Isha Raju and Shreeya Rajanarayanan

Final Report

Goal :

The goal of this project is to design and develop a search application for a tweet store that is able to search by hashtag, word, user and time range.

Data collection:

Using the Twitter API, both of us were able to get access to a Twitter developer account. Originally 30,000 tweets were gathered using the hashtag “Covid19”. However, removing duplicate retweets reduced it to only around 7,500 tweets. So for the sake of complexity, we decided to gather more tweets.

Finally, a total of 50,000 tweets were gathered using the hashtags “Covid19” and “crypto”. These hashtags were chosen as both topics were prevalent in the news and were interesting to us. This was done on April 23rd, 2021. The data collected was not interrupted. Parsing through tweets in one pass and removing duplicates resulted in 16,170 entries for user information and 11,282 entries for unique tweets. Only tweets in English were gathered.

Originally, the cursor method was used to collect data into a text file, but this collection method only gave us the tweet text and not other important information such as the user information, timestamp, etc. After consulting our cohort, we were able to build upon the streaming method shown in class to collect all the necessary data into a JSON file.

Databases Used:

PostgreSQL is an object-relational database system (it includes additional features like table inheritance) while MySQL is a relational database. We first attempted to proceed using MySQL, but we had some problems with downloading and implementing the software on our Macs. For this reason, we decided to use PostgreSQL to store and query the user data. Postgres also has some handy features

such as the ability to custom define data types, index types and functional language. We used PostgreSQL to store user data.

MongoDB is an open source platform written in C++ with a very easy setup environment. It is a cross-platform, document-oriented and non-structured database. MongoDB provides high performance, high availability, and auto-scaling. It is a NoSQL database and has flexibility with querying and indexing. We chose to use MongoDB because we found it to be accessible and well-documented, which made it easy to use. We used MongoDB to store tweet data.

Column Name in user_table	Tweet Attribute Name	Data Type	Description
user_id	id	bigint	User Id number
user_name	name	varchar(250)	User's name
user_screen_name	screen_name	varchar(250)	User's screen name
user_verified	verified	boolean	Whether user is verified or not
user_location	location	varchar(350)	User identified Location
user_followers_count	followers_count	integer	Number of people following user
user_friends_count	friends_count	integer	Number of people user is following
user_favorites_count	favorites_count	integer	Number of tweets liked by the user
user_status_count	statuses_count	integer	Number of tweet including retweets created by user
user_created_at	created_at	varchar(300)	The UTC datetime that the user account was created

Table 1: Data stored in PostgreSQL

Name of Attribute in MongoDB	Tweet Attribute Name	Description
user_id	Id (from user object)	User Id number
tweet_id	id_str	Tweet id number
tweet_favorite_count	favorite_count	Number of times tweet has been liked
tweet_retweet_count	retweet_count	Number of times tweet has been retweeted
tweet_language	lang	Language of tweet. Only English tweets were extracted
tweet_timestamp	timestamp_ms	Epoch time tweet was published
tweet_text	full_text	Tweet text
tweet_hashtags	hashtags	Used for tagging (inside hashtags object, which is inside entities object)
tweet_created_at	created_at	Date and time that tweet was created

Table 2: Data stored in MongoDB

Data Storage:

Next the JSON file is read one tweet at a time and processed to be split into SQL and NosSQL databases. Tables 1 and 2 on the previous page outline the information stored in each database. In our data storage code, we considered whether the tweet was a regular tweet (case 1) or a retweet (case 2). If the tweet was a retweet, each tweet object would contain a retweeted_status nested object and this retweeted_status nested object would not exist for regular tweets. Therefore, we checked for the existence of the retweeted_status object and gathered data for storage differently based on whether or not the tweet was a retweet.

For case 1 (regular tweets), all user information was gathered from the user json object nested within each general tweet object. Most tweet information would be gathered from the general tweet object except for hashtags and tweet text. Hashtags were gathered from the entities nested object. If there was an extended tweet object, the tweet text was taken from the extended_tweet object. If there was no extended tweet object, the text was taken from the general tweet object.

For case 2 (retweet), all information was gathered from within the retweeted_status object, which is nested within the general tweet object. User info was gathered from the user object within the retweeted_status object. Most tweet

information would be gathered from the retweeted_status object except for hashtags and tweet text. Hashtags were taken from the entities nested object within the retweeted_status object. If there was an extended tweet, the tweet text was taken from the extended_tweet object nested within the retweeted_status object. If there was no extended tweet, the text was taken from the retweeted_status object.

