Twitter Search Application

16:954:694:01 Data Management for Advanced Data Science Applications (2024)

Rutgers University

Team 15

Hsiao-Chun Hung	hh617
Yuyue Sun	ys898
Yu Wang	yw1029
Zhuoer Liu	zl413

The URL of the GitHub repository:

https://github.com/yuwang1028/694_Team15_DBMS_2024

GitHub usernames for each team member:

- Hsiao-Chun Hung: hsiaochunhung
- Yu Wang: yuwang1028
- Zhuoer Liu: Jkjk1965
- Yuyue Sun: yuyuesun

ABSTRACT

The objective of this project is to design and develop a search application for a Twitter dataset utilizing both relational and non-relational databases. This application will enable users to efficiently search for tweets based on usernames, strings, and hashtags. Additionally, it will provide advanced features such as listing top 10 users by follower count. To accomplish this objective, the project encompasses several essential steps, including data collection, database design, cache implementation, and application development. Click this GitHub link to see more details for implementation.

INTRODUCTION

With social media being ubiquitous, Twitter stands as a prominent platform for real-time information exchange and communication. With millions of users generating a large flow of data daily, the need for efficient search applications becomes crucial. This project aims to address this need by designing and developing a search application tailored for a Twitter dataset. Leveraging both relational and non-relational databases, the application seeks to empower users with access to tweets based on various parameters such as usernames, strings, and hashtags.

Beyond basic search functionalities, the application sets itself apart by offering advanced features, including the ability to identify and showcase the top 10 users by follower count, as well as the top 10 tweets based on their retweet count. This additional functionality not only enhances user experience but also provides valuable insights into the Twitter landscape.

This project involves designing and implementing a sophisticated data management and search system for Twitter data, divided into several key components. Firstly, the project requires the acquisition and summary of a Twitter dataset, which is to be stored in both relational and non-relational databases to leverage the strengths of each storage type. Additionally, a Python-based caching mechanism will be implemented to enhance data retrieval speeds by storing frequently accessed data. The project also includes the creation of a search application that allows users to perform detailed searches on tweets, users, and hashtags, incorporating a range of query types and metrics. Lastly, the project culminates in a series of presentations and code repository that detail the system's performance, showcasing the practical applications and implications of the designed system.

DATASET

The dataset employed for the project is 'corona-out-3', consisting of 80,943 unique users and 101,894 unique tweets/retweets.

The stored data structure is as follows:

Tweet (in each JSON line)

- **created_at**: Timestamp of when the tweet was created.
- **id**: Unique identifier for the tweet.
- **text**: Content of the tweet.
- **source**: Source application of the tweet.
- **user**
  - **id**: User's unique identifier.
  - **name**: User's name.
  - **screen_name**: User's Twitter handle.
  - **followers_count**: Number of followers the user has.
  - **friends_count**: Number of users this user follows.
  - **statuses_count**: Number of tweets (including retweets) posted by the user.
- **retweeted_status**
  - **created_at**: Timestamp of the original tweet (for retweets).
  - **id**: Unique identifier for the original tweet.
  - **text**: Content of the original tweet.
  - **user**
    - **id**: Original tweeter's unique identifier.
    - **name**: Original tweeter's name.
    - **screen_name**: Original tweeter's Twitter handle.
  - **extended_tweet**
    - **full_text**: Full text of the original tweet if longer than standard tweet length.
- **entities**
  - **hashtags**: List of hashtags mentioned in the tweet.
  - **urls**: List of URLs mentioned in the tweet.
- **geo**: Geographical information of where the tweet was posted.
- **coordinates**: Coordinates of where the tweet was posted.
- **place**: Contains information about the place associated with the tweet.

PERSISTED DATA MODEL AND DATASTORES

Relational Datastore- MySQL

MySQL Database Connection Setup

Upon obtaining credentials from the MySQL database on Google Cloud, a custom utility function, `connect_to_mysql()`, was developed to establish a connection with the MySQL database. This function is stored in the `utils.py` file. Additionally, private APIs were authorized for each team member to ensure a seamless connection to the database.

