# TWEET SEARCH APPLICATION REPORT

## TEAM 4

Mayukh Sen
Brandon Salter
Max Jacobs
Divya Shah

## GITHUB REPOSITORY

https://github.com/brandonsalter/TwitterSearchApp_694_Team4_2024/

## GITHUB USERNAMES

Brandon Salter- brandonsalter

Max Jacobs- maxleejacobs

Divya Shah- divya370

Mayukh Sen- mayukhsen13

## ABSTRACT

This project aims to store and retrieve tweet/retweet data while optimizing time and space performance. The raw data is processed and transformed before being stored in different databases and tables. The databases are loaded by emulating a standard Extract, Transform, Load (ETL) procedure, streaming data into databases. MongoDB serves to store user data, while BigQuery manages the storage of tweet information. A client-side cache has been implemented to speed up repeated searches and enhance search result efficiency.

## INTRODUCTION

The architecture of this project primarily focuses on facilitating faster tweet retrieval from a massive JSON dataset of 101,916 tweets/retweet objects. Each tweet in the dataset consists of various components, such as the type of tweet (tweet, retweet, quoted tweet, etc.), user information (user name, screen name, favorite count, etc.), and post information (favorite count, retweet count, reply count, and quote count). The project aimed to segregate tweet/retweet storage into a relational datastore and user information into a NoSQL data source for easy information retrieval. Since BigQuery excels with relational operations, the first point of segregation was to use it to store all tweet and retweet information in related tables, encompassing around 80,943 unique users. Then, information from these tables can be efficiently combined and retrieved for user searches. NoSQL data stores allow storing data as a document, providing the freedom to accommodate different data types without pre-defining a schema. Therefore, MongoDB is being used to offer adaptability and flexibility in storing user information, such as screen names. The main components of this project are BigQuery data storage and retrieval, MongoDB data storage and retrieval, and caching.

## DATASET

In processing the corona-out-3 file, a virtual environment was created to support all project dependencies and was shared with team members via a remote repository. Subsequently, iterative acquisition of the following summary statistics for the file yielded: 101,916 total lines, 40,804 tweets, 61,112 retweets, 80,943 users, and 101,894 unique tweets/retweets. After these calculations, the JSON schema was explored by first printing out the last data item and then

selecting retweet and tweet sample objects. The field values associated with the ‘user’ key were also examined separately.

Upon initial analysis and exploration, the decision was made at a high level to separate the data based on tweets/retweets and user information. The tweet and retweet data were directed to separate tables within the relational database, BigQuery, while the user data were directed to the non-relational database, MongoDB. The specific keys chosen for each final data source were as follows:

Retweets:

- ['created_at'], ['entities']['hashtags'] (with some string manipulation), ['id_str'], ['text'], ['lang'], ['retweet_count'], ['user']['id_str'], ['retweeted_status']['id_str'],['retweeted_status']['user']['id_str']

Tweets:

- same as Retweets excluding the last two keys

Users

- ['created_at'], ['description'], ['favourites_count'], ['following'], ['friends_count'], ['id_str'], ['location'], ['name'], ['screen_name'], ['statuses_count'], ['verified']

Retweet data was distinguished from tweet data by the ‘text’ field starting with the string “RT”. It was made a point to avoid overlapping keys across databases so as not to place the onus on the developer to maintain consistency. The decision was made to have tweet/retweet data be relational initially because it would allow for arbitrary joins of retweet data onto tweet data and vice-versa. However, this functionality was not utilized as frequently as anticipated in later iterations of application development. The primary and index key of the Tweets table was chosen to be the ‘id_str_tweet’ key with no foreign key. A similar decision was made for the Retweets table, with ‘id_str_retweet’ being both the primary and index key, but with a foreign key of ‘id_str_tweet’. The relational schema between Tweets and Retweets tables is shown in the entity relationship diagram displayed in Figure 1. Likewise, user id_str was chosen as the index for our Users collection. The choices of index keys were made to optimize for searches based on IDs of tweets/retweets and users. As such, a tradeoff was made between ID and text-based searches.

