



STAT 694

DATABASE MANAGEMENT FOR DATA SCIENCE

Instructor: Dr. Ajita John

---

## Twitter Search App

---

TEAM 6

Final Project

GitHub Repository <https://github.com/d-manishag/Twitter-Search-App.git>

*Authors:*

Vivek Reddy Chithari

Manisha Gayatri Damera

*NetID:*

vc508

md1723

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Dataset</b>	<b>2</b>
<b>3</b>	<b>Data Model and Data Stores</b>	<b>2</b>
3.1	Relational Database . . . . .	3
3.2	Non Relational Database . . . . .	3
3.3	Indexing . . . . .	3
3.3.1	MySQL Indexing . . . . .	4
3.3.2	MongoDB Indexing . . . . .	4
<b>4</b>	<b>Search Application Design</b>	<b>4</b>
4.1	Data Retrieval and Processing . . . . .	5
4.2	Search and Filter Options . . . . .	6
4.3	Query Translation . . . . .	8
4.4	Relevance and Result Ordering . . . . .	8
<b>5</b>	<b>Caching</b>	<b>9</b>
<b>6</b>	<b>Results</b>	<b>10</b>
<b>7</b>	<b>Conclusion</b>	<b>11</b>

## 1 Introduction

The present work is motivated by acknowledging that various database architectures handling volumes of data integrated to design search applications have great functionality. The work focuses on building a Twitter Search Application on the Twitter data. At the outset, the data is processed, and databases are designed for efficient data storage. The databases are consolidated with relevant interface design to develop a search application resembling Twitter.

## 2 Dataset

For designing the current application, data encoded using Java Script Object Notation, provided by Twitter, is used. The structure of the data for a tweet in the provided dataset is a dictionary containing various keys such as 'id', 'text', 'created\_at', 'source', 'retweet\_count', 'favorite\_count', 'lang', 'reply\_count', 'user\_id', and 'hashtags'. Additional keys may be added for retweets, quotes, extended\_tweet and replies. The summary of the tweets is as follows.

Objects	Count
Users	117795
Tweets	41529
Re-Tweets	72244
Quoted Tweets	8696
Replies	16132
Total No. of JSON objects	120434

Table 1: Summary Table of the Corona Tweets

```

Inserting tweets: 100%|██████████| 41529/41529 [00:12]
Inserting retweets: 100%|██████████| 72244/72244 [00:21]
Inserting quotes: 100%|██████████| 8696/8696 [00:02]
Inserting replies: 100%|██████████| 16132/16132 [00:04]
Data insertion completed.

```

Figure 1: Data Processing in Python

## 3 Data Model and Data Stores

Database design is an essential aspect of database management. The Twitter JSON document has a complex data structure, and our work implements a hybrid database system for easy access and retrieval of tweets and users. The information in a tweet is stored using MongoDB, a NoSQL database, and MySQL is used to store users' data. MySQL and MongoDB are installed in our systems. Python code is developed to interact with MongoDB and MySQL. The code uses the 'pymongo' library to interact with MongoDB and the 'mysql-connector-python' library to interact with MySQL.

### 3.1 Relational Database

The user-specific data from the JSON is retrieved, i.e., the user object in tweets with key-value pairs. User data is procured with user-id, username, user description, number of followers, and the date the account was created. This user data is encoded into new JSON, then retrieved for insertion into the table 'user\_table' in the user database created in MySQL. MySQL database is designed to store the users not only from the tweets JSON objects but also from the retweeted\_status, quoted\_status, and in\_reply\_to\_user resulting in a total of 117,795 unique users.

```
mysql> SELECT id, screen_name, LEFT(description, 40) AS truncated_description, followers_count, created_at, created_at_edt, created_at_epoch FROM users LIMIT 5;
```

id	screen_name	truncated_description	followers_count	created_at	created_at_edt	created_at_epoch
767	xeni	#STAYHOME. Healthy today 🙌. Support my	152608	Fri Jul 14 05:49:14 +0000 2006	Fri Jul 14 01:49:14 EDT 2006	1152856154
1991	clockwerks	C00 @voxmedia; work with our very best:	6091	Mon Jul 17 16:35:26 +0000 2006	Mon Jul 17 12:35:26 EDT 2006	1153154126
3987	Moe	NULL	NULL	NULL	NULL	NULL
5039	jrome	working on bright white teeth	642	Wed Aug 30 13:21:20 +0000 2006	Wed Aug 30 09:21:20 EDT 2006	1156944080
5242	ahmed	Reporter for @FT covering Saudi Arabia	149844	Mon Sep 04 00:18:12 +0000 2006	Sun Sep 03 20:18:12 EDT 2006	1157329092

