Data101 Final Project

Checkpoint Writeup

Manan Bhargava, Bianca Isabel Poblano, Nawoda Wijesooriya

Manan Bhargava	3036535321	manan.bhargava@berkeley.edu
Bianca Isabel Poblano	3035917442	bpoblano@berkeley.edu
Nawoda Wijesooriya	3037027827	nawodakw@berkeley.edu

Dataset Selection

For this project, we chose the **Yelp Dataset** from the Yelp Open Dataset, which includes various aspects of business reviews, check-ins, user data, and more. This dataset was selected because it provides a rich source of real-world data that can be analyzed for various purposes, such as business performance evaluation and user analysis.

The data is freely available on Yelp's official website (https://www.yelp.com/dataset). We downloaded the dataset in the form of JSON files. We focused primarily on the users.json, `reviews.json` and `business.json` files for this project, as these provided the most important information for our potential queries.

The `business.json` dataset contains **150,346 rows** and **14 columns,** has a file size of **118.9 MB,** and took **8.5s** to load. The exported **business.csv** file size is **99.1 MB**, and took **5.2s** to export from a pd dataframe to CSV format.

The `users.json` dataset contains **1,987,897 rows** and **22 columns**, has a file size of **3.36 GB**, and took **3 minutes and 21.0 seconds** to load. We cannot sample from the users dataset because of the foreign key constraint.

The `reviews.json` dataset contains **6,990,280 rows** and **9 columns**, has a file size of **5.34 GB**, and took **9 minutes and 52.2 seconds** to load the complete dataset. As a result of the substantial file size and load time, we decided to sample the dataset.

We performed the sampling using the **JSON reader object** provided by pandas, which allowed us to read the dataset in chunks and sample the rows as we went through them.

We specifically sampled **1M rows** of reviews, which allowed us to create a representative sample while keeping the dataset size manageable.

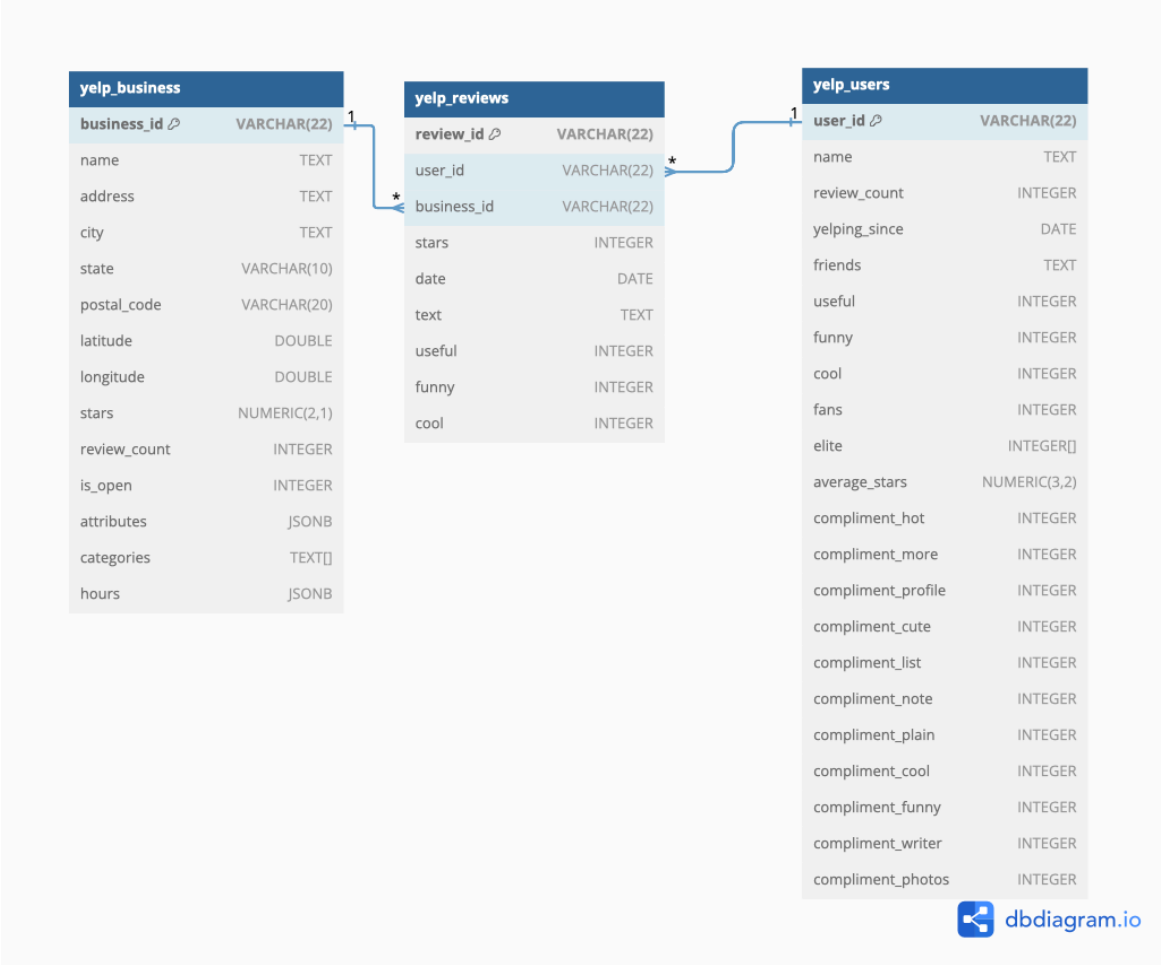
With sampling, the process took **1 minute and 42 seconds** to sample **1,000,000** rows from the review table. These sampled datasets are a lot more manageable than using the entire dataset.

