Group 21

Simple Yelp

Jiacheng Zhu Hongkai Ding Yuxiang Ying Jingyi Guo

Demo Video

1. Introduction and Project Goals

Our team is developing a website called Simple Yelp, designed to function similarly to Yelp, targeting the challenge of assessing the quality of businesses effectively. The primary goal of this project is to offer a user-friendly platform where individuals can easily evaluate restaurants and other businesses based on comprehensive reviews.

Simple Yelp allows users to search for businesses by name, city, and state, displaying results that include star ratings, recent reviews, and historical feedback. The site also features functionality to highlight trending users and businesses based on the volume of reviews they post or receive. Additionally, we have integrated a proximity search feature, enabling users to discover businesses near their current location.

The motivation for creating Simple Yelp stems from a desire to simplify the process of finding quality services and dining options. By providing a platform where users can access detailed reviews and ratings, we aim to enhance decision-making and foster a transparent business-reviewing environment.

2. Architecture and Technologies used

The team utilizes a standard client-server architecture for their web application. The application's database is hosted on MySQL through AWS RDS (credentials in the appendix). NodeJS and Express power the website's server, allowing the team to construct routes that execute queries to access the AWS database. React.JS drives the frontend, facilitating communication with the server and rendering pages for users in their browsers. For a detailed overview of the server and client architecture, refer to the clear layout in the team's GitHub Repository (GitHub Repo). The details of the data source, schema, and description of each distinct page will be provided in later sections.

3. Data Source

There are three datasets that the team uses in this project. They are all open-source data from https://www.yelp.com/dataset. Business.json contains business data, including location data, attributes, and categories. Review.json contains full review text data, including the user_id that wrote the review and the business_id for which the review is written (Table 1).

The statistics of the pre-processed dataset are provided in Table 2. Data processing overall is smooth, with no serious problems encountered (pre-process code in the appendix). The datasets are easily processed using the Pandas library in Python. During preprocessing, the team selects a relevant subset of attributes, and the table originally used IDs that were specially generated (e.g., Pns2l4eNsfO8kk83dixA6A). The team rewrites the keys to integers such as 1, 2, 3. Moreover, the team also decomposes the tables during preprocessing, which will be discussed further in the next section. During exploratory data analysis, one notable aspect of our data processing is that every user in user.json and every business in business.json has appeared at least once in review.json. With Pandas, we are able to derive summary statistics for important features (Table 2).

In the business dataset, the significant features include ratings (stars) and the count of reviews. The mean rating for businesses is 3.6, and the median rating is 3.5. The mean review count is 44.9, with a median of 15.0.

In the user dataset, the only meaningful numerical attribute worth summarizing is the stars (ratings). The mean rating is 3.75, with a median of 4.0. The histogram also illustrates that a 5-star rating is the most common. Furthermore, the range of dates in the review dataset spans from 2005-02-16 03:23:22 to 2022-01-19 19:48:45.

Our team utilizes the datasets to retrieve relevant information about businesses and users. Business.json and Review.json, on their own, can generate useful information about individual

users and businesses. Additionally, since review.json contains the user_id that wrote the review and the business_id for which the review is written, connecting these three datasets allows us to determine how many reviews one user posts and how many reviews one business receives. Therefore, we can generate a list of top users and top businesses based on the number of reviews.

4. Database

The team's data ingestion procedure and entity resolution efforts began with first identifying functional dependencies in the pre-processed dataset. The team has determined the existence of the following functional dependencies:

- review_id -> user_id, business_id, stars, useful, funny, cool, text, date
- user_id -> name, yelping_since
- business_id -> name, address_id, categories, hours
- address_id -> address, latitude, longitude, postal_code
- postal_code -> city, state

Then the team drew the following ER Diagram: Figure 1

After that, the team finally arrived at the derivation of the 3NF (Third Normal Form) of the database schema. The 3NF schema of the database was determined as follows:

1. review(review_id, user_id, business_id, stars, useful, funny, cool, text, date)
    - user_id REFERENCE user(user_id)
    - business_id REFERENCE business(business_id)
2. user(user_id, name, yelping_since)
3. business(business_id, name, address_id, categories, hours)
    - address_id REFERENCE address(address_id)
4. address(address_id, address, latitude, longitude, postal_code)
    - postal_code REFERENCE zipcode(postal_code)
5. zipcode(postal_code, city, state)

In addition to above tables, the team also created a View for easy access to combined business information:

- View BusinessInfoStatic(business_id, name, categories, hours, address, latitude, longitude, postal_code, city, state)

Here are the statistics for each 3NF table: Table 3

The team concludes that the current schema is in 3NF because we have followed the 3NF decomposition algorithm. Initially, each functional dependency is transformed into an individual schema, ensuring that the decomposition is dependency-preserving. Then, the team checks if each schema contains the candidate key from the original schema to ensure lossless join

decomposition. After these two steps, the team observes that no two schemas have the same key, leading to the conclusion that a 3NF schema for the database has been derived.

5. Web Application Description

HomePage: The homepage provides clear and concise guidelines for the main functionalities of this website. Users can navigate to search for businesses, find nearby businesses, and view trending users and businesses. Users can search for businesses by name, city, and state on the homepage and view star ratings, historical reviews, and recent reviews. As for the nearby business and trending users/businesses functionalities, users can access the results with just one click.

SearchResultPage: After a user searches for a business on the home page, a set of results will be returned and displayed on the search result page. Users can click on any of the business names to view details of its information.

BusinessInfoPage: After a user clicks on a specific business name through the search result page or any other pages, the Business Info page will be displayed with detailed information about the business, including its name, address, average star rating, operating hours, total review count, featured reviews, and all reviews ordered from latest to oldest.

UserInfoPage: After a user clicks on a username associated with each review, the User Info page will be displayed, showing the user's name, when they started using Yelp, their total review count, and all of the user's reviews ordered from latest to oldest.

Trending Business Page: On the trending business page, a list of the top 20 businesses based on the number of reviews received is displayed. The number of reviews is shown along with the name of the business. Since the query can take some time, a loading animation is added when accessing the Trending Business page. To see detailed information, including reviews and average star ratings, users can click on the name of the business and will be navigated to the dedicated page for each individual business.

Trending Users Page: Similar to the trending business page, on the trending users page, a list of the top 20 users based on the number of reviews posted is displayed alongside the username. Since the query can take some time, a loading animation is added when accessing the Trending Users page. By clicking on the username, users will be navigated to the dedicated page for individual users and can access detailed information such as profile pictures and detailed reviews.

Nearby Businesses Page: On the Nearby Businesses page, a list of businesses within one mile of the user's location will be displayed. The average rating and the distance in meters will be shown along with the business name. To see detailed information, including reviews and average star ratings, users can click on the name of the business and will be navigated to the dedicated page for each individual business.

6. API Specification

Route 1: Search Business

1. **Description**: Search for businesses by name, city, and state, and return a list of matching businesses.
2. **Request Path and Method**: GET /searchBusiness
    - Retrieves a list of businesses matching the given criteria.
3. **Request Parameters**:
    - `name` (query, string, optional): Part of the business name to search for.
    - `city` (query, string, optional): City where the business is located.
    - `state` (query, string, optional): State where the business is located.
4. **Response Parameters**:
    - `business_id` (integer): Unique identifier for the business.
    - `name` (string): Name of the business.

Route 2: List Cities

1. **Description**: Fetch a list of distinct cities from the database.
2. **Request Path and Method**: GET /cities
    - Retrieves a list of distinct city names.
3. **Request Parameters**: None
4. **Response Parameters**:
    - `cities` (array of strings): List of city names.