5 rows in set (0.00 sec)

Figure 2: MySQL server - Relational Database with users' table

### 3.2 Non Relational Database

A complete tweet object looks complex with no fixed keys, as the 'retweeted\_status,' 'quoted\_status,' and others are unforced in every tweet. The data is handled with MongoDB, a non-relational database management system to overcome this. As the idea is to efficiently store the data for fast access, the data is stored in four collections in non-relational databases. MongoDB database is designed to store tweets, retweets, quoted tweets, and in\_replies. Each collection in the tweets database holds the relevant data. While processing the data the duplicates are supervised.

- tweets collection - The tweets are stored in the database with the tweet as the unique Primary key and having 'id', 'text', 'created\_at', 'source', 'retweet\_count', 'favorite\_count', 'lang', 'reply\_count', 'user\_id', 'hashtags'. The extended\_tweet is also added to the tweet text if present.
- retweets collection - Retweets are handled by checking if the 'retweeted\_status' key is present in the tweet object. If so, the code processes the retweet data and appends it to the 'retweets\_list'. The original tweet (i.e., the tweet being retweeted) is also processed and stored in the 'tweets\_list'. It also stores the original tweet user\_id and the tweet\_id.
- quoted\_tweets collection - Quoted tweets are in the JSON 'quoted\_status' array; if such tweets exist, they are stored in this database. The quoted\_tweets stores all the features of tweets alongside the original tweet id and its user id and name to interact with other databases and fetch the original tweeted text.
- in\_replies collection - This stores the user information, the in\_reply text, and the status.id and user\_id to which it is replied.

This MongoDB database keeps track of all the tweets at different levels and user ids, monitoring the conversations, tweets, and their reach.

### 3.3 Indexing

In this report, we discuss the indexing strategy employed for a search app utilizing both MySQL and MongoDB databases. Indexing helps improve the performance of query operations by creating an efficient

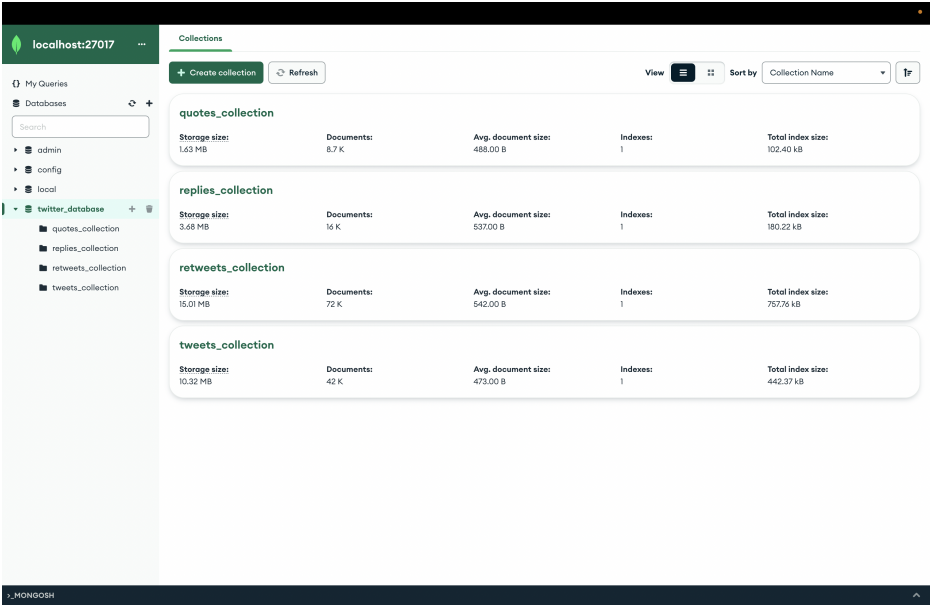


Figure 3: MongoDB server tweets processing compass

data structure to search for data in the database.