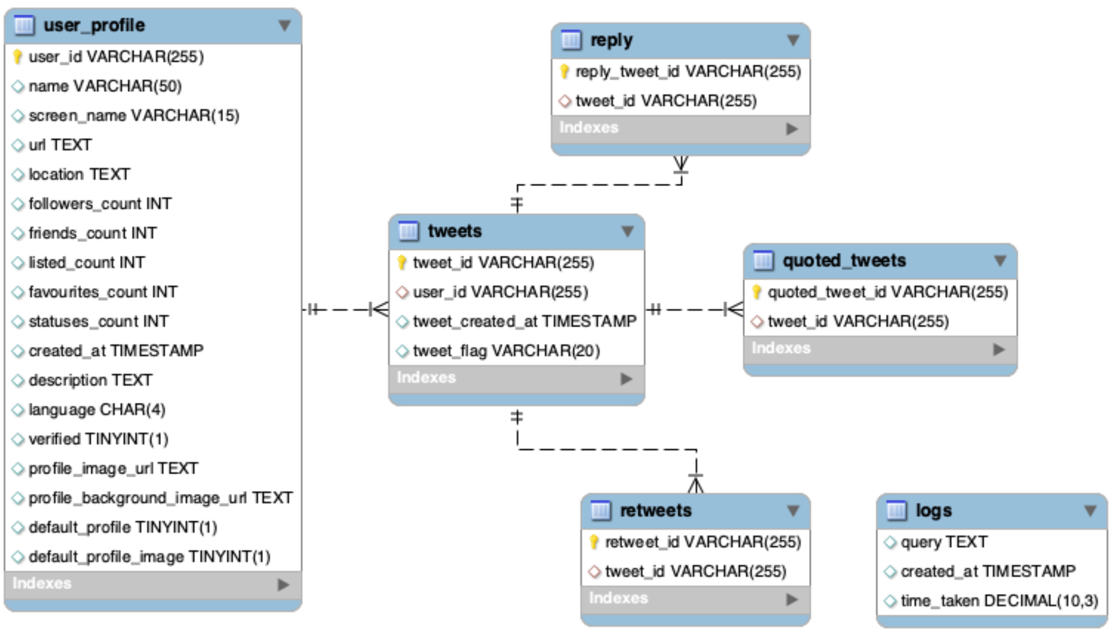


Figure 1: MySQL table schema

MySQL Data Model

Figure 1shows EER diagram from MySQL database, with one-many relationships between tables.

1) **User Profiles**: This table stores details about Twitter users such as user ID, name, screen name, profile URLs, location, follower and friend counts, and other relevant metadata.

2) **Tweets**: This table records each tweet's ID, associated user ID, creation timestamp, and a flag indicating if the tweet is original or a retweet.

3) **Replies**: This table captures information about replies to tweets, linking reply tweet IDs to the original tweet IDs.

4) **Quoted Tweets**: Like the Replies table, this table captures quoted tweets, linking quoted tweet IDs to original tweets.

5) **Retweets**: This table records retweet relationships by linking retweet IDs to the original tweet IDs.

6) **Logs**: A log of operations or queries is maintained in this table, tracking the SQL query, its execution time, and timestamp.

MySQL Optimization and Tradeoffs

The relational model of MySQL efficiently supports operations like joins and transactions, which are crucial for querying interconnected data like tweets and user data. MySQL requires a predefined schema, which can limit flexibility in handling dynamic data structures that might evolve with new types of data.

Indexing

An index is explicitly created on the `tweet_id` column of the `tweets` table (`idx_tweet_id`) to enhance query performance, particularly for operations that frequently access this column.

Non-Relational Datastore - MongoDB

MongoDB Database Connection Setup

First get MongoDB Atlas credentials after registering for a MongoDB Cloud account on the cloud service provider AWS. The MongoDB connection is managed through a custom utility function `connect_to_mongodb()` defined in the `utils.py` file. This function securely handles the database connections using credentials stored in the `config.ini` file, ensuring that the connection details remain secure and are not hard coded into the application. For team-wide access, private APIs and specific roles can be configured to control database access securely, facilitating both security and ease of use for all team members.

Storage of Data

Data is stored in MongoDB, a NoSQL document database that provides flexibility and is well-suited for handling semi-structured data like tweets. Each tweet is represented as a document in a collection, which in this application is set up within the **nonrelational** collection of the **twitter** database. Table 1: Processing of data into MongoDB displays the selection of fields processed into the NoSQL database.

Table 1: Processing of data into MongoDB

Category	Field in Original Data	Included in Processing?	NOTES
Tweet	created_at	Yes	Timestamp when the tweet was created
	id	No	Unique identifier, replaced with id_str
	text	Yes	Content of the tweet
	source	Yes	Source application of the tweet
User	id	Yes (as user_id)	User's unique identifier, processed as user_id
	name	No	User's name, not included
	screen_name	No	User's Twitter handle, not included

	followers_count	No	Number of followers, not included
	friends_count	No	Number of users followed, not included
	statuses_count	No	Number of tweets posted, not included
Retweeted status	created_at	No	Timestamp of the original tweet, not included
	id	No	Unique identifier of the original tweet, not included
	text	No	Content of the original tweet, not included
	user	No	Original tweeter's details, not included
Extended tweet	full_text	No	Full text of extended tweets, not included
Entities	hashtags	Yes (within entities)	Included within the JSON serialized entities field
	urls	Yes (within entities)	Included within the JSON serialized entities field
	geo	No	Geographical information, not included
	coordinates	No	Coordinates of the tweet, not included
	place	Yes	Information about the place associated with the tweet
Additional fields	contributors	Yes	Contributors to the tweet
	truncated	Yes	Whether the tweet is truncated
	lang	Yes	Language of the tweet
	quote_count	Yes	Number of times the tweet has been quoted