Route 3: List States

1. **Description**: Fetch a list of distinct states from the database.
2. **Request Path and Method**: GET /states
    - Retrieves a list of distinct state names.
3. **Request Parameters**: None
4. **Response Parameters**:
    - `states` (array of strings): List of state names.

Route 4: Business Information

1. **Description**: Retrieve detailed information for a specific business by its ID.
2. **Request Path and Method**: GET /business/:businessId
    - Retrieves information of a business specified by its ID.
3. **Request Parameters**:
    - `businessId` (path, integer, required): The unique identifier of the business.
4. **Response Parameters**:
    - Includes business ID, name, categories, operating hours, address details, and geographical coordinates.

Route 5: Business Reviews

1. **Description**: Fetch all reviews associated with a specific business ID.
2. **Request Path and Method**: GET /business/:businessId/reviews
    - Retrieves all reviews for a given business.
3. **Request Parameters**:
    - `businessId` (path, integer, required): The unique identifier of the business.
4. **Response Parameters**:
    - A list of reviews, each including review ID, user ID, user name, star rating, review text, date, and reaction counts (useful, funny, cool).

Route 6: Review Summary

1. **Description**: Get a summary of reviews for a specific business, including count and average star rating.
2. **Request Path and Method**: GET /business/:businessId/reviewSummary
    - Retrieves summary statistics for reviews of a specific business.
3. **Request Parameters**:
    - `businessId` (path, integer, required): The unique identifier of the business.
4. **Response Parameters**:
    - `review_count` (integer): Number of reviews for the business.
    - `average_stars` (float): Average star rating for the reviews.

Route 7: User Information

1. **Description**: Retrieve information for a specific user by their ID.
2. **Request Path and Method**: GET /user/:userId/info
    - Retrieves detailed information of a user specified by their user ID.
3. **Request Parameters**:
    - `userId` (path, integer, required): The unique identifier of the user.
4. **Response Parameters**:
    - `user_id` (integer): The unique identifier of the user.
    - `name` (string): Name of the user.
    - `yelping_since` (date): The date the user started using the service.

Route 8: User Reviews

1. **Description**: Fetch all reviews posted by a specific user.
2. **Request Path and Method**: GET /user/:userId/reviews
    - Retrieves all reviews written by a specified user.
3. **Request Parameters**:
    - `userId` (path, integer, required): The unique identifier of the user.
4. **Response Parameters**:

- A list of reviews, including review ID, business ID, star rating, review text, date, and reactions (useful, funny, cool).4

Route 9: User Review Summary

1. **Description**: Get a summary of reviews for a specific user, including count and average star rating.
2. **Request Path and Method**: GET /user/:userId/reviewSummary
   - Retrieves summary statistics for reviews written by a specific user.
3. **Request Parameters**:
   - `userId` (path, integer, required): The unique identifier of the user.
4. **Response Parameters**:
   - `review_count` (integer): Number of reviews written by the user.
   - `average_stars` (float): Average star rating for the reviews.

Route 10: Top Users (Complex, Response Time ~ 18s)

1. **Description**: Retrieve a list of top 20 users based on the number of reviews they have posted.
2. **Request Path and Method**: GET /top-users
   - Retrieves the top 20 users by review count.
3. **Request Parameters**: None
4. **Response Parameters**:
   - A list of top users, including user ID, name, and review count.

Route 11: Top Businesses (Complex, Response Time ~ 10s)

1. **Description**: Retrieve a list of top 20 businesses based on the number of reviews received.
2. **Request Path and Method**: GET /top-business
   - Retrieves the top 20 businesses by review count.
3. **Request Parameters**: None
4. **Response Parameters**:
   - A list of top businesses, including business ID, name, and review count.

Route 12: Featured Review (Complex, Response Time ~ 10s)

1. **Description**: Fetch a "featured" review for a specific business, determined by having the highest impact score.
2. **Request Path and Method**: GET /featured-review/:businessId
   - Retrieves the featured review for a specified business.
3. **Request Parameters**:
   - `businessId` (path, integer, required): The unique identifier of the business.
4. **Response Parameters**:

- `business_id` (integer), `review_id` (integer), `review_text` (string), and `impact_score` (integer).