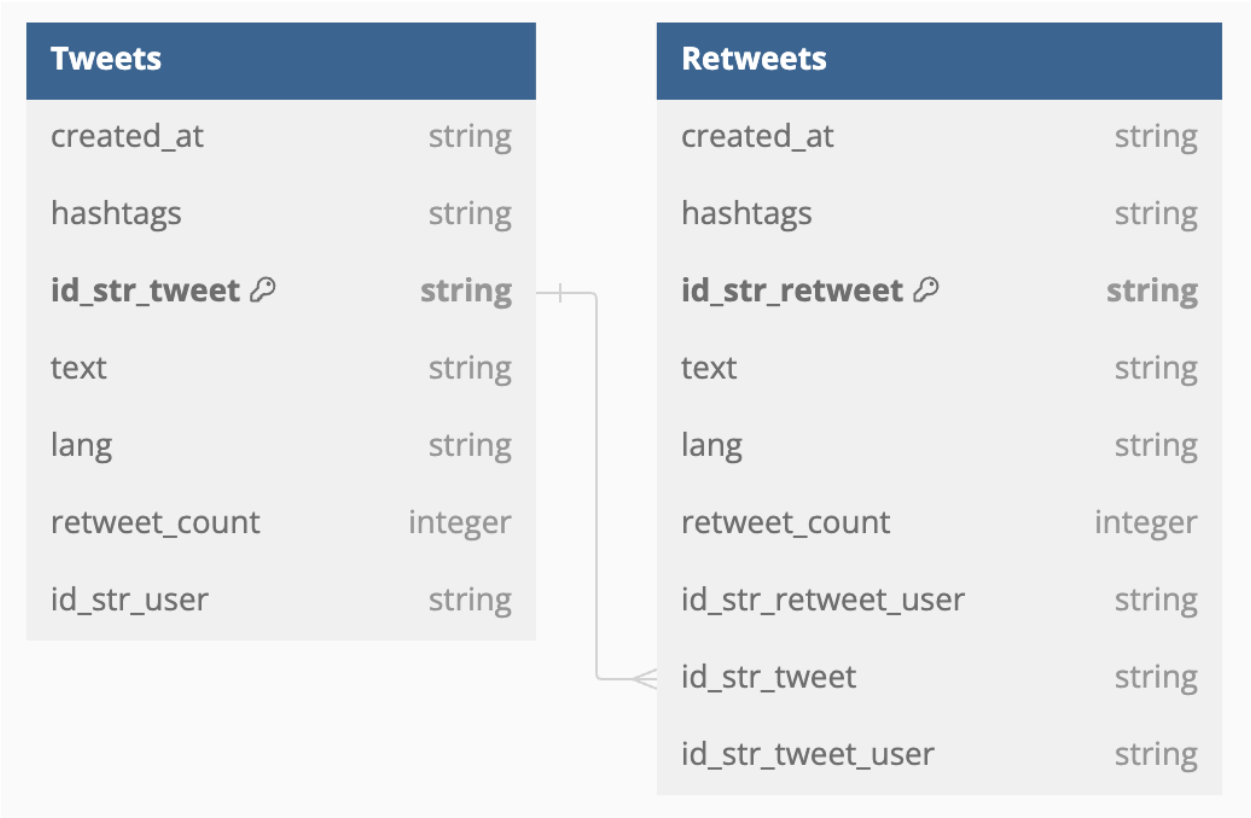


Figure 1: ERD between Tweet and Retweet BigQuery tables.
Made with dbdiagram.io

**PREPROCESSING**

To load the data into BigQuery, the initial strategy that was attempted was to iterate over the file and for each object in the file, check if the tweet/retweet ID had been processed and if not, then check if it is a tweet or retweet and take the appropriate keys. With these keys, a single row is written to its corresponding table within BigQuery. The problem encountered with this approach was that stream writes are not permitted by Google in the free sandbox tier. The workaround to this constraint was to rather than write the rows immediately to their final tables, instead first write them to in-memory DataFrames iteratively to retain the simulated streaming process. Similarly, upon encountering a retweet, the retweet count of the corresponding tweet was incremented by one if the tweet had been processed already, otherwise, the tweet's retweet count was initialized with a value of one. After iterating through the file, final retweet counts were appended and PyArrow was utilized with our specified schema along with the BigQuery Python API to load the DataFrames into BigQuery. This strategy was discussed with Professor

John and permitted, due to the encountered restrictions. Stream loading user data into MongoDB was able to be performed via the insert_one method of the PyMongo API, with pertinent key, value pairs. This resulted in a single document being added to the Users collection during each iteration over an object in the file only if that user had not yet been processed.

## TWEET SEARCH APPLICATION DESIGN

Tweets are stored in BigQuery to organize the text and metadata of tweets and retweets in separatable tables with linking keys and to optimize predictably expensive text-related reads as a continuous data stream would ultimately cause the warehouse to grow very large. BigQuery's columnar format optimizes queries to read only relevant columns instead of entire records, so a hypothetical user's search can be retrieved and returned at an acceptable speed in just a few seconds. Tweet search also queries the MongoDB user database to retrieve relevant user metadata such as username and follower count. The search application functions within a set of rigid rules in operation but maintains flexibility with regards to the conditions of the search; parameters for the search are obtained directly from the user via prompt by the *input( )* function. The user of the application can search for conditions on three attributes: words in a tweet, hashtags in a tweet, and number of retweets of a tweet. For each of these three attributes, the