	reply_count	Yes	Number of replies
	retweet_count	Yes	Number of retweets
	favorite_count	Yes	Number of favorites
	favorited	Yes	Whether the tweet has been favorited
	retweeted	Yes	Whether the tweet has been retweeted
	possibly_sensitive	Yes	Whether the tweet may contain sensitive content
	withheld_in_countries	Yes	Countries where the tweet is withheld
	extended_entities	Yes	JSON serialized extended media entities
	quoted_status	Yes	JSON serialized quoted tweet
	retweeted_status	Yes	JSON serialized original tweet in case of a retweet

Indexing

A unique index is created on the `id_str` field to prevent duplicate entries and improve search performance when querying specific tweets based on their unique ID.

Optimizations and Tradeoffs

JSON Serialization: Certain fields such as `entities, extended_entities, quoted_status,` and `retweeted_status` are stored as serialized JSON strings. This approach simplifies the schema but at the cost of making queries on these fields slower since they need to be deserialized for processing.

Index on `id_str` Only: The primary index is on the `id_str`, optimizing retrieval by tweet ID. However, this means queries on other attributes (e.g., `user_id` or `created_at`) might not be as efficient, potentially requiring full collection scans unless additional indexes are created based on usage patterns.

PROCESSING TWEETS FOR STORING IN DATASTORES

Designing techniques

Efficiency in Data Insertion: Through the utilization of batch processing and the verification of duplicates prior to insertion, the script effectively minimizes unnecessary database operations.

Handling Large Datasets: A mechanism was incorporated to bypass previously processed lines, serving as a valuable tool for resuming interrupted operations without necessitating a complete restart.

MySQL:

For MySQL, the Python function `pushMySQLData` manages the ingestion of tweet data from a file, taking special care to avoid duplications with the use of `seen_users` and `seen_tweets` sets that track processed entries. It uses SQL's `ON DUPLICATE KEY UPDATE` clause to efficiently handle potential duplicate entries without failing. This function not only stores tweet and user data but also handles related data types like retweets, replies, and quoted tweets. Special attention is given to transforming Twitter-specific date formats to SQL-compatible ones, ensuring data consistency. Error handling is meticulously implemented, capturing exceptions related to JSON parsing and SQL operations, which prevents the process from halting abruptly due to corrupt data or database errors.

MongoDB:

On the MongoDB side, the function `insert_data_from_file` demonstrates a streamlined approach to inserting tweets into a MongoDB collection, which has been set up with unique indices to prevent duplicates. This function handles large datasets with a timeout feature that stops the data ingestion process if it exceeds a specified duration, ensuring that the system remains responsive. It processes each line of the input file as an individual document, converting JSON into MongoDB documents, and tracking the insertion of documents with periodic status updates and duplicate count logging. This method is efficient for handling large-scale data ingestion where document uniqueness is crucial.

CACHING

The CacheManager implemented in the Twitter search app substantially reduces latency by avoiding redundant database queries, thus providing rapid data access.

System Design

Initiated with the CacheManager class, the LRU cache mechanism maintains up to 1024 query results. This approach balances performance with memory usage. The system intelligently preloads the cache from diskCache.json at start-up, ensuring data is readily available and persistent across sessions.

Query Processing

Queries are checked against the cache first, with hits served directly from it, enhancing speed, and reducing server load. Misses trigger data retrieval from databases, followed by cache storage via putQuery, which also manages cache size by evicting the oldest entries.

Maintenance and Efficiency

Entries are removed as needed using delQuery, and regular cache snapshots are taken for data integrity and recovery. The cache saving process is asynchronous, ensuring application performance remains unaffected by background data writing tasks.
The caching strategy employed enhances user experience by delivering quick responses and optimizes resource use, thereby demonstrating the efficacy of efficient cache management in web applications.

