Team Geek Squad
CIS 550 Spring 2022
Final Project Submission
Summary of Yelp Restaurant Data for Diners & Business Owners

Hoyoung Jang
Hyung Gyu Park
Nick Holt

Github Link : https://github.com/hoyoung-jang/TeamGeekSquad_CIS550/
Google Docs Link : https://docs.google.com/document/d/1LngPOMc3qSVr3rfIKG5uAKbHVCITCis4qG_hziZ9uSc/edit?usp=sharing
Google Drive Link : https://drive.google.com/drive/u/1/folders/0ALvHqAnJnMn4Uk9PVA

1. Introduction

The past two years have severely impacted restaurants around the world and often led to expectation disconnects between the owners and their customers. Our application aims to summarize the impact of COVID-related business additions on restaurant ratings and create an intuitive interface for users to use precise restaurant description metrics to identify their ideal restaurants.

Our team wanted to choose a data source relevant to current events that struck a good balance between complexity and information quality, with the potential to answer questions from multiple perspectives. We also deliberately opted for datasets outside of our collective area of expertise (finance) to expand our breadth of experience and problem-solving capabilities.

2. Architecture

Technologies Used:

1. Main DBMS
    a. MySQL : We will use MySQL database to control and query the given dataset for this final project.
    b. DataGrip : DataGrip will be used for browsing database and testing queries.
2. Python : Python will be used for revising dataset and simple operations.
    a. Pandas : Pandas library will be used for mainly pre-processing and cleansing the given dataset.
    b. Google Colab : Google tools for sharing and editing ipynb files to share the process of pre-processing and cleansing the given dataset.
3. JavaScript : JavaScript will be used for both server-side and client-side. JavaScript will be used as the main language for both sides in pursuit of concurrency .
    a. Node.js : Server-sided back-end runtime library.
        i. NPM(Node Package Manager) : Version control and package management library for Node.js.
        ii. Express.js : Simple and fast web framework library.
    b. React : Front-end library for making the user interface.

4. HTML/CSS Template: We repurposed code from the HW2 FIFA website, using the general structure and format as a template for our project.

3. Data & Database

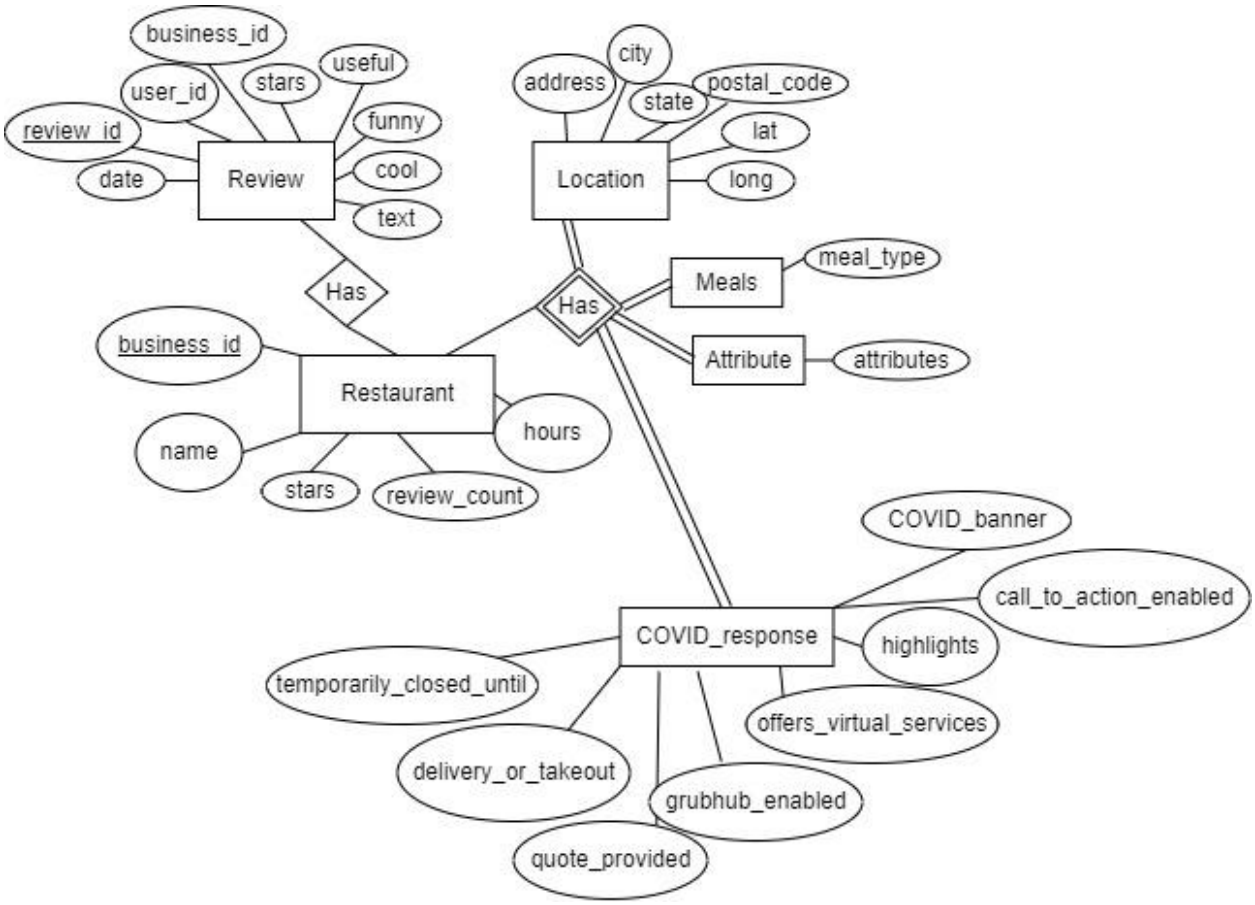
Yelp Dataset: https://www.yelp.com/dataset/download
The dataset consists of a tarball of 5 separate data files containing all of the business review data of Yelp.com. Of these, we accessed business.json and review.json, which were loaded as distinct databases and joined as necessary to produce our outputs. We used Python code to upload the whole JSON datasets.