user is also given the option to specify the search based on date and time. Date and time can be specified as a single range or as a combination of range conditions joined by the logical operators **AND** or **OR**. The option to include no specified range in date and time is also available. The user can also choose to restart or exit the search at any time. A search request begins with manual, prompted inputs from the application user and is followed by a series of backend scripts to create the queries, execute the queries on both databases, retrieve relevant data, and display the search result to the application user. The high-level workflow can be described as a sequential process that is automated after receiving user input: fill an SQL query template with the user's specified conditions, run the query on the BigQuery warehouse to find tweets which satisfy those conditions, run a query on the MongoDB database to match the user IDs obtained from the first query with their corresponding records, and finally return the results as a python dataframe containing the author's username and follower count, tweet text, retweet count, date and time of tweet creation, the ID of the user who posted the tweet, and the ID of the tweet itself. Query building only affects the 'WHERE' statement of the SQL query, where the entire query itself is

stored as an f-string with the clause as a parameter. The clause structure is dependent on whether the user includes a date and time range but will always include one of the three base conditions. The clause parameter is created by iteratively joining the application user's specified condition inputs. After the query is run on BigQuery, the user IDs are extracted and passed to a function that performs a match search with ID for username and follower count on the user store in MongoDB using the MQL *collection.find( )* statement. The resulting data frame is transformed to include attributes from both queries before being displayed. The notion of relevance for search results is the number of retweets; all returned data frames are sorted by retweets descending (except for retweeting user exploration, as all retweet counts for retweets are 0). When searching by words in a tweet, the user can filter for a single word, multiple words, or any of the given words. Similarly, when searching by hashtags in a tweet, the user can filter for a single hashtag, multiple hashtags, or any of the given hashtags. When searching by number of retweets of a tweet, the user can filter for a single condition, multiple conditions, or any of the given conditions. If no matching tweets are found, the search application will notify the user and return to the original prompt.