Route 13: Nearby Businesses (Complex, Response Time ~ 10s)

1. **Description**: Retrieve businesses within a 1-mile radius of the specified geographical coordinates, including average star ratings and review counts.
2. **Request Path and Method**: GET /nearby-businesses/:latitude/:longitude
   - Retrieves businesses near the specified latitude and longitude.
3. **Request Parameters**:
   - `latitude` (path, float, required): Latitude of the location.
   - `longitude` (path, float, required): Longitude of the location.
4. **Response Parameters**:
   - A list of nearby businesses, including business ID, name, distance in meters, average star rating, and review count.

Route 14: Top Businesses by City (Response Time depends on # Businesses in City)

1. **Description**: Retrieve a list of the top 20 businesses within a specified city based on the number of reviews they have received.
2. **Request Path and Method**: GET /top-business/:city
   - Retrieve a list of the top 20 businesses based on input city
3. **Request Parameters**:
   - `city` (path, string, required): The name of the city to filter top businesses. It is case-sensitive and must exactly match the city name stored in the database.
4. **Response Parameters**:
   - A JSON array of top businesses, each object containing: business_id: The unique identifier of the business. name: The name of the business. review_count: The number of reviews that the business has received.

7. Queries Explanation

All the queries for the 14 routes are attached in the appendix (queries appendix). Below are explanations of how they're used for the 5 most complex queries:

Route 12, featured review, is one of the complex queries in the system, taking around 2-3 seconds to run, depending on the number of reviews a business has. This query is called each time a business information page is rendered. It calculates an impact score for each review by summing the funny, useful, and cool scores. Then, it selects the review with the highest impact score as the featured review. Displaying the featured review at the top of all reviews on the business info page helps viewers quickly and accurately understand reviewers' opinions.

Route 14, top business by city, is also a complex query in the system, taking approximately 6-7 seconds to run, depending on the number of businesses in a city. The query takes a city as input and returns the top 20 businesses in that city, based on the number of reviews. This route is used on the Trending Business page, where users can select a city from a drop-down menu to display the top 20 businesses in that city.

Route 13, nearby business, is another complex query in the system, taking about 10 seconds to run, depending on the user's actual location. This query takes latitude and longitude as input and displays businesses within 1 mile that exist in the database. Please note that not every city has businesses in the database; however, the database does contain many Philadelphia businesses. This query is used on the homepage, where a nearby business button exists that, when clicked, retrieves the user's location and returns the nearby business page.

Route 10, top user, is another complex query in the system, taking about 18 seconds to finish. This query does not take any parameters but calculates the top 20 trending users in the database based on review counts. The route is called when a user clicks the trending user button on the menu bar, and then the top 20 users are displayed on the trending user page.

Route 11, top business, is a similar query to top user and takes about 10 seconds to run. This query also does not take any arguments but calculates the top 20 trending businesses in the database based on review counts. The route is called when a user selects "All Cities" in the drop-down menu on the trending business page, and then the top 20 businesses are displayed.

8. Performance Evaluation

Table 4 records the timings before and after optimization for each of the five complex queries discussed in the previous section, along with their respective optimization methods. In Route 12, an index is created on the `business_id` in the review table because the query joins the review table with the business table using the key `business_id`, thus enhancing performance. However, the team faced challenges in creating indexes, particularly on the review table; these issues will be explained in the next section. Route 14 involves creating multiple indexes as it requires joins across multiple tables—zipcode, address, business, and review—with indexes created on the foreign keys to speed up the queries. Similarly, Route 13 creates an index on the foreign keys for joins among the review, business, address, and zipcode tables. Routes 10 and 11 also see indexes created on the review table to optimize queries related to trending businesses and users, with further details on the indexing challenges to be discussed subsequently.