In its raw format, business.json consisted of 14 primary attributes, with additional nested attributes wrapped into two additional sub-levels. Fully unwound, this ended up totalling 61 attributes and 11GB of data to parse and import. To increase the speed of the application, we removed all but the 5 most important fields and ended up with 150,300 tuples after loading and cleaning the data, while at the sub-level we broke key information into the tables Meals, Location, and Attribute to be included only when specifically needed by their queries.

Review.json was smaller, with 9 attributes and no nesting initially, for a raw total of 6.46GB. The bulk Yelp dataset was too heavy to upload, so we had to split the data into more than 50 JSON files in order to upload to our AWS DB. All rows were imported, with 3,800,000 rows remaining after cleaning for a net size of 2.3GB. The majority of the rows removed were reviews pertaining to non-restaurant businesses.

COVID_response.json was a separate dataset provided by Yelp as supplementary information to capture the unique business circumstances of 2020+. Although the source was different, it could be mapped with the original Yelp dataset we found with the same business_id. After uploading the whole dataset, we had to cleanse the data in order to ensure that all data were formatted correctly. We dropped some null columns, and changed the value(“False” and null and 0) to “No Comment Regarding COVID” for COVID_banner column.

ER Diagram:



Our normalized schema is:

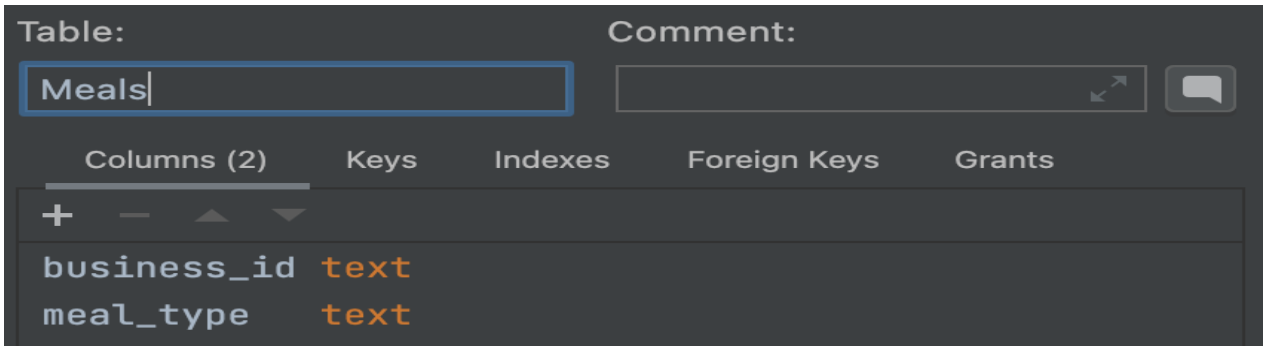
Restaurant(business_id, name, stars, review_count, hours)
Location(business_id, address, city, state, postal_code, lat, lon)
Attribute(business_id, attributes)
COVID_Response(COVID_business_id, COVID_banner)`
Meals(business_id, meal_type)
Review(review_id,user_id, businsess_id, stars, useful, funny, cool, text, date)