Before any query is executed on the BigQuery tweet warehouse, the application checks the user's request against the cache. The cache is a locally stored dictionary, where keys are f-strings made up of the user's inputs and values are completed search result dataframes. If the combination of user inputs exists in the cache, it indicates that the same search has been performed and stored already; thus, the corresponding value dataframe is returned in nearly instant time without executing any queries. If the key does not exist, the queries are executed and the resulting data frame is stored in the cache. When searches are performed, the application also displays the collective time it took for the BigQuery and MongoDB queries to perform with a note of whether the search successfully hit the cache or otherwise executed the queries. After any search is returned, the user is given the option to drill down or explore the results further. As drill-down options, the user can show the top 10 most retweeted tweets or top 10 tweets by most followed users. For exploration, there is an option to find all the users who have retweeted a tweet or an option to find all the tweets from a particular author of a tweet. Search results will always return both a username and the tweet ID, so the application user can copy and paste the relevant attribute as prompted when exploring. To find all retweeting users, the search application requires the tweet ID string as input, and to find all tweets from a given user, the

search application requires the username string as input. Then, queries for BigQuery and MongoDB are built and executed similarly to the base three conditions using an f-string SQL template with the username or tweet ID as the parameter. As retweets cannot be retweeted, results returned from the retweeting users exploration feature are sorted by follower count as a notion of relevance. The search application is built to be circular, such that the hypothetical user can continuously execute queries at a high level and drill-down the results, explore information about users and retweets, and restart when no matching results are returned. The application also clears its output with each call to search, so the user can search and explore uninterrupted and with minimal visual clutter within the application environment.

## TWEET SEARCH RESULTS

Our Search application enables search by words in tweet, hashtags in tweet, and number of retweets. It also enables Search in Time Range- specify a single or multiple date and time ranges Figure 2a and Figure 2b shows the default user interface.

