Yelp Insight: A Comprehensive Platform for Customers, Business Owners, and Administrators

Ouyang Mingyu, Lin Liangtao, Li Zeyun, Tang Zheng, Zhou Yuhao

DSA 5104 Final Project
Group 24

Abstract. In this project, we proposed a comprehensive data platform, namely *Yelp Insight*, for customers, business owners, and administrators. The highlights of our project can be concluded as follows: 1. We added more than 6,000 entries of Yelp data using the customized web crawler, and 180,000 rows of features via large language model generation techniques. 2. We carefully explored the characteristics of our Yelp dataset and presented many interesting insights into Yelp. 3. We provide more than 100 querying methods via the combination of our 16 basic queries and flexible parameters, which could cover a wide range of demands from customers, business owners, and platform administrators. 4. We built a demo website that can provide data access services via a non-coding interface. The code and demo of our project will be available at: https://github.com/Linlt-leon/Yelp-Insight.

Table of Contents

1 Introduction

1.1 Project Overview

The objective of our proposed *YELP Insight* project is to dive into the methodologies of data management and retrieval, focusing on simulating the practical applications in real-world scenarios via the given YELP dataset [7]. This mainly involves a sequence of critical tasks: data collection, processing, storage, and access.

Data Collection: The initial collection step involves gathering relevant data from diverse sources. For further emphasizing comprehensiveness, in addition to the original dataset, we also: 1. crawled data from the current date yelp.com, 2. generated complicated data attributes via the large language models. This step forms the foundation upon which subsequent processes are built.

Data Processing: Once collected, the raw data undergoes some preprocessing operations. This transforms the massive data into a structured format, making it suitable for the following analysis and storage. Processing includes data cleaning, file splitting, and preparing the data for further use.

Data Storage: The data is then stored in our databases. We will explore two primary storage methods: traditional SQL databases (MySQL [2]) and NoSQL databases (MongoDB [1]). This dual approach allows for a comparative understanding of different storage models and their characteristics for various data types.

Data Access: The accessing stage focuses on retrieving and utilizing the stored data effectively. This project developed efficient retrieval methods, ensuring that data is accessible and usable in our desired applications. This involves the creation of optimized SQL and NoSQL queries, and we also made comparisons of their performances.

Applications: The *YELP Insight* project is designed to fit these concepts into real-world applications, providing a comprehensive view of data management in a real-world setting. This hands-on approach ensures that the theoretical knowledge gained is grounded in practical, applicable skills, essential for tackling real-world data challenges.

1.2 Collaboration and Contributions

- Ouyang Mingyu: group leader, overall engagement
- Lin Liangtao: data generation
- Li Zeyun: data storage
- Tang Zheng: data access
- Zhou Yuhao: data analysis and cleaning

2 Data Analysis

According to the compressed JSON files provided by Dr. Yang, we proceed data understanding and cleaning in Python for YELP data.

2.1 Understanding the original YELP Dataset

The *business* dataset is of length 150346 and contains 14 attributes, which are *business_id*, *name*, *address* of each business, together with the *city*, *state*, *postal_code*, *latitude*, *longitude* of a particular business, as well as its *stars* rating which has been rounded to half-stars, with the collective *review_count*. Also, a binary indicator of whether a particular business *is_open* or not is also included. The embedded attribute *attributes* containing miscellaneous **binary** topics about whether a business *BusinessAcceptsCreditCards*, the method of parkings available *BikeParking*, *BusinessParking* et cetera are also stored. Also, an informative array indicating what kinds of *categories* a business belongs to, and the opening *hours* for it is also included in the dataset.
For data types, we see *business_id* (22 character unique), *name*, *address*, *city*, *state*, *postal_code*, *categories* are of **string** (or an array of **strings**) type, while *latitude*, *longitude*, *stars* are of **float** type, *review_count* and *is_open* (0 or 1 for closed or open, respectively) is of **integer** type, and *attributes* (key business attribute to a binary value) and *hours* (key day to value hours using 24-hour clock) are of the embedded **object** type.
After verification, we find that **the Primary Key of *business* is *business_id***.

The *checkin* dataset is of length 131930 and contains 2 attributes, which are *business_id*, which is a **Foreign Key** mapped to *business_id* in *business* dataset, and the *date* for each checkin at the business.
For data types, we see *business_id* is of 22 character unique **string** type, and *date* is also of type *string*, but is a comma-separated list of strings, each with format YYYY-MM-DD HH:MM:SS.
After verification, we find that **the Primary Key of *checkin* is *business_id***.

The *review* dataset is of length 6990280 and contains 9 attributes, which are *review_id* for each review, *user_id* for the user that posts a review, which is a **Foreign Key** mapped to *user_id* in *user* dataset (discuss later), *business_id* for the business that a user posts a review on, which is a **Foreign Key** mapped to *business_id* in *business* dataset, and the *stars* rating which has been rounded to full-stars that the user gave to a particular business, the number of *useful*, *funny*, *cool* votes received, respectively, as well as the textual *text* represents the full given review itself from a user, also the *date* of the review being posted was given.
For data types, we see *review_id* is of 22 character unique **string** type, and *user_id*, *business_id* and *text* are also of type *string*. The attribute *date* is of a special *string* type, for which each of them holds the format of YYYY-MM-DD HH:MM:SS, while the attributes *stars*, *useful*, *funny*, *cool* are of *integer* type.

After verification, we find that **the Primary Key of *review* is *review_id***.

The *tip* dataset is of length 908915 and contains 5 attributes, which are *user_id* for each customer given an amount of tips, which is a **Foreign Key** mapped to *user_id* in *user* dataset (discuss later). The attribute *business_id* for the business of a user that gave some tips to, which is a **Foreign Key** mapped to *business_id* in *business* dataset, and the full textual script of *text* for the tips given, as well as the exact time of the *date* that a tip was written, together with the number of *compliment_count* it has.
For data types, we see *user_id* and *business_id* are of 22 character **string** type and *text* is of normal **string** type. The attribute *date* is of a special *string* type, for which each of them holds the format of YYYY-MM-DD HH:MM:SS, while *compliment_count* is of *integer* type. Even after data cleaning (discuss later), we find that **there is no specific Primary Key of *tip* and we may need to create one**.