After gathering all the info, user info was saved in PostgeSQL. Only unique tweets were stored in MongoDB. If the tweet already existed, which could be the case for retweets, then the retweet count and favorite count were just updated for the existing tweet. Figure 2 below shows the above process in a picture.

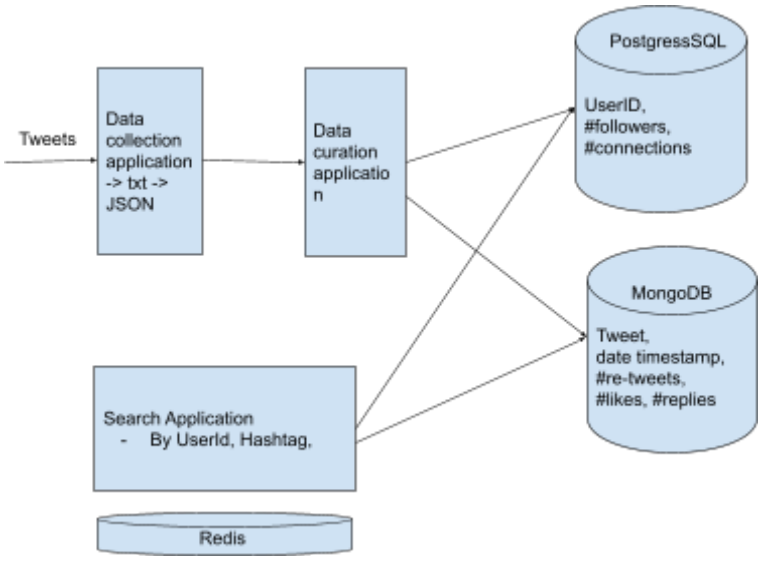

Figure 2: Overview of the storage process

Indexing is very important for improving the performances of search queries. When we continuously perform searches in a document, we should index those fields that match our search criteria. In MongoDB, we can index any field indexed with primary and secondary indices. By making query searches faster, MongoDB indexing enhances the performance. Indexes were added to the tweet database for the following: tweet_id, tweet_text, tweet_hashtags, tweet_created_at.

Caching:

Redis is an open-source (BSD licensed), in-memory data structure store used as a database, cache, and message broker. One of the most popular use cases of Redis is caching. Since Redis is an in-memory database, its data access operations are faster than any other disk-bound database could deliver. It makes Redis the perfect choice for caching. Its key-value data storage is another plus because it makes data storage and retrieval much simpler. Redis promises sub-millisecond long data processing operations. Hence we chose Redis for Caching data.

Search Application and Results

The search application interacts with the user in order to fetch the desired information. The application user is first prompted to enter “1” if they want user information or “0” if they want tweet related information.

If they select user information, the application user is asked if they would like to see the number of friends, number of followers or location of users. Finally, they are asked for the number of entries they would like to see. If they select “number of friends”, they will see the output presented in the figure result 1, which shows the friend count for users in descending order. For the output with caching, please refer to our full code. If they select “number of followers,” they get the output presented in result 2 and if they select “location of users” they get the output shown in result 3.

```
import time
user, myquery_friend, no_records = return_myquery_key()
start_time = time.time()
caching(user, myquery_friend)
print("--- %s seconds ---" % (time.time() - start_time))
```

```
Please enter whether you want user information or tweet information?:
-> For user information enter 1
-> For tweet information enter 0
1
What information would you like to see:
->  For number of friends of users enter 1
->  For number of followers of users enter 2
->  For location of users enter 3
1
By default 5 records will be displayed by descending order of the previous selection. How many records do you want to
display
-> Enter the number of your choice
-> else enter 5
3
Time taken when not in Cache
     user_id                user_name  user_friends_count
0   15210670  Harjinder Singh Kukreja             1436314
1  200583835             Tim Fargo 😷🚀                488112
2   35203319      🟣 Evan Kirstel $B2B                275322
--- 0.319537878036499 seconds ---
```

Result 1: Results for searching user number of friends.

```
import time
user, myquery_follower, no_records = return_myquery_key()
start_time = time.time()
caching(user, myquery_follower)
print("--- %s seconds ---" % (time.time() - start_time))
```

```
Please enter whether you want user information or tweet information?:
-> For user information enter 1
-> For tweet information enter 0
1
What information would you like to see:
->  For number of friends of users enter 1
->  For number of followers of users enter 2
->  For location of users enter 3
2
By default 5 records will be displayed by descending order of the previous selection. How many records do you want to
display
-> Enter the number of your choice
-> else enter 5
3
Time taken when not in Cache
    user_id       user_name  user_followers_count
0  18839785   Narendra Modi              67334826
1  37034483             NDTV              15052866
2  14159148  United Nations              13848616
--- 0.032950639724731445 seconds ---
```