The sampling process took some time, especially when dealing with the large sizes of the reviews and users datasets. We therefore stored the sampled data to a CSV file using `review_df.to_csv('../data/review.csv', index=False)` for ease of access in the future.

Database Structure

We used **PostgreSQL** to store and manage the sampled dataset. The database schema is relational, consisting of a table for `reviews`, `business,` and `users`. The tables are linked by foreign keys with `reviews` linking to `business` by a `business_id` column and reviews linking to users by the user_id column.

ER Diagram



Data Loading Process

1. **Data Extraction**: We started by downloading the Yelp dataset, which is available in JSON format.
2. **Sampling and Truncation**: We then used Python and pandas to sample and truncate the dataset. Specifically, we read the JSON files in chunks and randomly sampled rows to reduce the size of the dataset, ensuring it was manageable for processing and analysis.
3. **Data Loading into PostgreSQL**: The sampled data was then loaded into the PostgreSQL database using SQL commands. For each table, we created a `CREATE TABLE` statement with appropriate columns and data types. The data was imported from CSV files generated from pandas into the database via the `COPY` command.

System and Database Setup

The following is a comprehensive diagram of the different steps involved, with the data conversion pipeline, high level approach, and detailed approach. We described the detailed approach even more in detail down below.

Data Conversion Pipeline for Reviews

Step 1: Data Chunking & Sampling

Data Conversion

High Level Approach

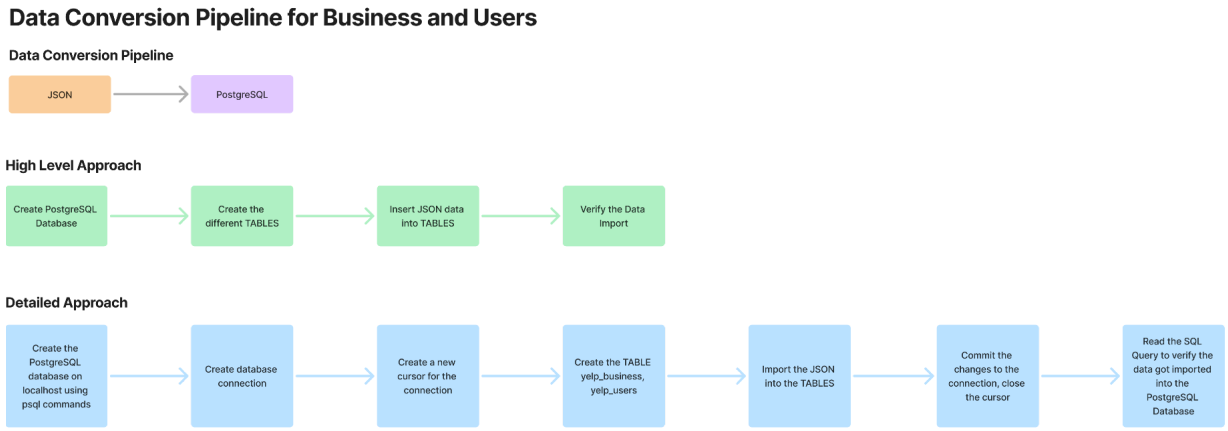
Detailed Approach

Step 2: Piping Data into PostgreSQL Database

Data Conversion

High Level Approach

Detailed Approach



The process of loading the data into PostgreSQL and setting up the database schema has been completed successfully, and a portion of the dataset is now available for querying.

```
with conn_str.cursor() as cursor:
    cursor.execute("""
    CREATE TABLE IF NOT EXISTS yelp_business (
        business_id VARCHAR(22) PRIMARY KEY,
        name TEXT,
        address TEXT,
        city TEXT,
        state VARCHAR(10),
        postal_code VARCHAR(20),
        latitude DOUBLE PRECISION,
        longitude DOUBLE PRECISION,
        stars NUMERIC(2,1),
        review_count INTEGER,
        is_open INTEGER,
        attributes JSONB,
        categories TEXT[],
        hours JSONB
    );
        """)
    cursor.execute("""
            CREATE TABLE IF NOT EXISTS yelp_users (
                user_id VARCHAR(22) PRIMARY KEY,
                name TEXT,
                review_count INTEGER,
                yelping_since DATE,
                friends TEXT,
                useful INTEGER,
                funny INTEGER,
                cool INTEGER,
                fans INTEGER,
                elite TEXT,
                average_stars NUMERIC(3,2),
                compliment_hot INTEGER,
                compliment_more INTEGER,
                compliment_profile INTEGER,
                compliment_cute INTEGER,
                compliment_list INTEGER,
                compliment_note INTEGER,
                compliment_plain INTEGER,
                compliment_cool INTEGER,
                compliment_funny INTEGER,
                compliment_writer INTEGER,
                compliment_photos INTEGER
            );
        """)
    cursor.execute("""
            CREATE TABLE IF NOT EXISTS yelp_reviews (
                review_id VARCHAR(22) PRIMARY KEY,
                user_id VARCHAR(22) NOT NULL,
                business_id VARCHAR(22) NOT NULL,
                stars INTEGER CHECK (stars >= 1 AND stars <= 5),
                date DATE,
                text TEXT,
                useful INTEGER,
                funny INTEGER,
                cool INTEGER,
                FOREIGN KEY (user_id) REFERENCES yelp_users(user_id) ON DELETE CASCADE,
                FOREIGN KEY (business_id) REFERENCES yelp_business(business_id) ON DELETE CASCADE
            );
        """)
    conn_str.commit()
```