9. Technical Challenges

There were two major technical challenges the team encountered during development. The first challenge occurred during the database population phase. Since the review table has around 7 million tuples, and each tuple contains a piece of text, uploading the review CSV to the database through DataGrip took a really long time and constantly failed before the upload

finished. The team's solution was to break the review table into several small CSV files and then upload the CSV files to the database separately.

The second challenge occurred during index creation, which was also related to the size of the review table. When the team tried to create an index on the review table, it took an unreasonable amount of time and constantly failed. The solution proposed by the team was to temporarily upgrade the CPU, memory, and storage on the AWS RDS index, and then switch the configuration back after the index creation was finished. After the team changed the CPU to the highest setting on RDS, the index creation finished within a reasonable time. Moreover, after the team switched the CPU and other settings to the lowest tier, query optimization still took advantage of the index created.

10. Appendix

Guest Credentials

Note: this guest credentials is READ ONLY

```
{
  "rds_host":
"database-cis550.c9wo06mw4elr.us-east-1.rds.amazonaws.com",
  "rds_port": "3306",
  "rds_user": "guest",
  "rds_password": "cGrc4CU3P1MUTaTgW2go",
  "rds_db": "YELP"
}
```

Github Page

https://github.com/jjz5463/cis550_yelp_review_web

EDA Code

https://colab.research.google.com/drive/11OIG_LP415m2y8TIlj0nZmb_OGdsqrY0?usp=sharing

SQL Queries

1. *Route 1: Search Business*

```
SELECT b.business_id, b.name
FROM business b
JOIN address a ON b.address_id = a.address_id
JOIN zipcode z ON a.postal_code = z.postal_code
WHERE b.name LIKE ? AND z.city = ? AND z.state = ?
```

2. *Route 2: List Cities*

```
SELECT DISTINCT z.city
FROM address a
JOIN zipcode z ON a.postal_code = z.postal_code
ORDER BY z.city ASC;
```

3. *Route 3: List States*

```
SELECT DISTINCT z.state
FROM zipcode z
```

```
ORDER BY z.state ASC;
```

4. *Route 4: Business Information*

```
SELECT
      business_id,
      name,
      categories,
      hours,
      address,
      latitude,
      longitude,
      postal_code,
      city,
      state
FROM BusinessInfoStatic
WHERE business_id = ?;
```

5. *Route 5: Business Reviews*

```
SELECT review_id, r.user_id, u.name, stars, text, date, useful,
funny, cool
FROM review r
JOIN user u ON r.user_id = u.user_id
WHERE business_id = ?
ORDER BY date DESC;
```

6. *Route 6: Review Summary*

```
SELECT
      COUNT(review_id) AS review_count,
      AVG(stars) AS average_stars
FROM review
WHERE business_id = ?
```

7. *Route 7: User Information*

```
SELECT
      user_id,
      name,
      yelping_since
FROM user
```

```
WHERE user_id = ?;
```

8. *Route 8: User Reviews*

```
SELECT review_id, business_id, stars, text, date, useful, funny,
cool
FROM review
WHERE user_id = ?
ORDER BY date DESC;
```

9. *Route 9: User Review Summary*

```
SELECT
      COUNT(review_id) AS review_count,
      AVG(stars) AS average_stars
FROM review
WHERE user_id = ?
```

10. *Route 10: Top Users (Complex, Response Time ~ 18s)*

```
SELECT
      u.user_id,
      u.name,
      COUNT(r.review_id) AS review_count
FROM
      user u
JOIN review r ON u.user_id = r.user_id
GROUP BY
      u.user_id
ORDER BY review_count DESC
LIMIT 20;
```

11. *Route 11: Top Businesses (Complex, Response Time ~ 10s)*

```
SELECT
      b.business_id,
      b.name,
      COUNT(r.review_id) AS review_count
FROM
      business b
JOIN review r ON b.business_id = r.business_id
GROUP BY
```

```
    b.business_id
ORDER BY review_count DESC
LIMIT 20;
```