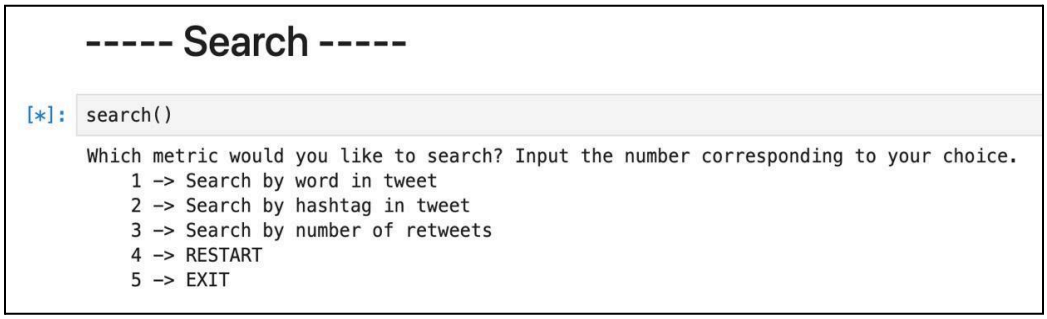


Figure 2a: Search Interface of application in Jupyter Notebook

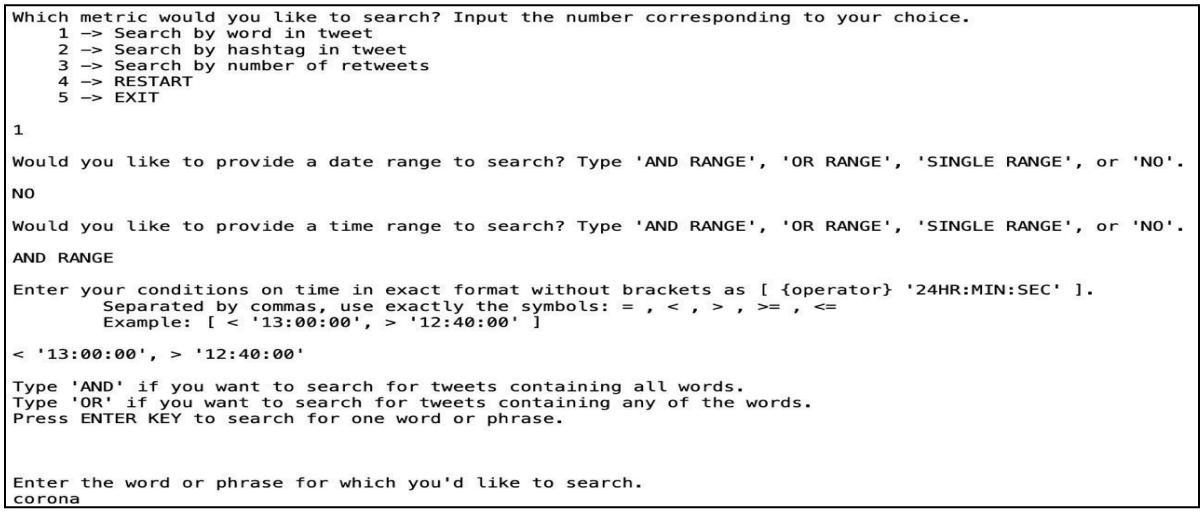


Figure 2b: Search Interface as search query progresses of application in Jupyter Notebook

Figure 3 displays a sample output in the application. Retweet count is default notion of relevance

```
################################################

Execution time without cache: 1.8991429805755615 seconds

################################################
Would you like to explore tweets further? Input the number corresponding to your search.
    1 -> Find all retweeting users
    2 -> Find all author's tweets
    3 -> Show top 10 tweets by retweet
    4 -> Show top 10 tweets by most followed user
    5 -> RESTART
    6 -> EXIT
```

| | username | followers | text | retweet_count | date | time | id_str_user | id_str_tweet |
|---|---|---|---|---|---|---|---|---|
| 0 | Aaron Mathew | 1219 | Modi ji our migrated working people are suffer... | 16 | 2020-04-25 | 12:53:52 | 1204476515993477120 | 1254030868152279042 |
| 1 | *ines*♡ | 203 | o verão q eu achava q ia ter antes do corona v... | 13 | 2020-04-25 | 12:45:48 | 807919391123849216 | 1254028839811981313 |
| 2 | Sözcü | 2398577 | CHP'li Özel: Soma corona virüsünde kırmızı ala... | 13 | 2020-04-25 | 12:57:26 | 218078497 | 1254031767906004992 |
| 3 | NTV Kültür Sanat | 508740 | Banksy'den İnci Küpeli Kız portresine corona g... | 12 | 2020-04-25 | 12:42:04 | 377964190 | 1254027898215956480 |
| 4 | Prerak Joshi | 1536 | #ગુજરાતનેપાછિસ્તાનબનતુઅટકાવો\n\nstopping coron... | 9 | 2020-04-25 | 12:59:03 | 152180683 | 1254032174052868102 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1074 | saBathembu | 2587 | @KwaneleZondih ngyeke yonke into ngzilele i co... | 0 | 2020-04-25 | 12:47:10 | 2421830228 | 1254029183904342016 |
| 1075 | Need24 | 29 | On March 25, there were 70 hospitals for coron... | 0 | 2020-04-25 | 12:47:57 | 1246284100995936256 | 1254029378998226946 |
| 1076 | Sach News / Flash News Live | 29 | India News: बीजेपी सांसदों का आरोप, पश्चिम बंग... | 0 | 2020-04-25 | 12:49:04 | 1176521388032577536 | 1254029660008169472 |
| 1077 | Sahan Dahanayake | 173 | #BREAKING\n\nAnother 2 positive corona patient... | 0 | 2020-04-25 | 12:59:22 | 1185895550111158272 | 1254032253732229122 |
| 1078 | Ravi Ranjan | 232 | @KapilMishra_IND Respect corona fighters\n ►►जय... | 0 | 2020-04-25 | 12:59:31 | 1052395370410475520 | 1254032292093321216 |

Figure 3: Search Results of a given query in Jupyter Notebook.