SEARCH APPLICATION DESIGN

The search application supports multiple search functionalities, including:

- **User Information:**
    - Search by **username**: Searches for tweets from users whose names match or contain the specified string (supports wildcards using **%**).
    - Search by **userscreenname**: Searches for tweets from users whose screen names match or contain the specified string (also supports wildcards).
- **User Verification Status:**
    - Search for tweets from all users, only verified users, or only non-verified users, as specified by **userverification**.
- **Tweet Content:**
    - Search by **tweetstring**: Searches for tweets containing a specific string in the tweet text.
    - Search by **hashtags**: Searches for tweets containing one or more specific hashtags.
- **Tweet Attributes:**
    - Search by **tweetsensitivity**: Searches for tweets that are marked as sensitive or not.
    - Search by **tweetcontenttype**: Searches for tweets containing media or not.
- **Date Range:**
    - Search within a specific start and end datetime range.
- **Interaction-based Searches:**
    - Search for retweets of a specific tweet (by `tweet_id`).
    - Search for replies to a specific tweet (by `tweet_id`).
    - Search for quoted tweets of a specific tweet (by `tweet_id`).
- **User-based Search:**
    - Fetches tweet metadata and user data for a specific user based on user_id.
- **Top Users by Followers (`top_users_by_followers function`):**
    - Searches SQL Database: Fetches the top 10 user profiles ordered by their follower count in descending order.

    - Query Focus: This search targets user-centric data and is specifically designed to identify the most influential users on the platform based on the number of followers.
- **Top Tweets by Retweets (`top_tweets_by_retweets` function):**
    - Hybrid SQL and MongoDB Search:
        - SQL Part: Retrieves tweet IDs along with their retweet counts from the SQL database. This involves aggregating data in the retweets table to calculate the total number of retweets for each tweet and then selecting the top tweets based on these counts.
        - MongoDB Part: Uses the tweet IDs obtained from the SQL query to fetch detailed tweet data from MongoDB, including the tweet text, hashtags, sensitivity flags, media types, and media URLs.

The search application incorporates multiple Drill-Down search features, including:

- **Temporal Drill-Down:**
    - The system can filter results within a specific time frame based on the `start_datetime` and `end_datetime`.
- **Content-Based Drill-Down:**
    - Drill-down into tweets that match specific content criteria, such as a phrase or hashtag.
- **User-Based Drill-Down:**
    - The system allows drilling down into data based on user-related attributes, such as whether the user is verified.
- **Sensitivity and Media Drill-Down:**
    - Drill-down based on whether a tweet is marked as sensitive and whether it contains media, offering a more nuanced view of the content.
- **Composite Drill-Downs:**
    - The system supports composite drill-downs where multiple criteria can be applied simultaneously. For instance, one could look for non-sensitive tweets containing a specific hashtag from verified users within a given date range.
- **Interaction Drill-Down:**

  - Users can drill down into interaction data for a particular tweet, obtaining IDs for retweets, replies, and quotes.
- **Composite Drill-Downs:**
  - Users can perform a compound search where they obtain metadata, user information, and interaction data (retweets, replies, and quotes) related to a set of tweets filtered by IDs.
- **Type-based Drill-Down:**
  - Segregates fetched tweets by type: original, quoted, retweeted, and replies.
- **Drill-Down by User Metrics:**
  - Specific to Users: The system allows for examining users with the highest engagement (as measured by followers), which can be critical for marketing strategies or influence analysis.
- **Drill-Down by Tweet Metrics:**
  - Engagement Metrics: For tweets, the system focuses on retweet counts, allowing an analysis of which tweets have garnered the most attention and may be considered "viral."
  - Content Specifics: By fetching additional tweet details from MongoDB, the system allows for a deeper look into the content that performs well on the platform. This includes examining the hashtags used, the presence of media, and sensitivity settings, which can offer insights into content strategy.

The search queries were converted into datastores queries through the following steps:

- **Building the SQL Query:**

Dynamic Time Filtering: Converts `start_datetime` and `end_datetime` from a string format to a SQL-compatible datetime format to filter records within a specific timeframe.

Conditional User Filters: Adds conditions based on provided username, userscreenname, and userverification:

Username and userscreenname conditions utilize a LIKE clause for partial matching, which is case-insensitive.

User verification explicitly checks for boolean conditions (TRUE for verified users, FALSE otherwise).

Tweet ID Filtering: If `filtered_tweet_ids` are provided, they are included in the query to fetch specific tweets.

Nested Subqueries: The SQL includes subqueries to count related retweets, quoted tweets, and replies, enriching the data with engagement metrics directly in the query output.

- **Query Execution:**

The query is executed using the `pandas.read_sql_query` function, which directly converts the SQL output into a DataFrame, making it ready for further processing or merging with NoSQL data.

The transformation of search queries into MongoDB queries occurred via the following steps:

- **Building the MongoDB Query:**

Text and Hashtag Filtering: Utilizes regex for tweetstring for flexible text matching and an $in filter for hashtags to match any of the specified tags.

Sensitivity and Media Filters: Adjusts the query based on the `tweetsensitivity` and `tweetcontenttype` to filter tweets by their sensitivity status and presence of media.