Result 2: Result for searching user followers count

```
import time
user, myquery_loc, no_records = return_myquery_key()
start_time = time.time()
caching(user, myquery_loc)
print("--- %s seconds ---" % (time.time() - start_time))
```

```
Please enter whether you want user information or tweet information?:
-> For user information enter 1
-> For tweet information enter 0
1
What information would you like to see:
->  For number of friends of users enter 1
->  For number of followers of users enter 2
->  For location of users enter 3
3
By default 5 records will be displayed by descending order of the previous selection. How many records do you want to
display
-> Enter the number of your choice
-> else enter 5
3
Time taken when not in Cache
              user_id      user_name user_location
0  1290605502834135041  Crypto Dizzle        Europe
1  1362836734304800770      GRVIPER99        Europe
2            156776475   Beatriz Ríos        Europe
--- 0.12944889068603516 seconds ---
```

Result 3: Result for searching user location

If the application user selects tweet information, the application user is asked if they would like to search by hashtag, keyword, timeframe or the tweet id. Finally, they are asked for the number of entries they would like to see. If they choose to search by hashtag, they will be asked to enter a hashtag to search by and they will see the output shown in the figure called results 4. If the application user chooses to search by keyword, they will be asked to enter a keyword and they will see the output shown in the figure called results 5. If the user chooses to search by time frame, the user will be prompted to enter a start date and end date and they will see the output shown in

results 6. Finally, if the user chooses to search by tweet id, they will be prompted to enter a tweet id and they will see the output shown in results 7.

```
Please enter whether you want user information or tweet information?:
-> For user information enter 1
-> For tweet information enter 0
0
To search by
->  Hashtag enter 1
->  Keyword enter 2
->  Timeframe enter 3
->  Tweet_id enter 4
1
By default 5 records will be displayed. How many records do you want to display
-> Enter the number of your choice
-> else enter 5
3
Please enter a word: mask
Time taken when not in Cache
   index                       _id             tweet_id              user_id  \
0      0  6092c819648231edea74d854  1385591796747575300           2988288404
1      1  6092c843648231edea74dd58  1385598453049352194  1293906975227404290
2      2  6092c853648231edea74df85  1385599339544944647           4001651412

   tweet_favorite_count  tweet_retweet_count tweet_language tweet_timestamp  \
0                     0                    2             en   1619186936448
1                     0                    0             en   1619187389038
2                     0                    0             en   1619187600395

                                      tweet_text  \
0  going home currently on train from #pryj to #l...
1  new softer #covid19 rules in #valencia #spain....
2  #covid19 is not over yet. take necessary preca...

                                  tweet_hashtags     tweet_created_at
0        [pryj, ltt, mask, covid19, peoplethinking]  2021-04-23 13:50:02
1                  [covid19, valencia, spain, mask]  2021-04-23 14:16:29
2  [covid19, pulseoximeter, thermometer, vaporize...  2021-04-23 14:20:00
```

Result 4: Results for hashtag search searching by #mask

```
Please enter whether you want user information or tweet information?:
-> For user information enter 1
-> For tweet information enter 0
0
To search by
->  Hashtag enter 1
->  Keyword enter 2
->  Timeframe enter 3
->  Tweet_id enter 4
2
By default 5 records will be displayed. How many records do you want to display
-> Enter the number of your choice
-> else enter 5
3
Please enter a hashtag(exclude #) in lowcase: trump
Time taken when not in Cache
   index                       _id             tweet_id              user_id  \
0      0  6092c8a4648231edea74e4a2  1385589133637156874            172748155
1      1  6092c9e5648231edea74f87a  1385611817628536850  1256824219532316674
2      2  6092c80c648231edea74d55b  1385594367788867590             22027975

   tweet_favorite_count  tweet_retweet_count tweet_language tweet_timestamp  \
0                     3                    2             en   1619188169702
1                     0                    0             en   1619190575402
2                  1011                  242             en   1619186734805

                                       tweet_text  \
0  @kevinmkruse this is where trump first showed ...
1  'it's a real lifesaver': trump gives strongest...
2  trump's bleach news conference was one year ag...

                          tweet_hashtags     tweet_created_at
0                              [covid19]  2021-04-23 13:39:27
1  [potus, trump, covid19, coronavirus]  2021-04-23 15:09:35
2              [trump, bleach, covid19]  2021-04-23 14:00:15
```