The *user* dataset is of length 1987897 and contains 22 attributes, which are *user_id* for a particular user, *name* for the user's first name, and the *review_count* which indicates the number of reviews a user has written. The *yelping_since* attribute shows the exact date and time of when the user joined Yelp. Also, *tip* contains the number of *useful*, *funny*, *cool* votes sent by a specific user, respectively, as well as the *elite* indicator that in which year(s) a user was badged as an elite. *friends* represents the an array of the user's friend presented as *user_id* forms, as well as the *fans* that user has. Subsequently, the *average_stars* records the average rating of all reviews that a user has given so far. Eventually, the attributes *compliment_hot*, *compliment_more*, *compliment_profile*, *compliment_cute*, *compliment_list*, *compliment_note*, *compliment_plain*, *compliment_cool*, *compliment_funny*, *compliment_writer*, *compliment_photos* specifies the number of different types of compliments which is self-explainable, received by the user.
For data types, we see *user_id* is of 22 character unique **string** type, and *name* is also of type *string*. The attribute *yelping_since* is of a special *string* type, for which each of them holds the format of YYYY-MM-DD HH:MM:SS, and the attribute *friends* is an array of *strings*, while the attributes *review_count*, *useful*, *funny*, *cool*, *fans*, *elite* (array of *integers*), *compliment_hot*, *compliment_more*, *compliment_profile*, *compliment_cute*, *compliment_list*, *compliment_note*, *compliment_plain*, *compliment_cool*, *compliment_funny*, *compliment_writer*, *compliment_-photos* are of *integer* type. Finally, the attribute *average_stars* is of *float* type.
After verification, we find that **the Primary Key of *user* is *user_id***.

2.2 Data Cleaning and Statistical Analysis

Through rectifying incorrect, incomplete, and duplicated data, we see that there are 13744, 103, and 23223 missing values in attributes *attributes*, *categories* and *hours* in *business* dataset, and 67 duplicates in *tip* dataset. After data cleaning, we see the *business* dataset contains 117618 rows and *tip* dataset is of length 908848.

After data cleaning, we perform Exploratory Data Analysis on meaningful attributes from the newly cleaned datasets *business* and *review*.
There are totally 86009 number of unique businesses collected in the dataset, for which **Starbucks** is ranked the top in the collection, and restaurant-related business seems to be the most popular business name for which we shall check later with attribute *categories*.

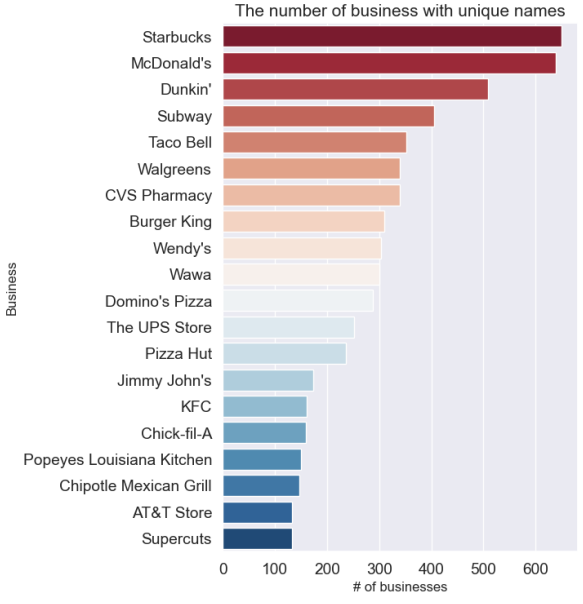


Fig. 1: Top count of unique businesses

Since the collective information *city* and *state* attributes provide are closely related with business counts, we may group them by each city and state to see top popular businesses for each city and state. We observe that there are 1261 city listed in the cleaned Yelp dataset and Philadelphia is the top of the city list in business listing in Yelp dataset followed by Tucson, Tampa etc. Meanwhile, there are 23 state listed, for which state PA is on the top of the state list. Interestingly, observing the full list of state rankings, there are almost half of states having very few business listing.

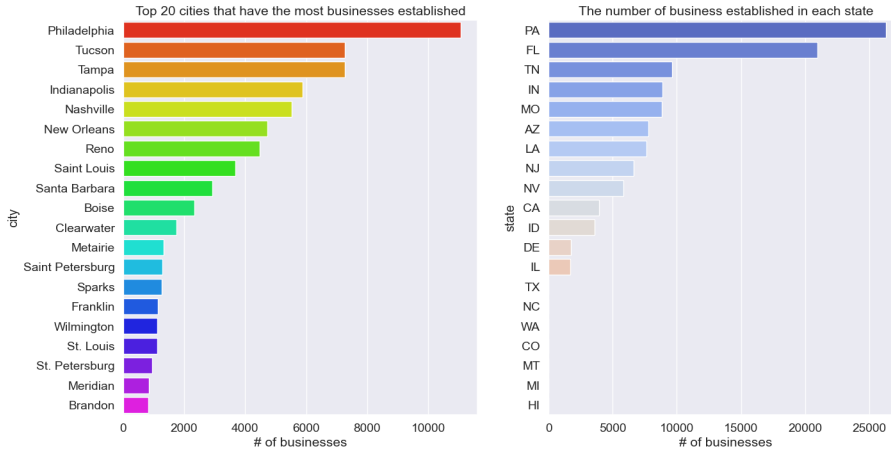


Fig. 2: Top count of businesses in each city and state

For geometric distribution of the businesses, we utilize *latitude* and *longitude* attributes, and view the plot from a satellite vision.

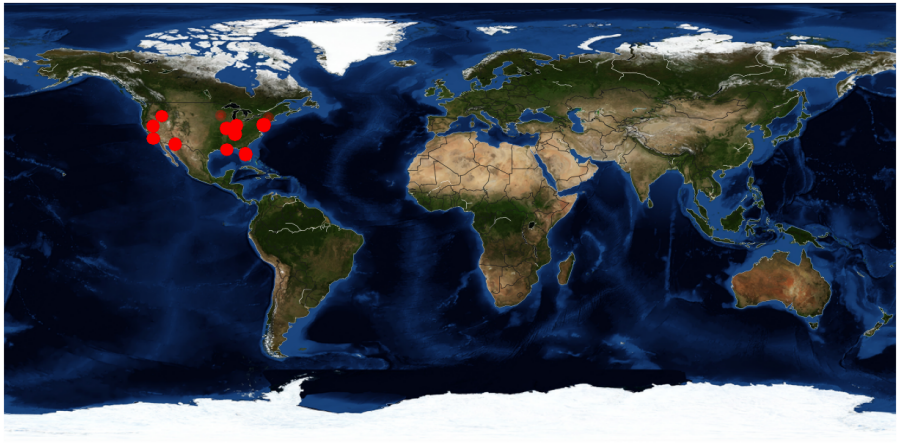


Fig. 3: Satellite vision of businesses in each city and state

To demonstrate the distribution of *stars* rating which are half-stars ranging from 1 to 5, we collect the counts for individual *stars* and use bar-plots. We see that most customers give 3.5, 4, and 4.5 stars to the businesses.