executed in 13ms, finished 16:13:08 2024-11-19

```
with open(business_file_path, 'r', encoding='utf-8') as f:
    for line in f:
        data = json.loads(line)


        business_id = data.get('business_id')
        name = data.get('name')
        address = data.get('address')
        city = data.get('city')
        state = data.get('state')
        postal_code = data.get('postal_code')
        latitude = data.get('latitude')
        longitude = data.get('longitude')
        stars = data.get('stars')
        review_count = data.get('review_count')
        is_open = data.get('is_open')
        attributes = json.dumps(data.get('attributes')) if data.get('attributes') else None
        categories = data.get('categories')
        if categories:
            categories = [category.strip() for category in categories.split(',')]
        hours = json.dumps(data.get('hours')) if data.get('hours') else None


        cursor.execute("""
            INSERT INTO yelp_business (
                business_id, name, address, city, state, postal_code,
                latitude, longitude, stars, review_count, is_open,
                attributes, categories, hours
            ) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
            ON CONFLICT (business_id) DO NOTHING;
        """, (
            business_id, name, address, city, state, postal_code,
            latitude, longitude, stars, review_count, is_open,
            attributes, categories, hours
        ))


conn_str.commit()
```

executed in 15.8s, finished 16:17:39 2024-11-19

```
cursor = conn_str.cursor()

with open(user_file_path, 'r', encoding='utf-8') as file:
    for line in file:
        data = json.loads(line)

        user_id = data.get('user_id')
        name = data.get('name')
        review_count = data.get('review_count')
        yelping_since = data.get('yelping_since')
        friends =  data.get('friends')
        useful = data.get('useful')
        funny = data.get('funny')
        cool = data.get('cool')
        fans = data.get('fans')
        elite = data.get('elite')
        average_stars = data.get('average_stars')
        compliment_hot = data.get('compliment_hot')
        compliment_more = data.get('compliment_more')
        compliment_profile = data.get('compliment_profile')
        compliment_cute = data.get('compliment_cute')
        compliment_list = data.get('compliment_list')
        compliment_note = data.get('compliment_note')
        compliment_plain = data.get('compliment_plain')
        compliment_cool = data.get('compliment_cool')
        compliment_funny = data.get('compliment_funny')
        compliment_writer = data.get('compliment_writer')
        compliment_photos = data.get('compliment_photos')


        try:
            cursor.execute("""
                INSERT INTO yelp_users (
                    user_id,
                    name,
                    review_count,
                    yelping_since,
                    friends,
                    useful,
                    funny,
                    cool,
                    fans,
                    elite,
                    average_stars,
                    compliment_hot,
                    compliment_more,
                    compliment_profile,
                    compliment_cute,
                    compliment_list,
                    compliment_note,
                    compliment_plain,
                    compliment_cool,
                    compliment_funny,
                    compliment_writer,
                    compliment_photos
                ) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
                ON CONFLICT (user_id) DO NOTHING;
            """, (
                user_id, name, review_count, yelping_since, friends,
                useful, funny, cool, fans, elite, average_stars,
                compliment_hot, compliment_more, compliment_profile,
                compliment_cute, compliment_list, compliment_note,
                compliment_plain, compliment_cool, compliment_funny,
                compliment_writer, compliment_photos
            ))
        except Exception as e:
            print(f"Error inserting record for user_id {user_id}: {e}")
            conn_str.rollback()


conn_str.commit()
```

executed in 9m 35s, finished 16:28:28 2024-11-19

In [25]:
```
print("Number of Rows in business")
print(pd.read_sql_query('SELECT COUNT(*) FROM yelp_business;', conn_str))
```

executed in 89ms, finished 16:35:53 2024-11-19

```
Number of Rows in business
    count
0  150346
```

In [27]:
```
print("Number of Rows in Users")
print(pd.read_sql_query('SELECT COUNT(*) FROM yelp_users;', conn_str))
```

executed in 208ms, finished 16:36:43 2024-11-19

```
Number of Rows in Users
     count
0  1987896
```

We are using our local computers for the compute resources, and are using the same setup to store the database.

Task Selection

To demonstrate our progress, here is an example query that selects some data from the `reviews` table:

```
SELECT *
FROM yelp_reviews
LIMIT 10;
```

This query retrieves the first 10 rows from the `reviews` table, showcasing basic functionality and ensuring that the data is correctly loaded into the database. This simple query serves as a basic check on the integrity of the dataset and provides a snapshot of the review data.