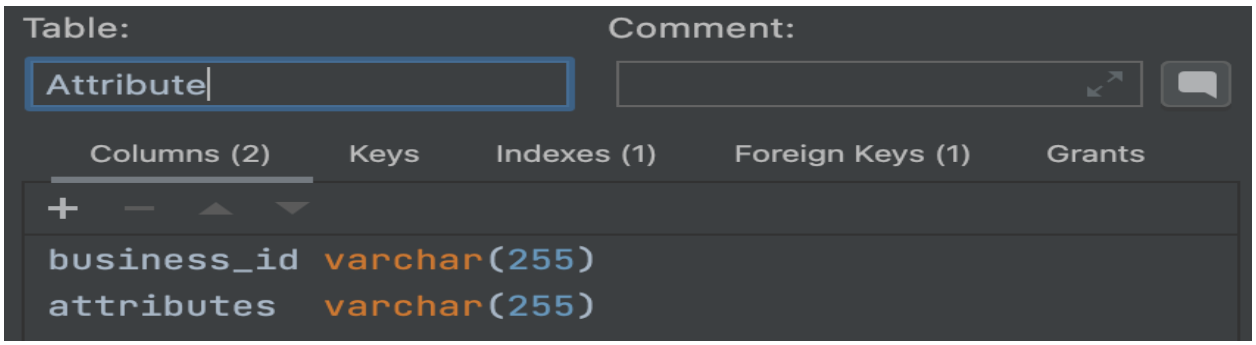
We split the datasets into different tables in order to remove the functional dependencies. Thus, for the CustomerPage in our app, we had to join 5 tables to extract the data which satisfy the attributes. There is no transitive dependency for all the tables except the Location table. Postal_code in the Location table can define the city and state, however, for the efficiency of table usage and lack of need for division of small size dataset, we left it as 3NF. The rest of the tables follow BCNF since if P→Q, the P(business_id) is the super key.

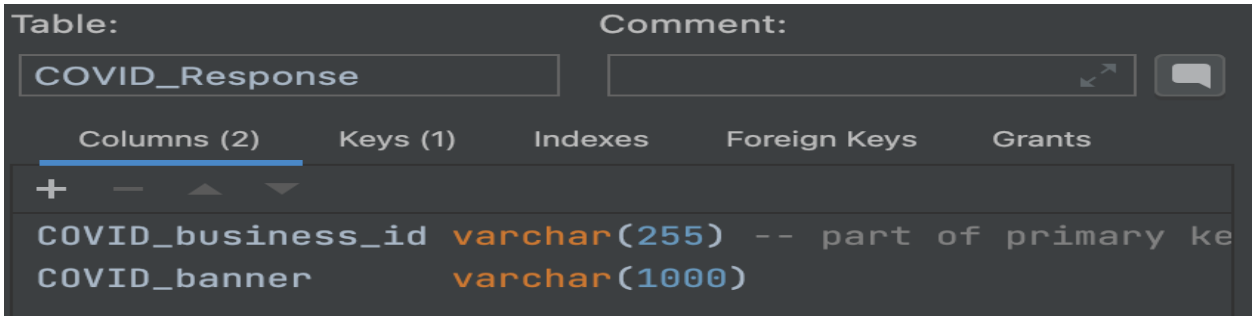
AWS MySQL Database is:

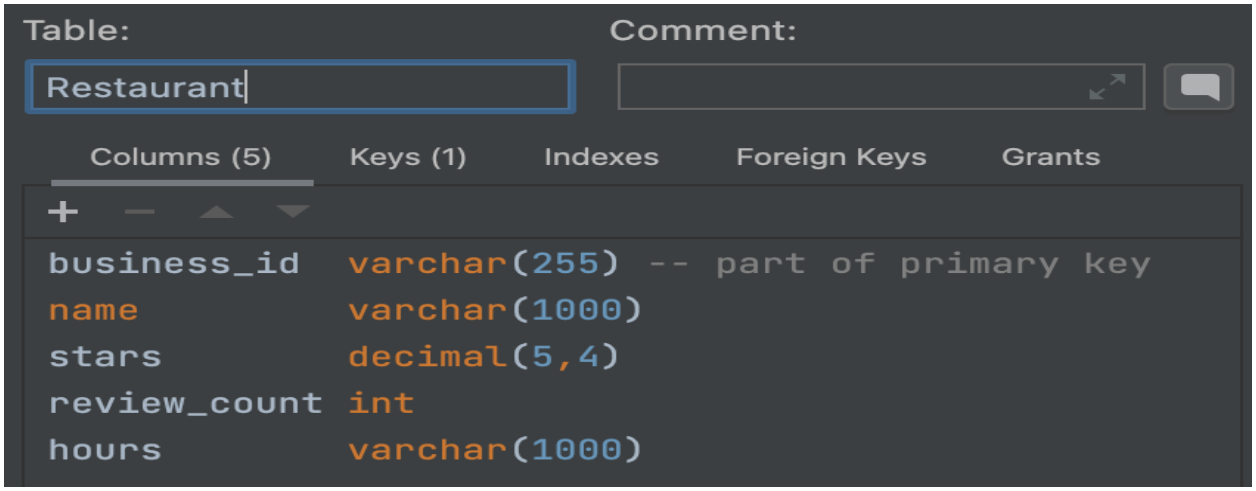
host : final.cwsn1nqds1jf.us-east-1.rds.amazonaws.com
port : 3306
ID : TeamGeekSquad
PW : WithoutOne

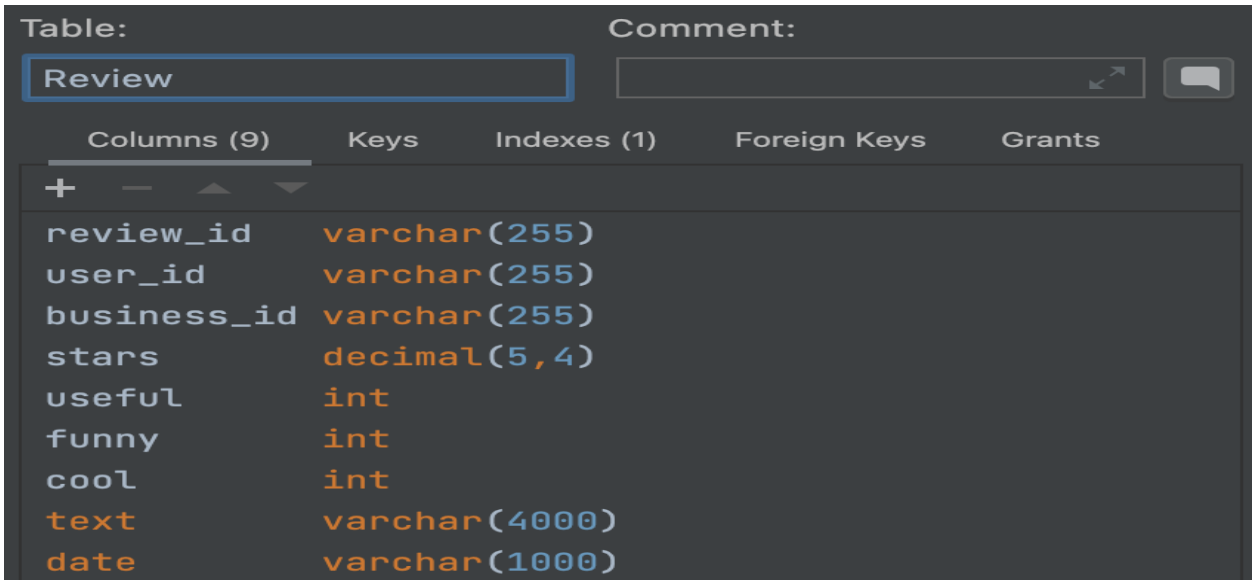
Table Schema:











4. Web App description

The application has three pages: HomePage, CustomerPage, and OwnerPage.

- HomePage is for overall users. It shows the whole list of restaurants data and current COVID policy of each restaurant.

- CustomerPage is for diners. It allows customers to search restaurants with certain attributes such as location, bike, and delivery and recent reviews of the specific restaurant.
- OwnerPage is for restaurant owners. It helps owners to analyze the current restaurant situation from various perspectives. Owners can see an analysis of a restaurant in a radar chart so that an owner can improve the quality of the restaurant. Also, by overlapping the data from other restaurants in the same postal code, they can do a competitor analysis as well.