3.3.1 MySQL Indexing

In the users database in MySQL, indexing is created on the following fields.

MySQL Indexed Field	Purpose
'id'	Quickly search users by ID (e.g. tweets)
'screen_name', 'created_at_epoch', 'followers_count'	Speed up search, filter by date range and followers count, and sort by followers count
'followers_count'	Quickly sort users by followers count

Table 2: MySQL Index and Purpose

3.3.2 MongoDB Indexing

Indexing in MongoDB is as follows:

4 Search Application Design

The present developed Twitter Search App is based on the Streamlit application.

**Streamlit Python:** The Streamlit application, built using the Streamlit Python library, allows for the rapid development of interactive web applications. It enables the Twitter search application to connect

MongoDB Indexed Field	Purpose
'user_id' and 'created_at_edt'	efficiently filter and sort tweets by user ID and creation date (e.g. tweets)
'text', 'created_at_epoch', and 'retweet_count'	Speed up search, filter by date range and followers count, and sort by followers count
'retweet_count'	sort tweets by retweet count
'hashtags'	speed up the aggregation of hashtags:
'source'	speed up the aggregation of sources
hashtags', 'created_at_epoch', and 'retweet_count'	efficient hashtag search, date range filtering, and sorting by retweet count

Table 3: MongoDB Index and Purpose

to a data store containing information about Twitter users and their tweets. It then retrieves, processes, and displays the data based on user inputs, enabling users to explore various insights and trends.

## 4.1 Data Retrieval and Processing

### A. Retrieving data from the database

#### 1. Top users by followers

The `'get_top_users_by_followers()'` function retrieves the top users based on their follower count. This function establishes a connection to the database, executes a query to fetch the top users in descending order of their follower count, and returns a list of dictionaries containing user data. The query also has a limit parameter that controls the number of top users to fetch.

#### 2. Top tweets by metric

The `'get_top_tweets_by_metric()'` function retrieves the top tweets based on a specified metric, such as retweet count or the favourite count. This function accepts the metric as an argument and fetches the corresponding data from the `tweets_collection`. It employs the pymongo `'find()'` method to obtain the tweets and the `'sort()'` method to order the results based on the given metric in descending order. The function returns a list of dictionaries containing tweet data.

### B. Processing and displaying data

#### 1. Displaying top users

The `'display_top_users()'` function processes and displays the top users fetched by the `'get_top_users_by_followers()'` function. It iterates over the list of users, assigns a random avatar seed for each user, and generates an HTML representation of the user data using the `'generate_user_html()'` function.

#### 2. Displaying top tweets

The `'display_top_tweets()'` function processes and displays the top tweets obtained from the `'get_top_tweets_by_metric()'` function. The function retrieves the corresponding user information using the `'get_user_info_with_cache()'` function. The tweet data and user information are

then used to generate an HTML representation of the tweet using the 'generate\_tweet\_html()' function.

### 3. Displaying top hashtags

The 'display\_top\_hashtags()' function retrieves the top hashtags using the 'display\_top\_hashtags\_helper\_with...' function, which in turn calls the 'display\_top\_hashtags\_helper()' function. The function performs an aggregation operation on the tweets\_collection using a pipeline that unwinds the hashtags, groups them, sorts them in descending order based on their count, and limits the number of results. The top hashtags and their counts are then displayed.