Now, we proceed with attribute pairs investigation and firstly look for businesses that received the most reviews counted globally which utilize attributes *name* and *review_count* and find that **Starbucks** received the most reviews,

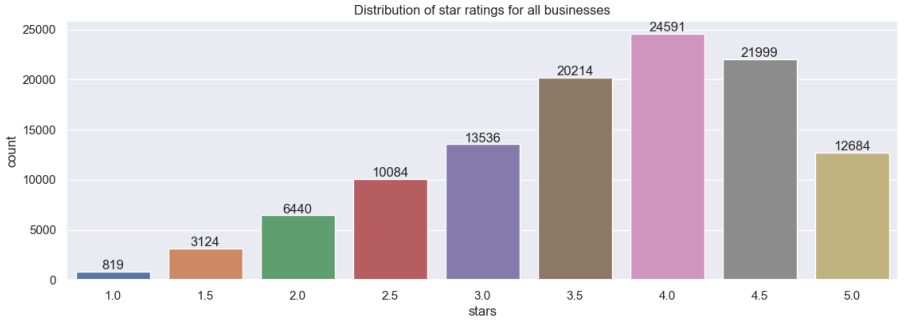


Fig. 4: Distribution of star ratings for all businesses

where one shall not be surprised about the result since **Starbucks** has the most number of stores counted in the region of our interest. Based on the result, we calculate the mean of the review count for all businesses is 71, as well as the median of review count for them is 19.

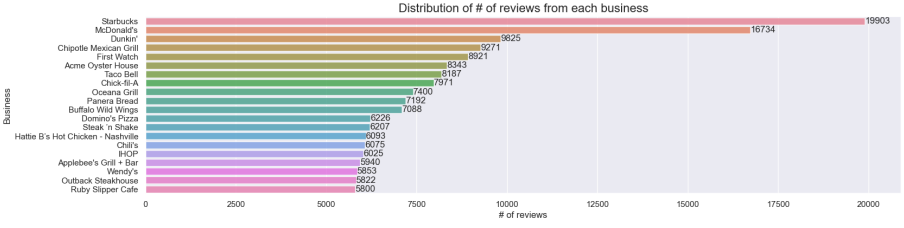


Fig. 5: Distribution of the number of reviews of each business

Based on inspiration to investigate cross-attribute pairs, we then look for businesses that received the most reviews counted in each city which utilize attributes *city* and *review_count* and find that businesses in **Philadelphia** received the most reviews, where one shall not be surprised about the result since **Philadelphia** has the most number of businesses established in the region of our interest. Based on the result, we calculate the mean of the review count for businesses recorded in all cities is 4869, as well as the median of review count for them is 88.

We then checked the proportion of businesses are still available utilizing the attribute *is_open* and find that 81% of them are still open.

Finally, for *business* dataset, we look into the embedded attribute *categories* and would like to show its distribution. Just as we have claimed, restaurant-related business are the most popular, including *Restorants*, *Food*, *Bars* and so on, for which we show the WordCloud of the category names since it contains

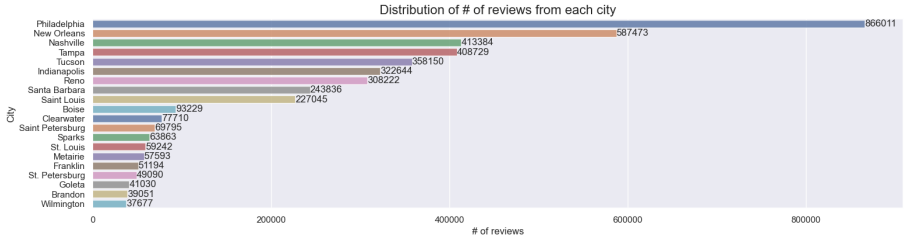


Fig. 6: Distribution of the number of reviews from each city

string elements only. Meanwhile, for *review* dataset, to analyse the textual attribute, we also use WordCloud of the *text* and find that most text reviews are nouns about the place, food, order and service, verbs go, got, know, try, and a few adjectives love and amazing to express their appreciation for a particular business. The combined wordcloud figure is presented in figure7.

(a) Categories (b) Reviews

Fig. 7: Word Cloud Visualization of Categories and Reviews

3 Data Collection

3.1 Web Crawler

To enrich our original dataset, we utilize the web crawler to get up-to-date businesses from yelp.com. The target location is set in Singapore to meet our exploring interests in local business. To go through our web crawler in detail,

The crawler is programmed to navigate through Yelp.com, targeting business listings within Singapore. It captures essential business details such as name, location, contact information, types of services or products offered, customer reviews, and ratings, etc. The YelpSpider class, a subclass from library

'scrapy.Spider', is the core of our crawler. It is initialized with parameters such as location, category, and starting_num to target specific business listings in Singapore. The start_requests method initiates the crawling process by building a URL using these parameters and sending a request to Yelp.com. Once the initial request is made, the parse method takes over, which navigates through the listings on Yelp. For each business listing encountered, it extracts the link to the detailed business page and makes a subsequent request to that page. This design ensures that we capture comprehensive details about each business.

Additionally, the crawler is equipped to handle pagination on Yelp's search results pages. It identifies the link to the next page (if present) and recursively calls the parse method to continue the extraction process until all relevant pages are covered.

Since Yelp.com limits a maximum search result to 20 pages, we have to design an iterative method to "scan over" all the locations in Singapore. We first collected the list of all places (165 subzones in total) in Singapore from Wikipedia [6], then ran our crawler script in each zone to perform the business collection locally.

The total crawling time was about 12 hours. Since there's some overlapping in each subzone, we applied uniqueness filtering at the post-processing step of the data crawling. In order to match the original form of business data, which benefits the crawled data fitting in the pipeline designed for the following steps, we also generate business_ids from every unique URL via MD5 hashing. After removing duplicated data, we finally collected **6606 rows of business** in Singapore.

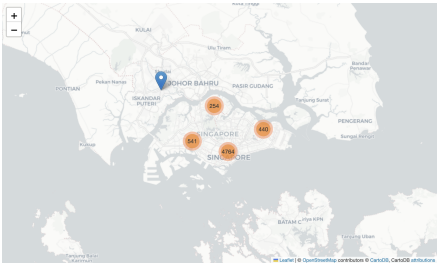
We further visualize the crawled data of Singapore based on the analysis script described in section 2.2. Thanks to the great analysis notebook, we could interact with the result with different zoon-in levels[4]. The crawled business data is shown in figure 8, with each graph clustered by region. We chose the whole of Singapore, NUS, and Marina Bay as our regions of interest to visualize our crawled data.

3.2 Large Language Model Generation

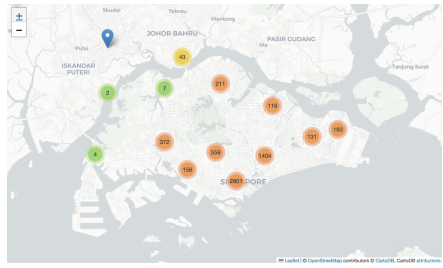
To enrich our original dataset, we also apply some generative methods to perform data augmentation on the original dataset. Due to its robust and versatile capabilities, Large Language Models (LLMs) are extensively employed in a variety of scenarios. We leveraged their classification and generative abilities for conducting intriguing explorations and expansions of our dataset. Specifically, we used Llama-2 to perform data augmentation on **Transaction, Category Classification, and Review Analysis** on the original dataset.