Datetime and ID Filters: Though commented out, there's a structure to include date range filters directly in the MongoDB query using $gte and $lte. The provided `filtered_tweet_ids` are used to fetch specific tweets via the $in operator.

- **Query Execution:**

Executes the MongoDB query using the find method, which retrieves a cursor to the result set that is then converted into a DataFrame.

The concept of relevance and the method of displaying search query results are outlined in the following steps:

- **SQL Data Ordering:**

The relevance of the data fetched from the SQL database (`fetch_searched_tweet_metadata_user_data`) is not explicitly defined in the provided SQL query beyond the datetime constraints. The data is fetched based on user-defined parameters such as username, userscreenname, user verification status, and specific datetime ranges. If the query included an ORDER BY clause, it would specify what the system considers most relevant (e.g., ordering by `retweet_count` descending could prioritize tweets with the highest engagement).

- **MongoDB Data Fetching:**

The MongoDB query in `fetch_searched_tweets_data` does not include a sorting mechanism within the query itself. The results are fetched based purely on content matches such as text, hashtags, sensitivity, and media presence without specifying an order. The implication here is that all results that match the criteria are equally relevant, or external sorting is expected post-fetch.

Results Merging and Final Ordering: After fetching data from both SQL and MongoDB, the final relevance in fetch_results when merging (pd.merge) the data is determined by the order of the `tweet_id`. Typically, the ordering in the final DataFrame would follow the order of IDs as they appear in the dataset unless a specific sorting mechanism (`sort_values`) is applied after merging. The lack of explicit post-merge sorting suggests that the merged results are not ordered by any specific engagement metric by default.

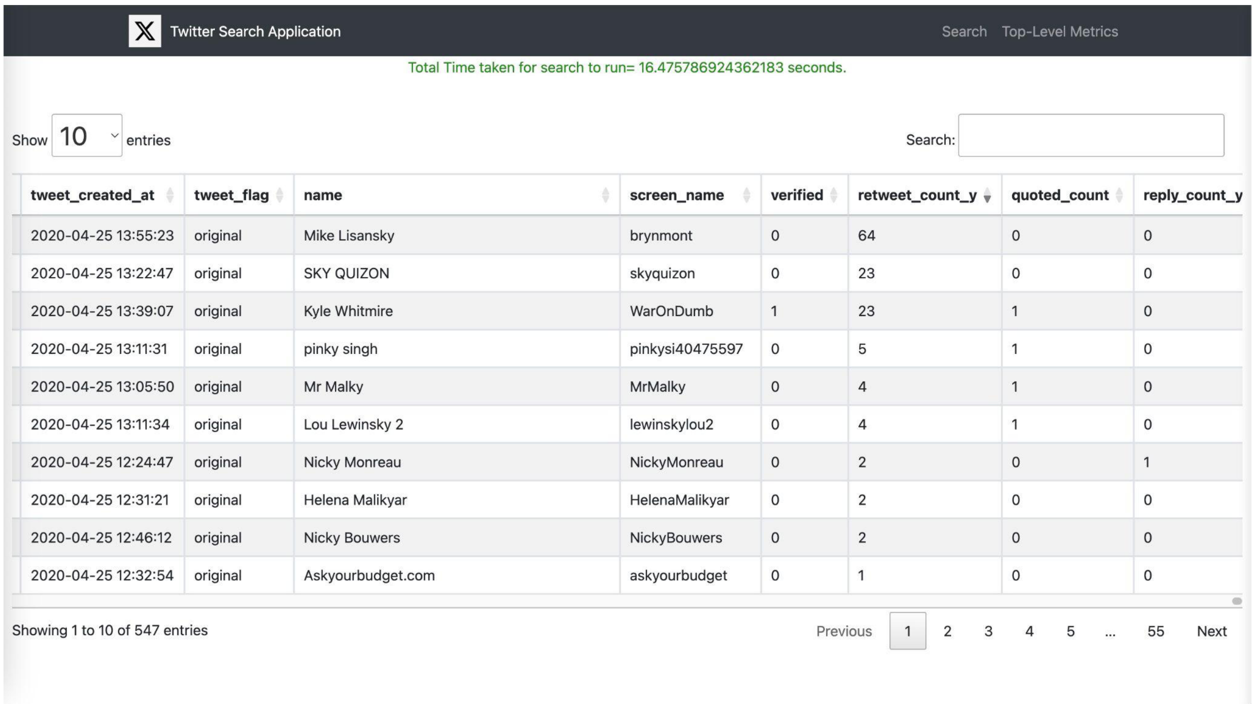
- **Cache Mechanism:**

The cache does not alter the notion of relevance but ensures that once the results are computed, they are stored and retrieved efficiently for repeated queries. This mechanism ensures speed and efficiency without recalculating results but does not impact how results are ordered.