### 4. Displaying top sources

The 'display\_top\_sources()' function retrieves the top sources by executing the 'display\_top\_sources\_helper\_with...' function, which calls the 'display\_top\_sources\_helper()' function. This function performs an aggregation operation on the tweets\_collection using a pipeline that groups the tweets by source, sorts them in descending order based on their count, and limits the number of results. The top sources and their counts are then displayed.

```
def get_user_info(user_id):
    try:
        cursor = connection.cursor()
        query = f"SELECT screen_name, description, created_at_edt, followers_count FROM users WHERE id = {user_id}"
        cursor.execute(query)
        result = cursor.fetchone()

        if result:
            return {
                "screen_name": result[0],
                "description": result[1],
                "created_at_edt": result[2],
                "followers_count": result[3]
            }
        else:
            return None

    except Error as e:
        print(f"The error '{e}' occurred")
        return None
```

Figure 4: Example of search query - function to retrieve user data from MySQL

## 4.2 Search and Filter Options

### A. Types of Searches Allowed

The application offers various search options to explore the data. Users can search within specific collections such as Users, Tweets, Retweets, Quoted tweets, Replies, and Hashtags. The application provides a sidebar with input fields for users to enter their desired search terms and choose a collection to search from.

### B. Drill-down Options

The application allows users to drill down and explore the data based on their selected search and filter criteria. The available drill-down options are:

#### 1. Users:

This option displays a list of users matching the search criteria. Users can click on a specific user to view more information about that user, including their tweets.

```

def generate_tweet_html(tweet, user):
    random_avatar = f"https://avatars.dicebear.com/api/avataaars/{tweet['avatar_seed']}.svg"
    tweet_html = f"""
    <a href="{user_id={tweet['user_id']}&avatar_seed={tweet['avatar_seed']}" style="text-decoration: none; color: inherit;">
        <div style="border: 1px solid #e1e8ed; border-radius: 4px; margin: 10px; padding: 10px; font-family: 'Helvetica Neue', Helvetica, #
            <div style="display: flex; align-items: center;">
                
                <div>
                    <strong>{user['screen_name']}</strong> @{tweet['user_id']} · {tweet['created_at_edt']}
                </div>
            </div>
            <p style="margin-top: 10px;">{tweet['text']}</p>
            <div style="display: flex; justify-content: space-between; color: #657786; margin-top: 10px;">
                <span>{tweet['reply_count']} 🗨 Replies</span>
                <span>{tweet['retweet_count']} 🔄 Retweets</span>
                <span>{tweet['favorite_count']} ♥ Likes</span>
                <span>{tweet['source']} 🔗</span>
                <span>{tweet['hashtags']} 💎 Hashtags</span>
            </div>
        </a>
    """
    return tweet_html

def generate_user_html(user, avatar_seed):
    random_avatar = f"https://avatars.dicebear.com/api/avataaars/{avatar_seed}.svg"
    user_html = f"""
    <a href="{user_id={user['id']}&avatar_seed={avatar_seed}" style="text-decoration: none; color: inherit;">
        <div style="border: 1px solid #e1e8ed; border-radius: 4px; margin: 10px; padding: 10px; font-family: 'Helvetica Neue', Helvetica, #
            <div style="display: flex; align-items: center;">
                
                <div>
                    <strong>{user['screen_name']}</strong> @{user['id']} · {user['created_at_edt']} · {user['followers_count']} Followers
                </div>
            </div>
            <p style="margin-top: 10px;">{user['description']}</p>
        </a>
    """
    return user_html

```

Figure 5: Rendering individual tweets and users using HTML

**2. Tweets:**

This option shows a list of tweets matching the search criteria, ordered by their retweet count. Users can click on a specific tweet to view more information about that user, including their tweets, quoted tweets and replies.

**3. Retweets:**

This option displays a list of retweeted tweets matching the search criteria, ordered by their retweet count. Users can click on a specific retweet to view more information about that user, including their tweets, quoted tweets and replies

**4. Quoted tweets:**

This option shows a list of quoted tweets matching the search criteria, ordered by their retweet count. Users can click on a specific quoted tweet to view more information about that user, including their tweets, quoted tweets and replies

**5. Replies:**

This option displays a list of reply tweets matching the search criteria, ordered by their retweet count. Users can click on a specific reply to view more information about that user, including their tweets, quoted tweets and replies

**6. Hashtags:** This option shows a list of tweets containing the searched hashtags, ordered by their retweet count. Users can click on a specific tweet to view more information about that user, including their tweets, quoted tweets and replies**C. Date Range Filtering:**



The application provides date range filtering, enabling users to narrow down their search results based on a specific time frame. Users can select start and end dates using the sidebar's date input fields. The application will display search results within the specified date range, allowing users to focus on data relevant to their desired time period.