Llama-2 Llama-2 is a collection of popular large language models open sourced by Meta, with state-of-the-art performance. These models are pretrained and fine-tuned generative text models ranging in scale from 7 billion to 70 billion parameters. For our specific task, we have selected 'Llama-2-7B-chat'.

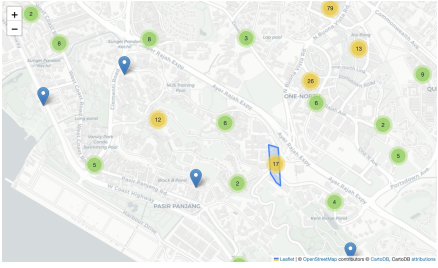
1. Transaction The 'check-in' table for each business includes only a series of check-in dates. Our approach involves generating an estimated transaction



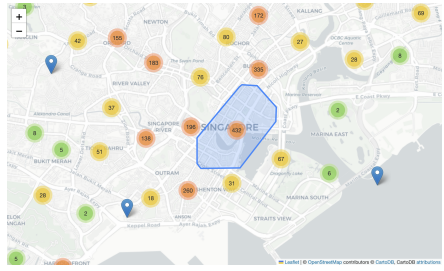
(a) Business in Singapore



(b) Business in Singapore (Zoomed in)



(c) Business around NUS



(d) Business around Marina Bay

Fig. 8: Different Locations in Singapore of Crawled Yelp Data

amount for each date. This will enable us to conduct detailed business analysis presentations using our database in this hypothetical scenario. We employed a technique that integrates random number generation with large language model creation to construct simulated trading scenarios.

Fig.9 is an example of our generated transaction data. For all 131,929 business in 'check-in' table, we generated a total of **13,356,875** pieces of data.


```
{'transaction_id': '1ebde96a-4452-4818-8680-d3f46617f47b',
 'business_id': '3XsBtnrxijQGLqhBeEoCkQ',
 'user_id': 'JkVL-iyda7MoMilA_ppJjQ',
 'time': '2017-08-10 03:48:58',
 'amount': '$476.89',
 'details': 'Purchase of office supplies consisting of printer, files and stationery items.'}
```


Fig. 9: Example of generated transaction data

2. Category Classification

We observed that the "categories" attribute in the "business" table contains disorganized original labels, making it challenging to isolate the specific business categories we need. To address this, we employed Llama-2 to assign each business a distinct category from the 36 categories we have meticulously designed.

For example, consider the initial "categories" attribute listing "Doctors, Traditional Chinese Medicine, Naturopathic/Holistic, Acupuncture, Health & Medical, Nutritionists." This extensive and disorganized categorization poses challenges for practical applications, such as conducting category statistics and screening. To streamline this, we merged the 32 categories listed on the Yelp website with 4 additional ones, creating a total of 36 distinct categories. In the case of the aforementioned example, it would be aptly classified under the "Health" category.

We refined the original business table and generated a total of **60,000** data entries for application analysis. Fig.10 displays the statistical outcomes for each category.

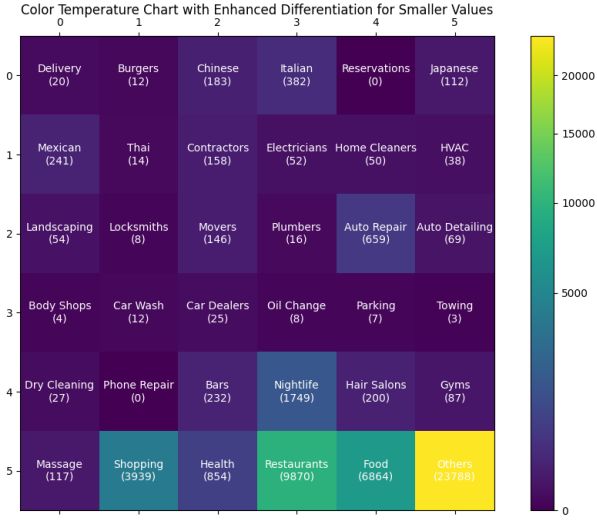


Fig. 10: Generated category distribution map

3. Review Analysis

For the "texts" of user reviews in the "review" table, we can perform natural language processing (NLP) analysis to gain deeper insights. We noticed that on one hand, these reviews are imbued with highly subjective and personal emotions, while on the other, some offer meaningful insights, albeit others resemble mere running accounts. In response to this, we engaged in sentiment analysis and human-robot recognition tasks on these reviews, utilizing Llama-2.

We segmented sentiment analysis into five categories (1:Very Negative, 2:Negative, 3:Neutral, 4:Positive, 5:Very Positive), and Human-Robot Detection into two categories (1 for Human, 0 for Robot). To achieve this, we constructed

specific prompts and employed a few-shot learning approach with Llama-2 to classify reviews. This process resulted in a total of **120,000** pieces of generated data.

For instance, for the first review "Sometimes this food is very very good. Unfortunately it's not consistent. Ordered something I've been getting for years and every other time it tastes incredible. It's like they have different people in the kitchen and you don't know who you will get. So 50 % of the time it's excellent.", the model's generated rating results are "{'sentiment': 3, 'human-robot': 1}". This indicates that the model interprets this review as Neutral and authored by an actual Human.

Furthermore, to validate the effectiveness of our approach, we computed the Mean Squared Error (MSE) and Mean Absolute Error (MAE) between the sentiment analysis scores generated by our method and the star ratings provided by users. We discovered that both metrics were approximately **0.68**, indicating that our method is effective. This is because it closely aligns with the subjective scores directly given by users, while our analysis is fundamentally based on the content of the user reviews themselves.

4 Data Storage

4.1 Schema Design in MySQL

4.1.1 Schema Design and Diagram

In the initial phase, we focused on data-centric aspects in designing the relational database. The attached figure 11 illustrates this initial design. However, upon further analysis, we identified several key deficiencies of simple design:

- Data Storage in JSON Files: A significant portion of our data is stored in JSON files, often as arrays or collections. This format, while flexible, poses challenges for efficient querying and data retrieval.
- Many-to-Many Relationships: The database design includes numerous many-to-many relationships between tables. This approach has led to considerable data redundancy, which not only consumes extra storage space but also complicates data management.
- Query Efficiency: The current design lacks optimization for query efficiency. This oversight can result in slower data retrieval times, impacting the overall performance of the database system.
- Compliance with BCNF (Boyce-Codd Normal Form): There is an absence of a thorough evaluation to determine if the database structure adheres to the BCNF constraints. Compliance with BCNF is crucial for reducing redundancy and ensuring data integrity.