RESULTS

Below are examples from Figure 2-5, demonstrating potential searches and rankings on the Search Application Website.

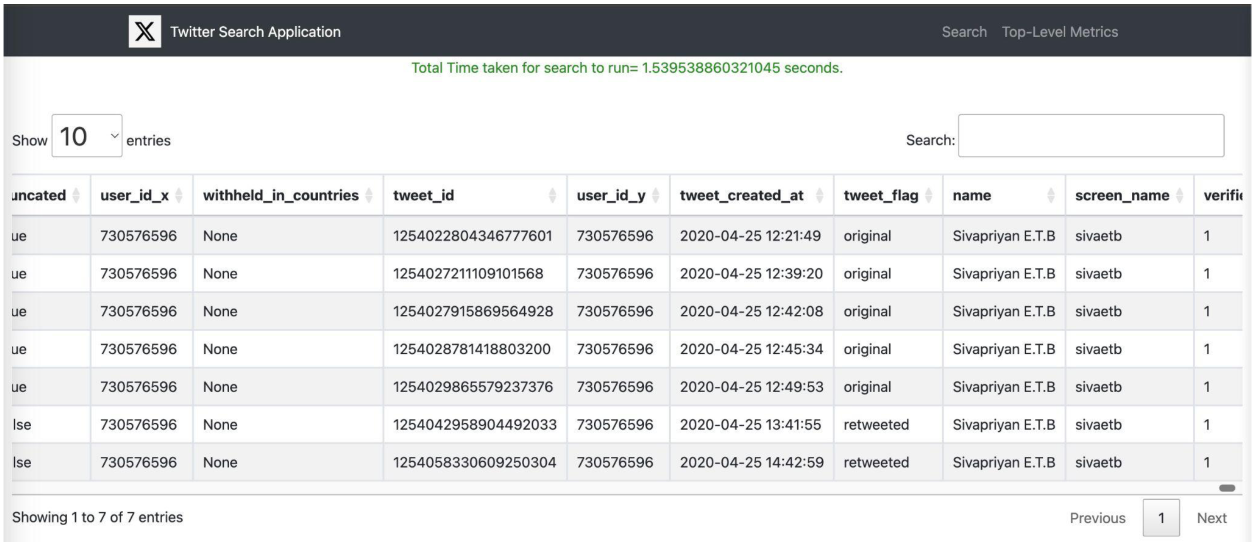
Search by names containing 'ky' (ranked by retweet_count), time taken 16.48 seconds (non-cached data):



tweet_created_at	tweet_flag	name	screen_name	verified	retweet_count_y	quoted_count	reply_count_y
2020-04-25 13:55:23	original	Mike Lisansky	brynmont	0	64	0	0
2020-04-25 13:22:47	original	SKY QUIZON	skyquizon	0	23	0	0
2020-04-25 13:39:07	original	Kyle Whitmire	WarOnDumb	1	23	1	0
2020-04-25 13:11:31	original	pinky singh	pinkysi40475597	0	5	1	0
2020-04-25 13:05:50	original	Mr Malky	MrMalky	0	4	1	0
2020-04-25 13:11:34	original	Lou Lewinsky 2	lewinskylou2	0	4	1	0
2020-04-25 12:24:47	original	Nicky Monreau	NickyMonreau	0	2	0	1
2020-04-25 12:31:21	original	Helena Malikyar	HelenaMalikyar	0	2	0	0
2020-04-25 12:46:12	original	Nicky Bouwers	NickyBouwers	0	2	0	0
2020-04-25 12:32:54	original	Askyourbudget.com	askyourbudget	0	1	0	0

Figure 2 Search by names containing 'ky'