12. *Route 12: Featured Review (Complex, Response Time ~ 10s)*

```
WITH ReviewCounts AS (
    SELECT
    business_id,
    COUNT(review_id) AS reviews_count
    FROM
    review
    GROUP BY
    business_id
    HAVING
    COUNT(review_id) >= 10
), ReviewImpactScores AS (
    SELECT
    r.business_id,
    r.review_id,
    r.text AS review_text,
    (r.useful + r.funny + r.cool) AS impact_score,
    ROW_NUMBER() OVER (PARTITION BY r.business_id ORDER BY
(r.useful + r.funny + r.cool) DESC) AS `rank`
    FROM
    review r
    JOIN ReviewCounts rc ON r.business_id = rc.business_id
)
SELECT
    ris.business_id,
    ris.review_id,
    ris.review_text,
    ris.impact_score
FROM
    ReviewImpactScores ris
WHERE
    ris.`rank` = 1 AND ris.business_id = ?
ORDER BY
    ris.business_id;
```

13. *Route 13: Nearby Businesses (Complex, Response Time ~ 10s)*

```
SELECT
      b.business_id,
      b.name,
      a.latitude,
      a.longitude,
      ST_Distance_Sphere(point(a.longitude, a.latitude), point(?,
?)) AS distance_in_meters,
      COALESCE(AVG(r.stars), 0) AS average_stars,
      COALESCE(COUNT(r.review_id), 0) AS review_count
FROM
      business b
JOIN address a ON b.address_id = a.address_id
LEFT JOIN review r ON b.business_id = r.business_id
WHERE
      ST_Distance_Sphere(point(a.longitude, a.latitude), point(?,
?)) <= 1609.34
GROUP BY
      b.business_id, b.name, a.latitude, a.longitude
ORDER BY distance_in_meters ASC;
```

14. *Route 14: Top Businesses by City (Complex, 6-7 seconds)*

```
SELECT
      b.business_id,
      b.name,
      COUNT(r.review_id) AS review_count
FROM
      business b
JOIN address a ON b.address_id = a.address_id
JOIN zipcode z ON a.postal_code = z.postal_code
JOIN review r ON b.business_id = r.business_id
WHERE z.city = ?
GROUP BY
      b.business_id
ORDER BY review_count DESC
LIMIT 20;
```

Tables

Table 1: Dataset Source & Description			
Dataset Name	Description	Source	Raw Schema
business.json	Contains business data including location data, attributes, and categories.	https://www.yelp.com/dataset	Business (business_id, name, address, city, state, postal_code, latitude, longitude, stars, review_count, is_open, attributes, categories, hours)
review.json	Contains full review text data including the user_id that wrote the review and the business_id the review is written for.	https://www.yelp.com/dataset	Review (review_id, user_id, business_id, stars, useful, funny, cool, text, date)
user.json	User data including the user's friend mapping and all the metadata associated with the user.	https://www.yelp.com/dataset	User (user_id, name, review_count, yelping_since, useful, funny, cool, elite, friends, fans ... compliment_funny, compliment_writer, compliment_photos)

Table 2: Pre-Processed Datasets Statistics			
Dataset Name	Size (mb/gb)	# rows	# columns
business.json	118.9 MB	150346	14
review.json	5.34 GB	6990280	9
user.json	3.36 GB	1987897	22

Table 3: 3NF Tables Statistics		
Table	# Rows	#Columns
address	143554	5
business	150346	5
review	6950640	9
user	1976566	3
zipcode	3360	3
View BusinessInfoStatic	150273	10

Table 4: Performance Before and After optimization			
	Optimization Method	Time before Optimization	Time after Optimization
Route 12, featured review	Index on review table's foreign key `business_id`: `idx_review_business_id`	1m 37s 499ms	6s 306ms
Route 14, top business by city	Index on the zipcode table's `city` column: `idx_zipcode_city` Index on the address's `postal_code`: `idx_address_postal_code` Index on the business's `address_id`: `idx_business_address_id` Index on the review's `business_id`:	1m 45s 227ms	6s 92ms