Figures 4a, 4b and Figure 4c show some drill-downs that are possible through our application.

```
Enter the username of the user for which you'd like to find all tweets.
Sözcü
```

| | username | followers | text | retweet_count | date | time | id_str_user | id_str_tweet |
|---|---|---|---|---|---|---|---|---|
| 0 | Sözcü | 2398577 | CHP'li Özel: Soma corona virüsünde kırmızı ala... | 13 | 2020-04-25 | 12:57:26 | 218078497 | 1254031767906004992 |
| 1 | Sözcü | 2398577 | SON DAKİKA \| Corona virüsünden etkilenen bir b... | 2 | 2020-04-25 | 14:41:46 | 218078497 | 1254058022160171009 |

Figure 4a: Search Results showing all tweets of a particular username

| | username | followers | text | retweet_count | date | time | id_str_user | id_str_tweet |
|---|---|---|---|---|---|---|---|---|
| 2 | Sözcü | 2398577 | CHP'li Özel: Soma corona virüsünde kırmızı ala... | 13 | 2020-04-25 | 12:57:26 | 218078497 | 1254031767906004992 |
| 3 | NTV Kültür Sanat | 508740 | Banksy'den İnci Küpeli Kız portresine corona g... | 12 | 2020-04-25 | 12:42:04 | 377964190 | 1254027898215956480 |
| 7 | Thanjavur Movies | 3942 | No new positive cases today in thanjavur...\nC... | 6 | 2020-04-25 | 12:42:16 | 1097954362339450880 | 1254027948295774208 |
| 5 | cindy | 3745 | adek nya umur 4 tahun positif corona 😭 ayah, i... | 8 | 2020-04-25 | 12:57:02 | 1014835874335375360 | 1254031665485103104 |
| 4 | Prerak Joshi | 1536 | #ગુજરાતનેપાછિસ્તાનબનતુઅટકાવો\n\nstopping coron... | 9 | 2020-04-25 | 12:59:03 | 152180683 | 1254032174052868102 |
| 0 | Aaron Mathew | 1219 | Modi ji our migrated working people are suffer... | 16 | 2020-04-25 | 12:53:52 | 1204476515993477120 | 1254030868152279042 |
| 1 | *ines*♡ | 203 | o verão q eu achava q ia ter antes do corona v... | 13 | 2020-04-25 | 12:45:48 | 807919391123849216 | 1254028839811981313 |
| 9 | Starwiz whizboi💙 | 137 | #if Wizkid x management Release \n#Made in Lag... | 5 | 2020-04-25 | 12:46:26 | 811632205541888000 | 1254028999455580161 |
| 6 | shrestha | 42 | #FightCoronaNotActivists \nUsing corona to thr... | 7 | 2020-04-25 | 12:42:57 | 291707503 | 1254028122846093313 |
| 8 | ntr_die_hard_fans_ | 12 | #SultanNTRBdayMonthCDP 33 Jai ntr .... corona ... | 6 | 2020-04-25 | 12:53:55 | 1243509705143508992 | 1254030880076476417 |

Figure 4b: Search Results ordered by the tweets of most followed users

```
Enter the id_str_tweet of the tweet for which you'd like to find all retweeting users.
1254031767906004992
```