Result 5: Results for searching by keyword Trump

```
Please enter whether you want user information or tweet information?:
-> For user information enter 1
-> For tweet information enter 0
0
To search by
->  Hashtag enter 1
->  Keyword enter 2
->  Timeframe enter 3
->  Tweet_id enter 4
3
By default 5 records will be displayed. How many records do you want to display
-> Enter the number of your choice
-> else enter 5
3
Please enter a start date(format:yyyy-mm-dd hh:mm:ss): 2021-03-01
Please enter a end date(format:yyyy-mm-dd hh:mm:ss): 2021-05-01
Time taken when not in Cache
   index                       _id             tweet_id              user_id  \
0      0  6092c988648231edea74f4a4  1366210500204314628             94637037
1      1  6092cb00648231edea750371  1366281411989233666  1183698123719200768
2      2  6092cb78648231edea75076d  1366301397541552134           4316815947

   tweet_favorite_count  tweet_retweet_count tweet_language tweet_timestamp  \
0                    76                   23             en   1619190062175
1                   420                   57             en   1619192235056
2                    19                   22             en   1619192780945

                                       tweet_text  \
0  vulnerable say they've been 'forgotten' as cov...
1  #hungary's prime minister viktor orban was vac...
2  in support of putting billions of human lives ...

                                  tweet_hashtags     tweet_created_at
0  [vaccinepriority, covid19, group6, mecfs, cfsm...  2021-03-01 02:15:40
1                      [hungary, covid19, chinese]  2021-03-01 06:57:27
2          [tripswaiver, covid19, nocovidmonopolies]  2021-03-01 08:16:52
```

Result 6: Results for searching by tweets between 03/01/21 and 05/01/21

```
Please enter whether you want user information or tweet information?:
-> For user information enter 1
-> For tweet information enter 0
0
To search by
->  Hashtag enter 1
->  Keyword enter 2
->  Timeframe enter 3
->  Tweet_id enter 4
4
By default 5 records will be displayed. How many records do you want to display
-> Enter the number of your choice
-> else enter 5
1
Please enter a tweet id: 1366301397541552134
Time taken when not in Cache
   index                       _id             tweet_id     user_id  \
0      0  6092cb78648231edea75076d  1366301397541552134  4316815947

   tweet_favorite_count  tweet_retweet_count tweet_language tweet_timestamp  \
0                    19                   22             en   1619192780945

                                       tweet_text  \
0  in support of putting billions of human lives ...

                             tweet_hashtags     tweet_created_at
0  [tripswaiver, covid19, nocovidmonopolies]  2021-03-01 08:16:52
```

Result 7: Results for searching by tweet id

We included caching with our search application using Redis. The application would first search the cache, which would store cached results for 60 seconds. If the search term was not in the cache, the application would search the databases to find the results, save the search term and results as a key value pair in the cache, and return the search results. Table 3 provides a breakdown of the time taken with and without caching for all 7 of the search application results shown and discussed earlier. We see a clear improvement in time when the results are retrieved from the cache instead of the databases.

Search Type	Time Taken Without Caching	Time Taken With Caching
Result 1: Search by user friends count	0.320 seconds	0.135 seconds
Result 2: Search by user followers count	0.033 seconds	0.014 seconds
Result 3: Search by user location	0.129 seconds	0.013 seconds
Result 4: Search by hashtag	0.247 seconds	0.085 seconds
Result 5: Search by keyword	0.128 seconds	0.030 seconds
Result 6: Search by time frame	0.775 seconds	0.032 seconds
Result 7: Search by tweet id	0.059 seconds	0.031 seconds

Table 3: Time taken to run search queries with and without caching

Concluding Remarks

This project gave us some insight into the complexities of building a search application, which is something we did not put much thought into before this class. Though searching from the database itself only took a fraction of a second, getting the results from the cache did visibly reduce the time taken to return the results. Generally, we are happy with the results we achieved, especially since we felt overwhelmed during parts of the project, but we think that our design can be improved by adding a GUI for our search application.

Referenes:

https://data-flair.training/blogs/mongodb-tutorial/
https://data-flair.training/blogs/mongodb-features/
https://data-flair.training/blogs/advantages-of-mongodb/
https://data-flair.training/blogs/mongodb-index/
https://www.xplenty.com/blog/postgresql-vs-mysql-which-one-is-better-for-your-use-case/