Django extensions. The application is configured to use an SQLite3 database on the backend. The frontend utilizes Django's built-in Jinja2 templating engine. In addition to the application root URL used for searching the database, Django include an admin view for manually entering records which was extremely useful for testing purposes.

What follows is a typical use-case of our developed web application. The user reaches the landing page via the root URL of the application where they are presented with a simple search bar. The search bar allows users to search for partial phrases and input is case-insensitive. Below the search bar are four headings which are populated with relevant information upon the user submitting a request to the server through the search bar's input. These four headings are for restaurants, categories, cities, and states respectively. Originally, the search bar's input was supposed to be queued upon key presses to perform asynchronous AJAX requests to the database so

that suggestions would populate the appropriate headings in realtime without making any HTTP requests. Unfortunately, time constraints and confusion about how to use Django's selectize widget, which apparently realized this functionality, prohibited this feature's implementation. Clicking on any of the divisions generated for each response item will trigger an onClick redirect event. If the item clicked is not, itself, a restaurant, then a secondary view is rendered that displays all the restaurants that match that clicked item on the attribute corresponding with the header that the clicked item falls under. From this view, clicking on a restaurant name performs the same redirect as if the use has clicked on a restaurant name in the landing page view. This redirect takes the user to a view generated for that restaurant instance of the database and includes the restaurant's name, price range, rating, number of reviews, phone number, address, as well as an image and sampling of review items (the Yelp API restricts the amount of reviews and the amount of content per review fairly heavily). Because the Yelp API provides a URL link the the Yelp page for each restaurant, this link is also included in the restaurant view so that users can find the full amount of information on that restaurant from the source.