To summarize these, while the initial database design addresses basic data storage needs, it falls short in terms of query efficiency, data normalization, and

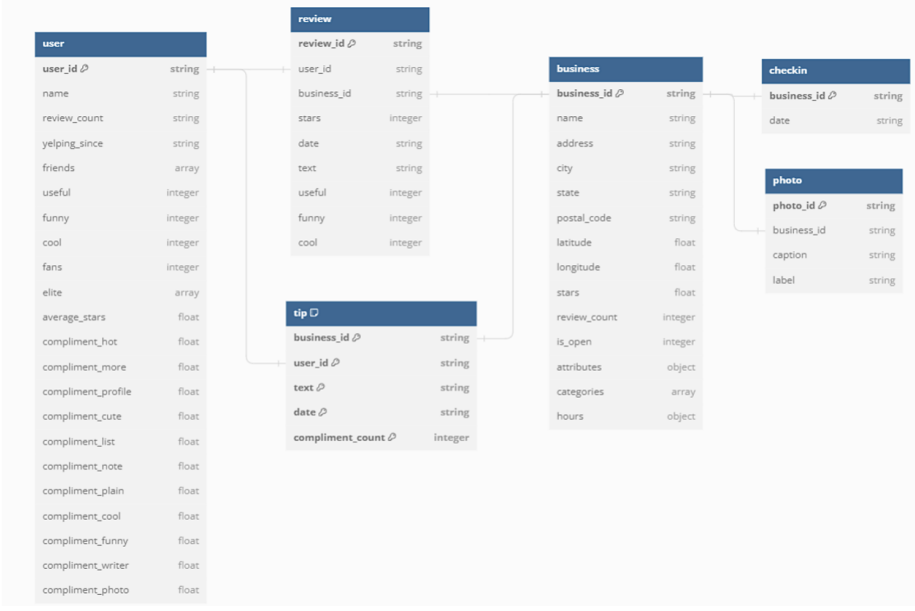


Fig. 11: Initial Design of our Schema

optimal utilization of storage resources. A reevaluation and restructuring of the database schema are recommended to address these issues.

Subsequently, our team focused on optimizing the relational database structure in MySQL, informed by an in-depth analysis of the data structure. The revised design strategically divides different types of data into different tables to improve organization and efficiency. Business-related information is now stored only in the "Business" table, while user-related data is assigned to the "Users" table. Similarly, we have designated the "Review", "Check-ins" and "Photos" tables to store comments, check-ins and photo data respectively. In addition, we introduced the "Tips" and "Transactions" tables to manage tip-related information and transaction data. An important improvement in our redesign is the creation of a specialized "Friends" table, which is dedicated to encapsulating friend relationships between users. This optimized structure of the relational database is intended to simplify data management and improve the efficiency of the database system. An illustration of the optimized relational database design is shown in figure 12.

Our database system is designed to meet the needs of different user roles, providing users, business owners, and platform administrators with rich functions to serve a variety of application scenarios.

4.1.2 Storage Logic

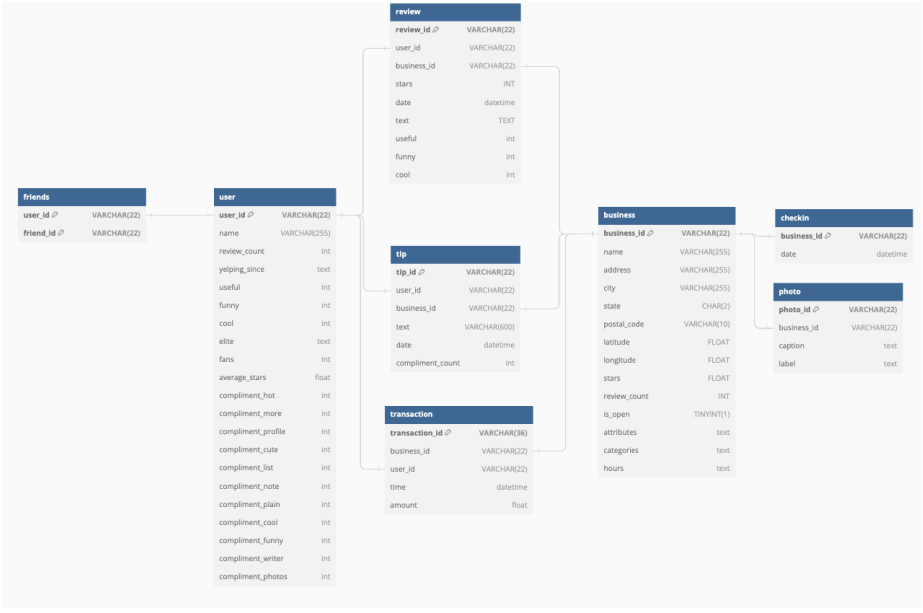


Fig. 12: Final Design of our Schema

Through the above table design, we are able to efficiently store and retrieve information related to business, users, reviews, etc. Associations between tables are established by foreign keys, ensuring data integrity and consistency. This approach not only enhances the relational database's organization but also facilitates effective data management and retrieval. We list our storage logic for MySQL database as follows:

Service data storage: The business table contains basic information about services, including the address, latitude and longitude, and star rating. The attributes, categories, and hours fields use text types to store JSON data.

User data storage: The User table contains basic information about the user, as well as a series of statistics related to the user, such as the number of likes, followers, etc. review data storage: The review table stores users' comments on services, including ratings and dates.

Check-in data store: The checkin table stores the check-in records of users for a service, including the date and time.

Photo data storage: The Photo table stores photos uploaded by users, including photo descriptions and classifications.

Tip data storage: The Tip table stores user prompt information about services, including the date and number of likes.

Transaction data storage: The transaction table stores the transaction records of a user in a business, including the date, time, and amount.

4.1.3 Advantages and Disadvantages

Advantages: The structure of the database is clear, conforms to the design principle of the database paradigm, avoids data redundancy, and establishes the association between tables by using foreign keys to ensure the consistency of data.

Cons: For some business attributes, such as attributes, categories, and hours, storing JSON data using text types may require additional parsing at query time, may affect query performance, and may require regular maintenance of the table structure, especially if business requirements change.

4.2 Collection Design in MongoDB

4.2.1 Non-relational Design Overview

MongoDB's design is similar to MySQL in that it uses Collection instead of tables. Each Collection corresponds to a different set of data for business, users, reviews, and so on. In particular to enhance query performance, MongoDB allows indexing on any document's field or subfield. This approach to indexing is notably different from MySQL's, primarily due to MongoDB's schema-less nature, which provides more flexibility but also requires careful index management to optimize performance.

This is particularly useful in collections with high query demands, such as those storing user reviews, check-ins, and transaction data. Indexing these critical fields, such as date elements in review and transaction collections, significantly speeds up search and sorting operations.