The screenshots for the queries are included below:

In [13]:
```
pd.read_sql_query('SELECT * FROM yelp_users LIMIT 5;', conn_str)
```
executed in 85ms, finished 16:30:29 2024-11-19

Out[13]:

	user_id	name	review_count	yelping_since	friends	useful	funny	cool	fans	
0	j14WgRoU_-2ZE1aw1dXrJg	Daniel	4333	2009-01-25	ueRPE0CX75ePGMqOFVj6IQ, 52oH4DrRvzzl8wh5UXyU0A...	43091	13066	27281	3138	2009,2010,2011,2012,2013,2014,2
1	2WnXYQFK0hXEoTxPtV2zvg	Steph	665	2008-07-25	LuO3Bn4f3rlhyHIaNfTlnA, j9B4XdHUhDfTKVecyWQgyA...	2086	1010	1003	52	2009,20
2	SZDeASXq7o05mMNLshsdIA	Gwen	224	2005-11-29	enx1vVPnfdNUdPho6PH_wg, 4wOcvMLtU6a9Lslggq74Vg...	512	330	299	28	
3	hA5lMy-EnncsH4JoR-hFGQ	Karen	79	2007-01-05	PBK4q9KEEBHhFvSXCUirIw, 3FWPpM7KU1gXeOM_ZbYMbA...	29	15	7	1	
4	q_QQ5kBBwlCcbL1s4NVK3g	Jane	1221	2005-03-14	xBDpTUbai0DXrvxCe3X16Q, 7GPNBO496aecrjJfW6UWtg...	14953	9940	11211	1357	2006,2007,2008,2009,2010,20

5 rows × 22 columns

In [15]:
```
pd.read_sql_query('SELECT * FROM yelp_business LIMIT 4;', conn_str)
```
executed in 51ms, finished 16:30:47 2024-11-19

Out[15]:

	business_id	name	address	city	state	postal_code	latitude	longitude	stars	review_count	is_open	
0	Pns2l4eNsfO8kk83dixA6A	Abby Rappoport, LAC, CMQ	1616 Chapala St, Ste 2	Santa Barbara	CA	93101	34.426679	-119.711197	5.0	7	0	{'ByAppointment
1	mpf3x-BjTdTEA3yCZrAYPw	The UPS Store	87 Grasso Plaza Shopping Center	Affton	MO	63123	38.551126	-90.335695	3.0	15	1	{'BusinessAccepts(
2	tUFrWirKiKi_TAnsVWINQQ	Target	5255 E Broadway Blvd	Tucson	AZ	85711	32.223236	-110.880452	3.5	22	0	{'WiFi': 'u'no'', 'Ha
3	MTSW4McQd7CbVtyjqoe9mw	St Honore Pastries	935 Race St	Philadelphia	PA	19107	39.955505	-75.155564	4.0	80	1	{'WiFi': 'u'free'', 'C

Future Plan

The next steps involve integrating non-relational parts of the dataset into the project using Apache Spark for processing and storing large-scale data. Apache Spark will be used to handle the unstructured portions of the Yelp dataset, such as review text and business categories. Spark's distributed processing capabilities will allow us to process large datasets efficiently, and its ability to work seamlessly with both structured (SQL-based) and unstructured data (using Spark SQL and DataFrames) makes it an ideal choice for handling complex queries and large-scale analytics.

We will compare the performance of PostgreSQL (a relational database) and Apache Spark (a distributed data processing system) by evaluating query response times, scalability, and flexibility in handling large-scale data. This comparison will help us determine the most effective system for different parts of our dataset and inform our decisions about future analysis.

By integrating both relational and distributed non-relational data processing systems, we aim to optimize our dataset storage, processing, and query execution, ensuring that the dataset is easy to access, scalable, and analyzable for future tasks.