The functionalities we implemented for each pages are as follows :

A. HomePage
   a. Get all lists of restaurants in our database
   b. If we click one of the restaurants we want to take a look, we can check what COVID policy the restaurant implements

B. CustomerPage
   a. Get a list of restaurants filtered with certain attributes
   b. If we click one of the restaurants we want to take a look, we can check the recent reviews of the restaurant

C. OwnerPage
   a. Get a list of restaurants filtered with certain attributes
   b. If we click one of the restaurants we want to take a look at, we can see the radar chart of the restaurant. The chart shows revisiting rate, review_count, stars, neighborhood density(number of restaurants nearby) and number of regular customers(customers with more than 2 reviews)
   c. Compare with other restaurants by analyzing two different radar charts at the same time to visualize competency of selected restaurant compared to target.

5. API Specification & Queries

A. `function getRestaurant (req, res)`

```
`SELECT a.*, b.*
  FROM Restaurant a, Location b
  WHERE 1=1
     and b.business_id = '${businessId}'
     and a.business_id = b.business_id;`
```

   ○ Get a restaurant data from Restaurant and Location by matching business_id

B. `function getAllRestaurants (req, res)`

```
`SELECT business_id,name,stars,review_count,hours
FROM Restaurant
LIMIT ${pagesize}*(${req.query.page}-1), ${pagesize}`
```

- Get all the restaurants data from Restaurants and show them on HomePage

C. function getRestaurantsByPostalCode (req, res)

```
`SELECT a.*, b.*
FROM Restaurant a, Location b
WHERE 1=1
        AND a.busin ess_id = b.business_id
        AND b.postal_code = ${postal_code}
ORDER by a.stars DESC;`
```

- Get a restaurant data from Restaurant and Location by matching business_id
- This is used to collect nearby restaurants for OwnerPage

D. function getRestaurantsByStateCity(req, res)

```
const basicQuery = req.query.mealType ?
  `SELECT R.business_id AS businessId, R.name, R.stars, R.review_count, L.state,
L.city, L.address, L.postal_code AS postalCode, L.lat, L.lon, A.attributes
  FROM Restaurant R, Attribute A, Location L, Meals M
  WHERE R.business_id = A.business_id
  AND R.business_id = L.business_id
  AND R.business_id = M.business_id`
  : `SELECT R.business_id AS businessID, R.name, R.stars, R.review_count, L.state,
L.city, L.address, L.postal_code AS postalCode, L.lat, L.lon, A.attributes
    FROM Restaurant R, Attribute A, Location L
    WHERE R.business_id = A.business_id
    AND R.business_id = L.business_id`

const stateQuery = req.query.state ? ` AND L.state LIKE '%${req.query.state}%'`: ``
const cityQuery = req.query.city ? ` AND L.city LIKE '%${req.query.city}%'`: ``
const starsHigh = req.query.starsHigh ? req.query.starsHigh : 5
const starsLow = req.query.starsLow ? req.query.starsLow : 0
const starsQuery = ` AND R.stars >= ${starsLow} AND R.stars <= ${starsHigh}`
```

```
const bikeParkingQuery = (req.query.bikeParking > 0) ? ` AND A.attributes LIKE
'%"BikeParking": "True"%'` : ``
const creditCardsQuery = (req.query.creditCards > 0) ? ` AND A.attributes LIKE
'%"BusinessAcceptsCreditCards": "True"%'` : ``
const deliveryQuery = (req.query.delivery > 0) ? ` AND A.attributes LIKE
'%"RestaurantsDelivery": "True"%'` : ``
const takeOutQuery = (req.query.takeOut > 0) ? ` AND A.attributes LIKE
'%"RestaurantsTakeOut": "True"%'` : ``
const mealTypeQuery = req.query.mealType ? ` AND M.meal_type LIKE
'%${req.query.mealType}%'` : ``
const pageQuery = (req.query.page && !isNaN(req.query.page))? `LIMIT ` +
pagesize*(req.query.page-1) + `, ${pagesize}` : ``

const query =
`${basicQuery}${stateQuery}${cityQuery}${starsQuery}${bikeParkingQuery}${creditCards
Query}${deliveryQuery}${takeOutQuery}${mealTypeQuery} ORDER BY stars DESC
${pageQuery};`
```

- Get restaurants data that satisfy certain attributes(i.e. State, city, star rating, meal type, bike, credit card, delivery, take out) and show them on CustomerPage.js so that a customer can look up a restaurant with the attributes he/she wants
- The most complex query in our web implemented on CustomerPage
- By using conditional statements, we can drop out unnecessary join if it is not needed
- Adopted left deep join for optimization
- By making query by building partial query blocks, we can optimize and reduce time on unused merge and search. Also we can avoid using multiple conditional statements.