### 4.3 Query Translation

#### A. Translating Search Queries into Database Queries

The application translates search queries into appropriate database queries to fetch relevant data from the underlying datastores. Depending on the collection filter selected by the user, the application constructs different queries using the search term and date range provided by the user. For instance, when searching within the "Hashtags" collection, the application uses regular expressions in the query to match the search term with the hashtags present in the tweets.

#### B. Examples of Query Translations

##### 1. Hashtags search:

When searching for a hashtag, the application constructs a query with the search term and date range. The query looks like this:

```
{
  'hashtags': {'$elemMatch': {'$regex': search_string, '$options': 'i'}},
  'created_at_epoch': {'$gte': date_range[0], '$lte': date_range[1]}
}
```

##### 2. Users search:

When searching for users, the application constructs a query with the search term and date range. The query looks like this:

```
{
  '$or': [
    {'screen_name': {'$regex': search_string, '$options': 'i'}},
    {'description': {'$regex': search_string, '$options': 'i'}}
  ],
  'created_at_epoch': {'$gte': date_range[0], '$lte': date_range[1]}
}
```

### 4.4 Relevance and Result Ordering

#### A. Defining Relevance in the Application

Relevance in the application is determined by the search term, collection filter, and date range specified by the user. The application considers results relevant if they match the search term and fall within the specified date range.

#### B. Ordering Search Results

The search results are ordered based on the collection filter and specific metric selected by the user. For example:

##### 1. Tweets/Retweets/Quoted Tweets/Replies:

When searching within the user selected filter collection, the results are ordered by retweet count in descending order.

##### 2. Users:

When searching for "Top Users by followers count", the results are ordered by the number of followers in descending order. For a specific user, the tweets are displayed in the decreasing order of created date.

## 5 Caching

**Cache implementation in the application:** The application employs a caching mechanism to enhance its performance and efficiency. It uses the LRU ‘cache’ to cache the results of various functions that involve retrieving data from the datastore or performing expensive computations.

### Code Overview:

The code defines a class `LRUCache` with the following methods:

- (a) `init(self, capacity, checkpoint_file, checkpoint_interval, ttl=None)`: Initializes the cache with the specified capacity, checkpoint file, checkpoint interval, and optional time-to-live (TTL).
- (b) `_evict(self)`: Evicts the least recently used item from the cache.
- (c) `get(self, key)`: Returns the value associated with the given key, updating its access time.
- (d) `put(self, key, value)`: Stores the key-value pair in the cache, evicting the least recently used item if the cache is full.
- (e) `load_cache(self)`: Loads the cache from the checkpoint file if it exists.
- (f) `checkpoint(self)`: Writes the cache to the checkpoint file.
- (g) `purge_stale_entries(self)`: Removes cache entries that have exceeded their TTL (if provided).
- (h) `start_periodic_checkpoint(self)`: Starts an infinite loop to periodically checkpoint the cache and purge stale entries based on the specified checkpoint interval.

### Key Features:

- (a) **Capacity:** The cache is initialized with a specific capacity, which determines the maximum number of items it can store.
- (b) **Persistence:** The cache can be persisted to a file through the use of checkpointing, which saves and loads the cache from a specified file.
- (c) **Automatic Checkpointing:** Checkpointing occurs periodically, as determined by the specified checkpoint interval.
- (d) **Time-to-live (TTL):** An optional TTL can be provided during cache initialization, which will cause entries to be removed if they have not been accessed within the specified duration.