4.2.2 Storage Logic

In MongoDB, data storage is handled through collections, each serving a specific purpose similar to tables in MySQL. Understanding the storage logic in MongoDB, especially in comparison to the more traditional relational model of MySQL is crucial for effective database design. For each collection, here we provide a brief explanation of why its structure is beneficial in MongoDB's document model compared to the relational model of MySQL.

User data storage: The user collection, similar to the User table in MySQL, holds basic user information along with user-related statistics. MongoDB's document model enables the storage of these statistics in a more dynamic and scalable manner compared to the rigid structure of MySQL.

Review data storage: the review collection of MongoDB stores user comments on services, paralleling the review table int'm MySQL.

Check-in data storage: Check-in collection stores the check-in records of users on certain services, which corresponds to the Check-in table in MySQL.

Tip data store: the Tip collection stores user information about services and corresponds to the TIP table in MySQL.

Transaction data storage: the transaction collection stores the transaction records of users in a certain business, which corresponds to the transaction table structure in MySQL.

4.1.3 Advantages and Disadvantages

Advantages: MongoDB supports the storage of unstructured data and adapts to complex json structures such as categories and attributes in business data. Its data model is more flexible and can better respond to changes in business requirements.

Cons: In some scenarios that require complex queries, it may be necessary to write more aggregation pipelines to achieve complex queries, and there are some limitations in constraint support in MongoDB compared to MySQL for scenarios that require strict constraints.

4.2.3 Summary of Database Design

Our database design has been successful in meeting the needs of different application scenarios. Through reasonable table structure and index design, we can store and retrieve data efficiently, while ensuring data integrity and consistency. The design of application scenarios and tasks takes into account the requirements of different user roles, and provides rich functions and flexible query methods. Database systems can support large-scale data storage and efficient queries, providing powerful tools and functions for businesses, users, and administrators.

4.3 Efficient Data Warehousing

When loading cleaned JSON data into MySQL, we experimented with two different approaches. Initially, we attempted to read each document from the JSON file one by one and input the data line by line into the corresponding tables we had created in MySQL. However, due to the substantial size of our JSON file, this method proved to be highly time-consuming.

Subsequently, we adopted an alternative strategy, wherein we employed a chunking mechanism to read the JSON file in segments. We utilized the chunksize parameter along with pandas data frames to store the data temporarily. Then, leveraging the SQLAlchemy engine, we efficiently transferred the entire dataset into the MySQL database. This approach significantly enhanced the efficiency of the data loading process. The approximated data loading time are given in Table 1.

5 Application Scenarios and Tasks

Application Overview: Aiming at providing services for customers, business owners and platform administrators, the proposed application of our database is divided into following 3 scenarios:

Table 1: Comparisons of Execution Time via Different Loading Methods.

Loading Method	**Dataset**	**Execution Time (Approx.)**
PyMySQL + Line by line	business	1 hour
PyMySQL + Line by line	friends	10 hours +
SQLAlchemy + Chunksize	business	30 seconds
SQLAlchemy + Chunksize	friends	1 hour, 10 minutes

5.1 For Customers

The most general and widely-used functions of our database. These applications provide some general information about the businesses for individuals, and could enhance the individual's experience through personalized and relevant data.

1. Get top-n businesses in the city n=10, 20, 50, 100
   - Detail: Rankings are based on a composite score considering factors like ratings, review counts, and customer feedback.
2. Get top-10 businesses within a specified radius m=0.1, 0.5, 1, 5, 10 km
   - Detail: Uses geolocation data (latitude, longitude) to find businesses in proximity.
3. Retrieve all reviews for a specific business by "business_id", sorted by "stars"
   - Detail: Sorts reviews in descending order of star ratings, offering a quick overview of the most favorable customer feedback.
4. Access and update the user's own profile
   - Detail: Allows users to view and modify their personal information, including name, reviews, likes, etc.
5. View reviews of friends, sorted by datetime
   - Detail: Integrates social connections to display recent activity of friends, enhancing the social aspect of the platform.
6. Access recent reviews from influencers
   - Detail: Focuses on providing insights from reputable sources such as elite or popular users.

5.2 For Business Owners

Features for business owners focus on providing insights into customer behavior and relative business trends, with a strong emphasis on data privacy and sensitive information control across businesses and users.

1. Retrieve check-in histories for their business
   - Detail: Enables business owners to analyze customer visit patterns over time, aiding in business planning.

2. Access partial customer information
   - Detail: Provides essential customer data while maintaining strict adherence to privacy regulations.
3. Analyze time-series data for peak and off-peak times
   - Detail: Time-series characteristic retrieval. Assists in identifying business rush hours for better staffing and resource management.
4. Obtain average sentiment value from reviews within duration: 1, 7, 30 days
   - Detail: Uses sentiment analysis to rate customer reviews, offering insights into overall customer satisfaction.
   - Note: sentiment values are scores varying from 0 to 5, generated by LLM
5. Access transaction records within the duration specified
   - Detail: Provides detailed financial transaction data for specified periods, aiding in financial management and analysis.
   - Note: transaction data are simulated based on the real review data.

5.3 For Platform Administrators

For the most advanced tracking throughout businesses and customers. All functions are yielded with the highest accessibility. Provide YELP administrators with a toolbox for handling and maintaining the entire platform.

1. Identify active users within a specified period
   - Detail: Focuses on users who have engaged with the platform, particularly through reviews, in a given timeframe.
2. Detect robot-generated feedback
   - Detail: Employs advanced algorithm(LLM) to identify and filter non-human interactions and feedback.
   - Note: robot-generated values are scores in 0 or 1, generated by LLM
3. Analyze average business ratings across categories or regions
   - Detail: Provides a macro view of business performance across various sectors and geographical areas.
4. Find out influencers: Identify top-1000 popular users
   - Detail: Ranks users based on the sum of ratings like useful, funny, and cool, highlighting influential platform participants.