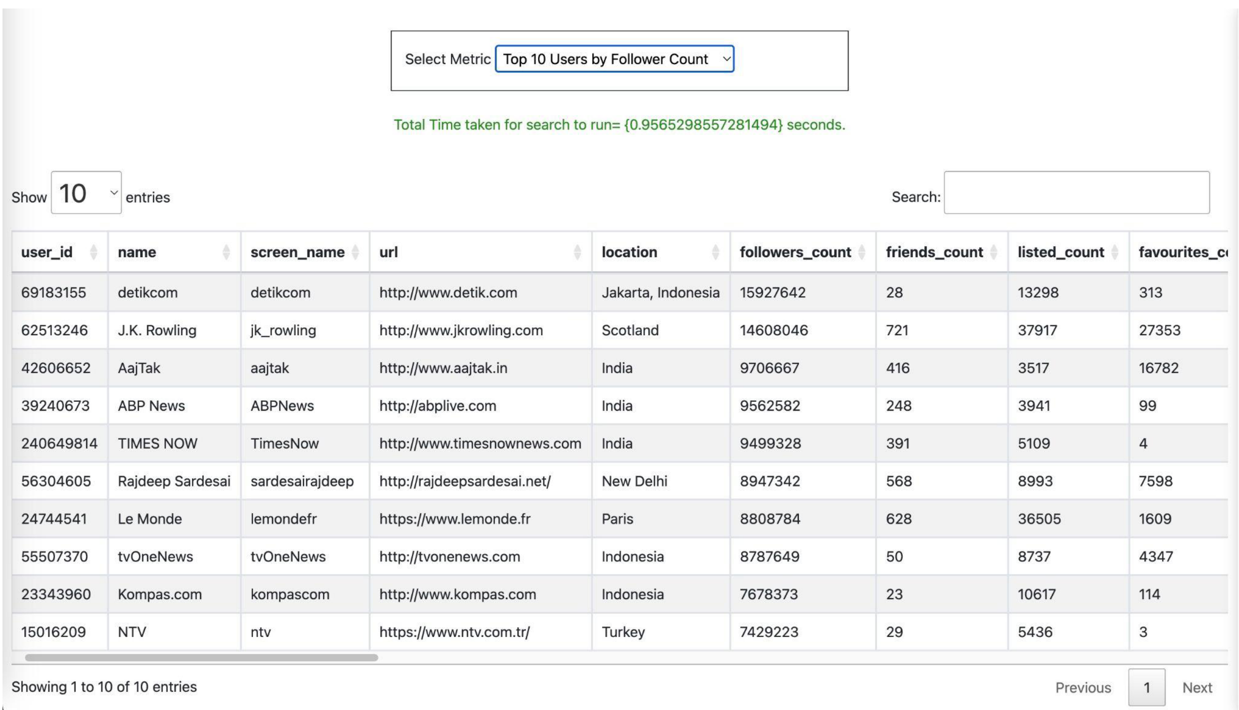
Search by screen_name 'sivaetb', time taken 1.54 seconds (non-cached data):



uncated	user_id_x	withheld_in_countries	tweet_id	user_id_y	tweet_created_at	tweet_flag	name	screen_name	verifi
ue	730576596	None	1254022804346777601	730576596	2020-04-25 12:21:49	original	Sivapriyan E.T.B	sivaetb	1
ue	730576596	None	1254027211109101568	730576596	2020-04-25 12:39:20	original	Sivapriyan E.T.B	sivaetb	1
ue	730576596	None	1254027915869564928	730576596	2020-04-25 12:42:08	original	Sivapriyan E.T.B	sivaetb	1
ue	730576596	None	1254028781418803200	730576596	2020-04-25 12:45:34	original	Sivapriyan E.T.B	sivaetb	1
ue	730576596	None	1254029865579237376	730576596	2020-04-25 12:49:53	original	Sivapriyan E.T.B	sivaetb	1
lse	730576596	None	1254042958904492033	730576596	2020-04-25 13:41:55	retweeted	Sivapriyan E.T.B	sivaetb	1
lse	730576596	None	1254058330609250304	730576596	2020-04-25 14:42:59	retweeted	Sivapriyan E.T.B	sivaetb	1

Figure 3: Search by screen_name 'sivaetb'

Top 10 users based on their follower count; time taken 0.96 seconds (cached data):



user_id	name	screen_name	url	location	followers_count	friends_count	listed_count	favourites_c
69183155	detikcom	detikcom	http://www.detik.com	Jakarta, Indonesia	15927642	28	13298	313
62513246	J.K. Rowling	jk_rowling	http://www.jkrowling.com	Scotland	14608046	721	37917	27353
42606652	AajTak	aajtak	http://www.aajtak.in	India	9706667	416	3517	16782
39240673	ABP News	ABPNews	http://abplive.com	India	9562582	248	3941	99
240649814	TIMES NOW	TimesNow	http://www.timesnownews.com	India	9499328	391	5109	4
56304605	Rajdeep Sardesai	sardesairajdeep	http://rajdeepsardesai.net/	New Delhi	8947342	568	8993	7598
24744541	Le Monde	lemondefr	https://www.lemonde.fr	Paris	8808784	628	36505	1609
55507370	tvOneNews	tvOneNews	http://tvonenews.com	Indonesia	8787649	50	8737	4347
23343960	Kompas.com	kompascom	http://www.kompas.com	Indonesia	7678373	23	10617	114
15016209	NTV	ntv	https://www.ntv.com.tr/	Turkey	7429223	29	5436	3

Figure 4: Top 10 users by follower count

Top 10 tweets based on their retweet count; time taken 1.78seconds (cached data):

Select Metric Top 10 Tweets by Retweet Count

Total Time taken for search to run= {1.7775928974151611} seconds.