| | username | followers | text | retweet_count | date | time | id_str_user | id_str_retweet |
|---|---|---|---|---|---|---|---|---|
| 0 | filiz Gülbay | 2165 | RT @gazetesozcu: CHP'li Özel: Soma corona virü... | 0 | 2020-04-25 | 12:58:04 | 1001104206810505216 | 1254031923728482305 |
| 8 | Aynur Yesilkaya | 2023 | RT @gazetesozcu: CHP'li Özel: Soma corona virü... | 0 | 2020-04-25 | 13:07:23 | 1477560440 | 1254034272069050368 |
| 2 | DURAN ADAM | 382 | RT @gazetesozcu: CHP'li Özel: Soma corona virü... | 0 | 2020-04-25 | 12:58:17 | 149844072 | 1254031979856756737 |
| 5 | Ceyhun Kotan | 313 | RT @gazetesozcu: CHP'li Özel: Soma corona virü... | 0 | 2020-04-25 | 13:02:29 | 993883851452833792 | 1254033037588865024 |
| 12 | Oğuz Bekemler | 118 | RT @gazetesozcu: CHP'li Özel: Soma corona virü... | 0 | 2020-04-25 | 14:03:50 | 217092652 | 1254048477140697088 |
| 6 | Ayşe Atılgan | 115 | RT @gazetesozcu: CHP'li Özel: Soma corona virü... | 0 | 2020-04-25 | 13:02:51 | 3297544665 | 1254033127359553537 |
| 9 | aquila | 76 | RT @gazetesozcu: CHP'li Özel: Soma corona virü... | 0 | 2020-04-25 | 13:07:36 | 755459572605784064 | 1254034322794872835 |
| 7 | Sevgi | 69 | RT @gazetesozcu: CHP'li Özel: Soma corona virü... | 0 | 2020-04-25 | 13:04:36 | 995680692397846528 | 1254033569946755072 |
| 1 | Kenan Kaya | 36 | RT @gazetesozcu: CHP'li Özel: Soma corona virü... | 0 | 2020-04-25 | 12:58:12 | 2893714467 | 1254031961083043842 |
| 10 | suna tezel | 35 | RT @gazetesozcu: CHP'li Özel: Soma corona virü... | 0 | 2020-04-25 | 13:09:43 | 1523047442 | 1254034858243997696 |
| 3 | filiz | 30 | RT @gazetesozcu: CHP'li Özel: Soma corona virü... | 0 | 2020-04-25 | 12:58:32 | 1145810064617496579 | 1254032041768882181 |
| 11 | Sonay Alpay | 26 | RT @gazetesozcu: CHP'li Özel: Soma corona virü... | 0 | 2020-04-25 | 13:23:39 | 1115904108194607104 | 1254038363801366529 |
| 4 | Kaan | 2 | RT @gazetesozcu: CHP'li Özel: Soma corona virü... | 0 | 2020-04-25 | 12:59:42 | 1246924512207212546 | 1254032338222186498 |

Figure 4c: Search Results showing all the retweeting users of a particular tweet

## CACHE

The cache database consists of a dictionary with a min-heap data structure. The dictionary's key is the search request, and the value is the search result. For achieving O(1) computation complexity, the dictionary is the only best choice. However, the dictionary data structure can't save information about the insertion order. To implement the feature of deleting the oldest record automatically, a min-heap with a maximum size of 50 is used to save the insertion order. The root node of the heap is the oldest searching query. Insertion and deletion operations would take O(n) = log(n). Since n is capped at 50, retrieval time is considered constant. Once the heap size reaches the limit set, a batch of the oldest record will be deleted, and the persistent system will write the heap and dictionary into the disk.

In Figure 5, the execution time without cache is observed to be approximately 1.9 seconds. Conversely, when results are retrieved from the cache, a mere 7×10^-6 seconds is required for their retrieval. This stark contrast underscores the functionality of the cache, as evidenced by the significant reduction in retrieval time displayed in Figure 5.

```
##############################################
Execution time without cache: 1.8991429805755615 seconds
##############################################
```

```
##############################################
Execution time with cache: 7.152557373046875e-06 seconds
##############################################
```

Figure 5: Time difference between execution with cache and without cache

Figure 6 outlines the cache flowchart, detailing the sequence from program initialization where the cache is populated. Initially, the system checks if the requested data exists in the cache. If not, it queries the databases for the information. The retrieved result is then presented to the user. After printing the result, subsequently, the system subsequently revisits the cache status. Should the cache be full, it evicts the oldest query to accommodate the new search query. The new search query is loaded onto the cache and the cache is saved.