6 Implementation and Performance Analysis

6.1 SQL Queries in MySQL

1. Get top-n businesses in the city n=10, 20, 50, 100

```
SELECT * FROM business WHERE city = 'Philadelphia'
ORDER BY stars
DESC LIMIT 100;
```

2. Get top-10 businesses within a specified radius m=0.1, 0.5, 1, 5, 10 km

```
SELECT * FROM business
WHERE SQRT(POW((latitude - (39.9298)), 2) +
            POW((longitude - (-85.984)), 2)) * 111.32 <= 1
ORDER BY stars
DESC LIMIT 100;
```

3. Get all reviews for a specific business, sorted by stars

```
SELECT * FROM review
WHERE business_id = (SELECT business_id from business
    WHERE name = "West Side Kebab House")
ORDER BY stars
DESC LIMIT 10;
```

4. Get and update user's own profile (e.g. reviews, likes, elite records etc....)

```
SELECT user_id, name, yelping_since FROM user
WHERE user_id = 'j14WgRoU_-2ZE1aw1dXrJg';

UPDATE user SET useful = 520, funny = 520, cool = 520
where user_id = 'j14WgRoU_-2ZE1aw1dXrJg';
```

5. Get review of their own friends, sorted by datetime

```
SELECT c.* FROM review  c
JOIN friends f ON c.user_id = f.friend_id
WHERE f.user_id = 'j14WgRoU_-2ZE1aw1dXrJg'
ORDER BY c.date DESC LIMIT 20;
```

6. Get the recent review from influencers

```
WITH RankedUsers AS (
SELECT
    u.user_id,
    PERCENT_RANK() OVER (ORDER BY u.fans) AS percentile_rank
FROM user u
WHERE (u.elite LIKE '%2016%' OR u.elite LIKE '%2017%'
                                    OR u.elite LIKE '%2015%')
)
SELECT r.*
FROM review r
JOIN RankedUsers ru ON r.user_id= ru.user_id
WHERE ru.percentile_rank > 0.7
ORDER BY r.date DESC;
```

7. Get all check-in histories of their specific business

```
SELECT * FROM checkin c
WHERE c.business_id = -QI8Qi8XWH3D8y8ethnajA;
```

8. Get the most information of their customers

```
SELECT u.user_id, u.name, u.review_count, u.yelping_since
FROM user u
WHERE u.user_id IN
(SELECT DISTINCT user_id FROM review
WHERE business_id = '9n-1LQLX3ntBfBtMwgSpig');
```

9. Get the time-series characteristics (peak times and off-peak times) of their business

```
SELECT HOUR(date) hour, count(*) frequency FROM checkin
WHERE business_id = '-QI8Qi8XWH3D8y8ethnajA'
GROUP BY hour ORDER BY hour;
```

10. Get average sentiment value from the review table within a duration

11. Get transaction records of their business within a duration

```
SELECT * FROM transaction
WHERE business_id =
(SELECT b.business_id FROM business b
WHERE b.name = "Zio's Italian Market")  AND
time BETWEEN '2013-01-01' AND '2015-01-01';
```

12. Get all active users within a duration

```
SELECT u.* FROM user u
WHERE u.user_id IN
(SELECT DISTINCT r.user_id FROM review r
WHERE r.date BETWEEN '2015-01-01' AND '2017-01-01') ;
```

13. Average rating of businesses across different categories or regions.

```
SELECT city, AVG(stars) FROM business
GROUP BY city;
```

14. Filter out influencers: get top-1000 users with the most popularity

```
SELECT  u.*, (useful + funny + cool) as popularity
FROM user u
order by popularity
DESC
LIMIT 1000;
```

6.2 NoSQL Queries in MongoDB

1. Get top-n businesses in the city n=10, 20, 50, 100

```
db.business.find({ city: "Cityname" }).sort({ stars: -1 })\
.limit(n);
```

2. Get top-10 businesses within a specified radius m=0.1, 0.5, 1, 5, 10 km

```
var bulkOps = [];

db.business.find({}).forEach(function(business) {
    var location = {
        type: "Point",
        coordinates: [business.longitude, business.latitude]
    };
    bulkOps.push({
        updateOne: {
            filter: { _id: business._id },
            update: { $set: { location: location } }
        }
    });
    if (bulkOps.length === 1000) {
        db.business.bulkWrite(bulkOps);
        bulkOps = [];
    }
});
if (bulkOps.length > 0) {
    db.business.bulkWrite(bulkOps);
}
db.business.createIndex({ location: "2dsphere" });
let userLocation = { type: "Point", coordinates: [userLng, \
userLat] };
db.business.aggregate([
    {
        $geoNear: {
            near: userLocation,
            distanceField: "distance",
```

```
                spherical: true,
                maxDistance: m * 1000,
                query: { is_open: 1 }
            }
        },
        {
            $limit: 10
        },
        {
            $project: {
                business_id: 1,
                name: 1,
                address: 1,
                distance: "$distance"
            }
        }
]);
```

3. Get all reviews for a specific business, sorted by stars

```
var targetBusinessName = "business_name";
var targetBusiness = db.business.findOne({ name:\
targetBusinessName });
if (targetBusiness) {
    var targetBusinessId = targetBusiness.business_id;
    var reviewsForTargetBusiness = db.review.find({\
    business_id: targetBusinessId }).sort({ stars: 1 });
```

4. Get and update user's own profile (e.g. reviews, likes, elite records etc....)

```
db.user.find({ user_id: "YourUserID" });
db.user.update({ user_id: "YourUserID" }, { $set: \
{ /* Update fields */ } });
```

5. Get review of their own friends, sorted by datetime

```
var targetUserId = "your_target_user_id";
var targetUser = db.user.findOne({ user_id: targetUserId });
if (targetUser) {
    var friendsList = targetUser.friends.split(",") : [];
    var friendsReviews = db.review.find({ user_id: { $in:\
    friendsList } }).sort({ date: 1 });
} else {
    print("User not found with user_id: " + targetUserId);
}
```

6. Get the recent review from influencers

```
var currentYear = new Date().getFullYear();
var threeYearsAgo = currentYear - 3;
var eliteUserIds = db.user.find({
  $expr: {
    $and: [
      { $regexMatch: { input: "$elite", regex: new RegExp\
      (threeYearsAgo + "|" + (threeYearsAgo - 1) + "|" + \
      (threeYearsAgo - 2)) } },
      { $gt: ["$fans", 0] }
    ]
  }
}).sort({ "fans": -1 }).limit(Math.ceil(0.3 * db.user.\
countDocuments())).map(user => user.user_id);
var recentInfluencerReviews = db.review.find({ \
"user_id": { $in: eliteUserIds } }).sort({ "date": -1 })\
.limit(10);
```

7. Get all check-in histories of their specific business

```
db.checkin.find({ "business_id": \business_id" })
```

8. Get the most information of their customers

```
var businessUserIds = db.review.find({'business_id':\
'XQfwVwDr-v0ZS3_CbbE5Xw'}, {'user_id': 1, '_id': 0}).\
map(function(doc) {
    return doc.user_id;
});
var users = db.user.find({
    'user_id': { $in: businessUserIds } }, \
    { '_id': 0, 'user_id': 1, 'name': 1, \
    'review_count': 1, 'yelping_since': 1 });
```

9. Get the time-series characteristics (peak times and off-peak times) of their business

```
db.transaction.aggregate([
    {
        $match: {
            "business_id": "XQfwVwDr-v0ZS3_CbbE5Xw"
        }
    },
    {
```

```
            $project: {
                hour: { $hour: "$time" }
            }
        },
        {
            $group: {
                _id: "$hour",
                count: { $sum: 1 }
            }
        },
        {
            $sort: { count: -1 }
        },
    ]);
```