Show 10 entries

Search:

etweet_count	_id	id_str	possibly_sensitive	text
86	662aea67676c16a8c2b2087c	1254030161403674624	False	Milwaukee’s health commissioner has now tied 40 coronavirus infections to the /
32	662ae9fe676c16a8c2b1f48a	1254028244166356998	False	Gözün çıksın corona😷 Ülkece asabii olduk 🤷‍♀️muhtemel psikoloji ektedir 😂😂😂👇
00	662af1bc98079b56c6c980ba	1254045905252167681	False	A MUST READ...Coronavirus Restrictions: Government Bears the Burden of Proof
22	662ae9c7676c16a8c2b1e9dd	1254027175122153479	None	In Illinois, liberal politicians cut sweetheart deals with corrupt union bosses to ke
22	662aeedd98079b56c6c8f315	1254032746361417729	False	Thalapathy fans from Sivakasi Helped the Poor Family who are affected by this c
22	662af03398079b56c6c9343d	1254038838403493889	True	If that’s how they are following lockdown/social distancing then Mumbai is sitting
85	662aea55676c16a8c2b20530	1254029837460799490	False	Süleyman Özışık durumu çok iyi açıklamış. \nNedir sizin bu bitmek bilmeyen düşr
03	662aefd598079b56c6c92231	1254037192952995840	False	A maioria dos cientistas acreditam que o coronavírus não veio da China e sim do
78	662ae927676c16a8c2b1cb83	1254024050889949186	None	Glückwunsch: In Sachsen-Anhalt wurden die Daten aller Corona-Infizierten oder
77	662aef5198079b56c6c908ce	1254034810231717889	False	Please, if you read one article today, read this. \nhttps://t.co/pZ7OSbu0V1

Showing 1 to 10 of 10 entries

Previous 1 Next

Figure 5: Top 10 tweets by retweet count

CONCLUSION

The search application embodies a meticulous approach to web application development, prioritizing performance, user experience, and maintainability. Future efforts should concentrate on further optimizing these aspects and expanding the application's capabilities based on user feedback and technical requirements.

Conclusions on Design

The application adopts a user-centric design, segmenting functionalities logically and offering diverse endpoints for search, toplevelmetrics, and explore. Performance optimization is evident through the implementation of a caching mechanism, reducing database load and enhancing responsiveness. Data handling focuses on accessibility and clarity, with formatted_df facilitating user-friendly data presentation. Asynchronous processing enhances scalability by facilitating non-blocking operations.

Insights from Experiments

Experiment results validate the efficacy of the caching layer in improving performance, highlighting its role in mitigating database load. The integration of MySQL and MongoDB showcases the database architecture's versatility in accommodating diverse data models efficiently. Integration between React front-end components and Flask endpoints is essential for a cohesive user experience. The iterative development process, demonstrated by Flask's debug mode, ensures continuous improvement aligned with evolving user needs.

Recommendations for Future Development

Seamless integration between Flask and React layers is essential for a cohesive user experience, potentially facilitated by RESTful API or GraphQL protocols. Comprehensive testing, including load testing, is crucial to validate scalability and robustness, particularly concerning the caching system. Security considerations should prioritize data safeguarding and robust sanitization measures. Continued utilization of monitoring tools is recommended for proactive issue detection and resolution. UI refinement based on user feedback can enhance usability and satisfaction.

Key Insights from the Project

The project underscores the importance of selecting appropriate databases and caching mechanisms for optimal performance. Modular design principles enhance maintainability and scalability, while asynchronous operations improve application responsiveness. Separation of front-end and back-end components facilitates independent development and scaling. Secure user input handling ensures data integrity and application security. Effective data presentation enhances usability and user satisfaction.

REFERENCES

1. *Cloud SQL for MySQL Managed Database | Google Cloud*. (n.d.). Google Cloud. https://cloud.google.com/sql/mysql?hl=en
2. *Get Started with Atlas - MongoDB Atlas*. (n.d.). https://www.mongodb.com/docs/atlas/getting-started/
3. Korth, H. F., Sudarshan, S., & Professor, A. S. (2019). *Database System Concepts*. McGraw-Hill Education.
4. *Use cases, tutorials, & documentation*. (n.d.). Twitter Developer Platform. https://developer.twitter.com/en

WORKLOAD DISTRIBUTION

Hsiao-Chun Hung (hh617): Conducted Exploratory Data Analysis, created Non-Relational Database for Tweets using MongoDB, processed streaming data, created presentation slides.

Yuyue Sun (ys898): Got Cloud credentials, created Relational Database for User Information using MySQL, table schema design, processed streaming data for MySQL, search query debug.

Yu Wang (yw1029): Set up the Django environment, connected to databases; built search application, query script developing.

Zhuoer Liu (zl413): Cache and UI; managed API interactions and state management in React.