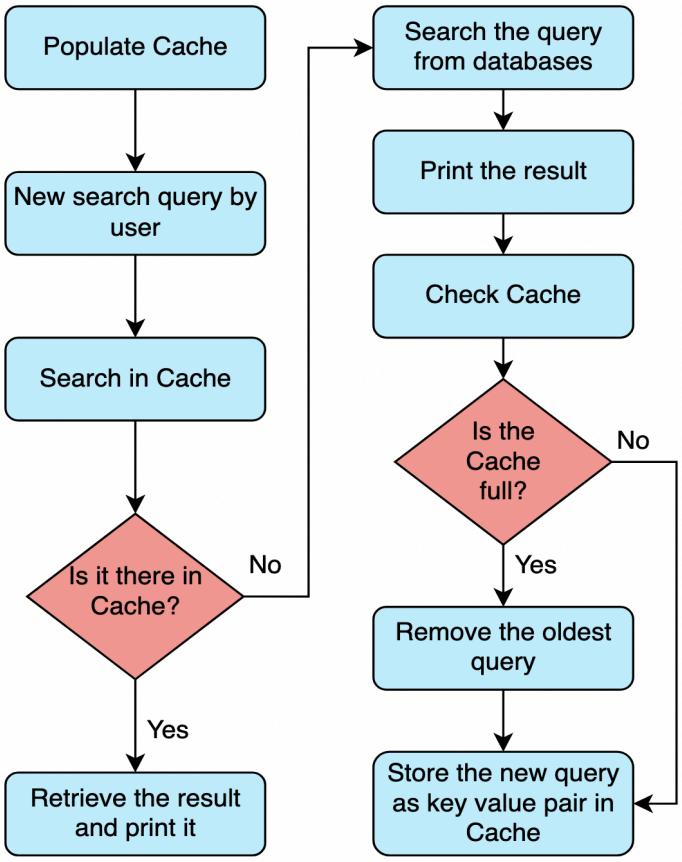


Figure 6: Cache flowchart
Made with draw.io

## CONCLUSION

This project efficiently managed to query and retrieve information from a large tweet dataset, using BigQuery for structured tweet data and MongoDB for flexible user data storage. The addition of a robust client-side cache significantly improved search speeds and system performance by a significant factor. The project empirically establishes the importance of having a cache to enhance the performance of data retrieval systems. The project highlighted the importance of strategic database selection. MongoDB and non-relational databases are extremely

powerful when their ability to tackle unstructured data is leveraged like storing user data in our case. Similarly, Relational databases can be very useful when a relationship and tabular form of data is to be exploited in a database like storing tweets and retweets in BigQuery in separate tables in the project. The ETL process and adherence to ACID properties ensured data integrity and robust management. This was particularly important as these are crucial factors that allow seamless scaling up of applications. The project helped us understand how key strategic decisions of database selection, streaming process, and data storing can affect the long-term performance and robustness of applications. The project enabled us to gain valuable experience in potential challenges that may arise in a data pipeline.

## REFERENCES


- OpenAI. (2024). ChatGPT (4) [Large language model]. https://chat.openai.com
- Big Query Documentation - https://cloud.google.com/bigquery/docs
- Lecture: Buffer Management
- Blog for streaming into MongoDB using Python https://www.mongodb.com/developer/languages/python/python-change-streams/
- Literature on MinHeap Cache - https://arxiv.org/pdf/2110.11602
- Dataset and Processing:
    - https://hex.tech/blog/connecting-bigquery-python/
    - https://www.mongodb.com/docs/drivers/pymongo/
    - https://cloud.google.com/bigquery/docs/write-api-batch
    - https://pymongo.readthedocs.io/en/stable/api/pymongo/collection.html


## PROJECT RESPONSIBILITIES

Initial dataset analysis and data preprocessing - Brandon Salter

Tweet search query building and input logic - Max Jacobs

Cache architecture and optimization - Divya Shah

MongoDB search functionality, Class structure, and environment setup - Mayukh Sen