10. Get average sentiment value from the review table within a duration

11. Get transaction records of their business within a duration

```
db.transaction.find({'business_id': 'D8y8ethnajA',\
'date': {$gte: '2014-01-01',$lte: '2017-01-01'}},\
{_id:0, business_id: 1, date: 1} )
```

12. Get all active users within a duration

```
var startDate = ISODate("2010-06-07T22:09:11.000+00:00");
var endDate = ISODate("2020-06-08T22:09:11.000+00:00");
const reviewUserIds = db.review.distinct("user_id", {
  "date": {
    $gte: startDate,
    $lt: endDate
  }
});
db.user.find({
  "user_id": { $in: reviewUserIds }
}, { _id: 0, user_id: 1, name: 1 })
```

13. Average rating of businesses across different categories or regions.

```
db.business.aggregate([
  {
    $group: {
      _id: "$categories",
```

```
      average_rating: { $avg: "$stars" }
    }
  },
  {
    $sort: { average_rating: -1 }
  }
]);
```

14. Filter out influencers: get top-1000 users with the most popularity

```
db.user.aggregate([
  {
    $project: {
      user_id: 1,
      name: 1,
      popularity: { $sum: ["$useful", "$funny", "$cool"] }
    }
  },
  {
    $sort: { popularity: -1 }
  },
  {
    $limit: 1000
  }s
]);
```

6.3 Running Time Comparison

We report the execution time of each query as follows in table 2. To maintain a fair comparison, we conducted the execution of queries on the same device. The schema and collection design are implemented as described in the previous section. Please refer to Sections 5.1, 5.2, and 5.3 for query semantic details.

Please note that for queries with the symbol * in table 2, we either designed extra indexes, or pre-updated extra attributes to speed up the MongoDB execution. Thus there is a huge difference in execution time from MySQL.

6.4 Database Performance Analysis

From Table 2, we can generally observe:

Faster Execution in MongoDB: Queries 1, 2, 4, 5, and 6 showcase MongoDB's superior performance, attributed to its efficient handling of unstructured data and effective use of indexes. For instance, in Query 2, the Geospatial index of MongoDB greatly boosts the location-based querying.

Exceptional Cases: Queries 8 and 12, where MongoDB takes longer time, highlight scenarios where relational databases might perform better. This could

Table 2: Comparisons of execution time via different databases. Note that for each row, we calculated the percentage of MongoDB execution time over MySQL execution time.

No. Query	Implementation Database MySQL Time	MongoDB Time	Ratio of Time Spent MongoDB/MySQL (%)
1	0.53 sec	0.13 sec	24.53%
2	0.33 sec	0.001 sec *	0.30%
3	2 min 6.45 sec	19.48 sec	15.38%
4	2.55 sec	0.001 sec	0.04%
5	7 min 42.40 sec	0.10 sec *	0.001%
6	3 min 1.35 sec	0.43 sec *	0.002%
7	10.31 sec	0.05 sec	0.48%
8	1 min 53.66 sec	2 min 13.76 sec	105.49%
9	10.00 sec	5.77 sec	57.70%
10	-	0.76 sec	-
11	26.32 sec	20.65 sec	78.46%
12	2 min 30.05 sec	2 min 55.87 sec	117.21%
13	0.43 sec	0.38 sec	93.02%
14	3.54 sec	1.97 sec	55.65%

be due to the nature of the query, which might be more suited to a structured query language or involve complex relations better handled by MySQL. For instance, Query 8 accesses partial customer information given a "business_id", which involves a join-like operation between "business" and "reviews" as tables or collections. In this case, a relational database like MySQL may have a better performance than MongoDB.

Special Considerations: The marked improvement in MongoDB's performance for certain queries (marked with *) is due to additional optimizations like extra indexes or pre-updated attributes. However, these outperforming may basically come from **our optimization focus on the MongoDB side**. It is worth pointing out that there may be possibilities for further optimization of MySQL as well, but since we will be using MongoDB in the web service demo, we have further emphasized the optimization of MongoDB.

To sum up our observation, MongoDB generally offers superior query performance in our cases due to its schema flexibility and efficient handling of unstructured data, but there are some complicated scenarios where a traditional relational database like MySQL may be more suitable.

Moreover, we'd like to share some experiences found in developing this project. When working with data storage, we found that the flexibility of MongoDB makes it less work to do data entry, but relatively complex to maintain and write query. MySQL, on the other hand, requires a lot of integrity constraints

to be added when performing data storage, which is a relatively large amount of designation work, but is easier to maintain the database later.

7 Prototype Demo

7.1 Backend Design

In our prototype, the backend is structured around three core components: MongoDB, Flask [3], and PyMongo. MongoDB serves as our database (DB), Flask acts as the request server, and PyMongo is utilized as the Database Management System API (DBMS API). To achieve a robust backend, Flask and PyMongo are integrated into a unified Python script. This integration enables a unified interaction between the database and the server.

Key functionalities implemented in the backend include the queries and corresponding parameters we defined above, each of which is accessible through specific API requests. We design our APIs following RESTful principles [5] to making it easier for frontend components to interact.

7.2 Frontend Design

We implemented a Vue-based front-end website to interact with our Yelp Insight database platform, making it simple to interact and retrieve the data with visualization. The details of our interactive front-end design will be released together with the demo. See the "demo" directory under our project GitHub directory (https://github.com/Linlt-leon/Yelp-Insight) for more details.

8 Conclusions

In this project, we proposed a comprehensive data platform, namely *Yelp Insight*, for customers, business owners, and administrators. The main contribution of our project can be concluded as follows: 1. We added more than 6,000 entries of Yelp data using the customized web crawler, and 180,000 rows of features via large language model generation techniques. 2. We carefully explored the characteristics of our Yelp dataset and presented many interesting insights into Yelp. 3. We provide more than 100 querying methods via the combination of our 16 basic queries and flexible parameters, which could cover a wide range of demands from customers, business owners, and platform administrators. 4. We built a demo website that can provide data access services via a non-coding interface. We believe the presented Yelp Insight well covers all aspects of the project tasks.

References

1. Mongodb. https://www.mongodb.com/ (2023)
2. Mysql. https://www.mysql.com/ (2023)
3. Grinberg, M.: Flask web development: developing web applications with python. " O'Reilly Media, Inc." (2018)
4. OpenStreetMap contributors: Planet dump retrieved from https://planet.osm.org . https://www.openstreetmap.org (2017)
5. Richardson, L., Ruby, S.: RESTful Web Services. O'Reilly (2007), https://www.safaribooksonline.com/library/view/restful-web-services/9780596529260/
6. Wikipedia contributors: List of places in singapore. https://en.wikipedia.org/wiki/List_of_places_in_Singapore (2020)
7. Yelp Inc.: Yelp dataset. https://www.yelp.com/dataset (2021)