	`idx_review_business_id`		
Route 13, nearby business	Index on the review table's `business_id`: `idx_review_business_id` Index on the business's `address_id`: `idx_business_address_id`	1m 38s 310ms	19s 118ms
Route 10, top user	Index on the review's `user_id`: `idx_review_user_id`	2m 33s 729ms	19s 240ns
Route 11, top business	Index on the review table's `business_id`: `idx_review_business_id`	1m 41s 402ms	8s 135ms

Figures

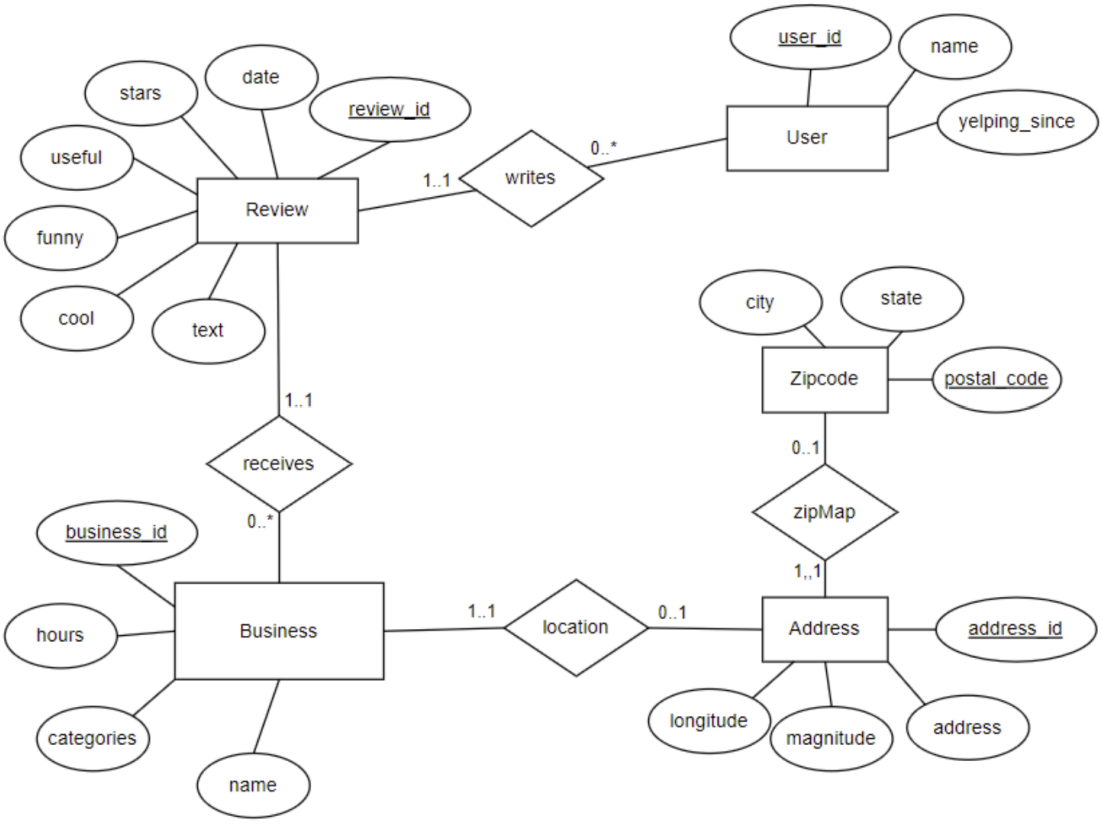

Figure 1: ER Diagram

Demo Video Link

https://drive.google.com/file/d/18rd7_UpZ90Nt7PLLhEIaUPQUYgZ1kZd1/view?usp=drive_link