E.
```
function getRevisitRate (req, res)
`WITH a AS (SELECT business_id, user_id, count(user_id) as visit_count
    FROM Review
    WHERE business_id = '${businessId}'
    GROUP BY user_id
    ORDER BY visit_count)

 SELECT a.business_id AS businessId, sum(a.visit_count) as totalCount,
```

```
    sum(a.visit_count)-count(a.business_id) as revisitCount,
    (sum(a.visit_count)-count(a.business_id))/sum(a.visit_count) as revisitRate
  FROM a;`
```

- Calculate the revisiting rate of a restaurant by calculating the number of users who went to the restaurant more than once and show them on OwnerPage so that an owner can analyze the current status of the restaurants
- Adopted CTE expression to speed up and optimize the query

F.
```
function getRegularCustomers (req, res)
`WITH customers AS (SELECT business_id, user_id, count(user_id) as visitCount
    FROM Review
    WHERE business_id = '${businessId}'
    GROUP BY user_id
    ORDER BY visitCount)
SELECT count(*) AS regularCustomers
FROM customers
WHERE visitCount >= ${number};`
```

- Calculate the number of users who reviewed more than the target number. They could be called regular customers or revisitors. Adopted CTE expression to speed up and optimize the query

G.
```
function getReviews(req, res)
`SELECT business_id AS businessId, date, stars, text AS review
FROM Review
WHERE business_id = '${businessId}'
ORDER BY date DESC`
```

- Get reviews of a certain restaurant that a client is searching and show them on the CustomerPage so that after looking up available list with the attributes he/she wants, a customer can take a look at the reviews of the restaurants

H.
```
function getCovidBanner (req, res)
```

```
`SELECT *
FROM COVID_Response
WHERE COVID_business_id = '${businessId}'
```

- Get the business_id and policies regarding COVID19 and show them on HomePage. If there is no comment, it shows “No Comment Regarding COVID”

6. Performance evaluation & Technical challenges

After optimization and indexing, we could not find any noticeable delay on any queries and the app was smooth except the one query on the HomePage showing the overall list of restaurants. It was a simple query looking up the whole dataset, so there was nothing we could do to enhance the speed of the app.

Optimization used for queries include using CTE expression for speeding up and adopting building partial query blocks for one final query so that we can avoid using multiple conditional statements and unnecessary joins.

Another technical modification we have made when we make a query for calculating the revisiting rate. We had to look up all the review data(2.3GB) linearly for the review counts. Without setting a primary key on the user_id, the query took 9s 780ms, but after setting a primary key on the user_id which we use for the group by function, the execution time of the query is reduced to 1s 174ms on average.

Our team ran into several challenges over the course of this project. Most significantly, the loss of our fourth team member due to illness greatly increased our workload and deprived us of our primary data evaluator. Additionally, one month into the project the majority of COVID restrictions eased in the United States, and Yelp removed their COVID-based dataset from their website. Technically speaking, the dataset also included a complicated series of nested documents that we needed to spend significant time troubleshooting in order to unwind and view properly. The team also largely lacked familiarity with Javascript and web programming at the outset of the project, increasing the learning curve at the outset.

One of the interesting features of our app is an in-depth analysis of two restaurants in the same city. It is important to analyze not only restaurant rating and customer experience itself, but the business environment and the performance of the competitors. Because of the lack of datasets we can analyze, the criteria of the radar chart were mostly focused on the reviews and number of restaurants. But for the further extension, we could implement the following features.

1) For Customer Page, a live twitter or instagram feed can be shown with the original yelp review dataset so that we can compare them with the realtime reviews data.
2) For Owner Page, if we can get income statements of each restaurant, we could analyze the financial status of each restaurant and cash efficiency of the restaurant by comparing with others. We could make an index with the financial data, we could build a radar chart in various angles(customer experience, degree of competition, financial soundness).

7. Division of Work

Hoyoung Jang
- Database Research
- MySQL queries
- Data upload to AWS
- Front-end development
- Testing
- Debugging

Hyung Gyu Park
- Database Research
- MySQL queries
- Data cleansing and database management
- Final Report
- Debugging

Nick Holt
- Admin & Coordinating
- Database Research
- MySQL queries
- ER Diagram
- Testing
- Final Report