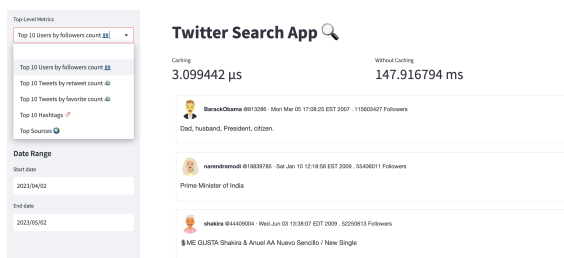
For instance, the ‘`get_top_users_by_followers_with_cache()`’ function utilizes caching to store and reuse the results of the ‘`get_top_users_by_followers()`’ function, which fetches the top users by follower count from the datastore. Similarly, caching is implemented in functions like ‘`get_top_tweets_by_metric_with_cache()`’, ‘`display_top_hashtags_helper_with_cache()`’, and ‘`display_top_sources_helper_with_cache()`’ to optimize their performance.

The caching mechanism also measures the time taken to retrieve results from the cache versus the time taken to perform the operation without caching. This allows for a comparison of response times, demonstrating the significant performance improvement the caching technique offers.

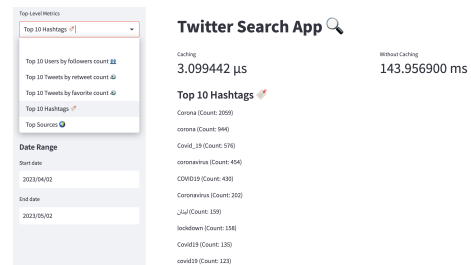
Implementing caching in the application ensures that the results of expensive operations are reused whenever possible, leading to faster response times, reduced computational overhead, and an overall better user experience.

## 6 Results

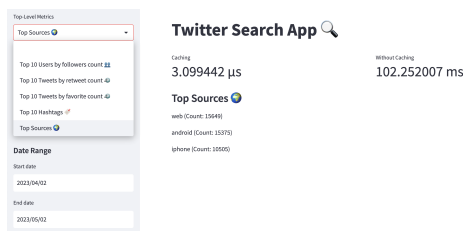
Figure 6 shows the results of search queries performed by the user on the search application, it includes with and without caching. It is noticed that the cached results and without cached results have a time difference of an order of 100 milli seconds or 100,000 micro seconds difference.



(a) Top 10 users by followers count



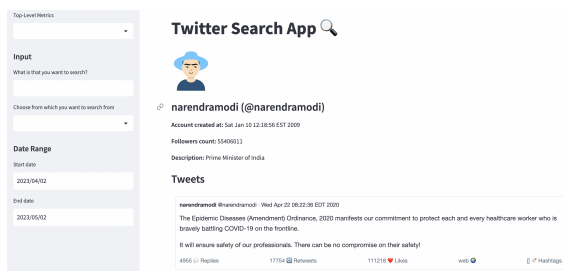
(b) Top 10 Hashtags by count



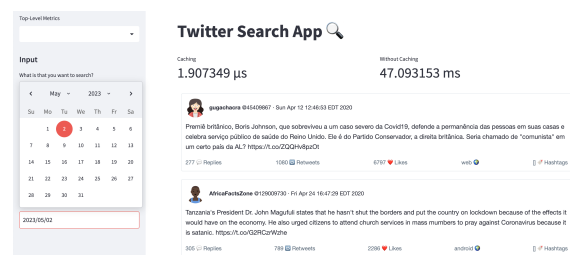
(c) Top Sources by their count



(d) Search for 'John' in Hashatgs



(e) Top Sources by their count



(f) Search for 'John' in Hashatgs

Figure 6: Screenshots of the results we obtain with caching

## 7 Conclusion

The Twitter search app design offers a user-friendly interface for searching and filtering data from a Twitter-like database. It supports various search options, drill-downs, and date range filtering to help users explore the data effectively. Our current design covers the salient features and brings inventiveness by using avatar and the layouts. The experimental framework leverages caching to improve performance and query execution times. Moreover, it defines relevance based on user-specified search criteria and orders the results accordingly. The implementation showcases an efficient way to search and explore data within a large datasets. Application also supports multiple search options, filters, and result ordering, the application caters to various user needs and preferences.

Building Twitter search app has been a fun and a big learning experience. Explored various database architectures handling volumes of data which are integrated to design search applications. Discovered the significance of Database Management Systems and Caching for Applications.