SJSU CHATGPT TWITTER ANALYSIS

Lab Report - 1 (Team - 8)

Shashank Reddy Kandimalla (016799523), Swati (016702413), Sakshi Manish Mukkirwar (016794765), Yamini Muthyala (016766165), Rohith Reddy Vangala (016762109)

ABSTRACT - This lab report describes the application for examining tweets related to ChatGPT, a language model built using the GPT-3.5 architecture. The tweets with the hashtag #chatgpt were collected from Twitter and used as the dataset[1] for this project. Here we have identified several interesting things about the dataset[1] and have generated the required outputs that are listed in Solution requirements (3). The Entity Relationship model (fig-2) was used to establish a conceptual design for the database (fig-3), and functional analysis was carried out to determine the essential functional elements. The SJSU ChatGPT Tweets Analysis database was moved to AWS because the application system is cloud-native. The application helps users understand the attitudes, viewpoints, and actions of people discussing ChatGPT on Twitter by offering insights into the online discourse surrounding this language model.

KEYWORDS - Amazon Web Services, Attributes, Cloud-Native, Data Processing, Entities, Normalization, Triggers, Stored Procedures.

1. INTRODUCTION

The creation of various language models that have transformed the field of natural language processing has been facilitated by the growth of artificial intelligence (AI) (NLP). Based on the GPT-3.5 architecture, ChatGPT is one such language model that has attracted a lot of attention recently. It is necessary to study this online discussion as people discuss ChatGPT and contribute their personal stories, viewpoints, and ideas in order to learn more about the attitudes and trends that surround this language model.

We propose an application that examines tweets concerning ChatGPT that have been scraped from Twitter and contain the hashtag #chatgpt in order to fill this demand. To help users understand the behaviors and patterns of those discussing ChatGPT, the application intends to provide insights into the online conversation surrounding this language model on twitter.

The development of this application is thoroughly described in this report. We start by identifying the functional requirements for the application, then use the Entity Relationship model to conceptually build the database. Then, we conduct a functional analysis to pinpoint the essential functional elements needed for the application. Finally, we talk about how the SJSU ChatGPT Tweets Analysis database was moved to Amazon using a cloud-native strategy. Ultimately, the purpose of this study is to give readers a thorough understanding of the design and operation of the ChatGPT Tweets Analysis program.

2. PROBLEM STATEMENT

With the rise of language models, like ChatGPT, it is crucial to understand the attitudes, viewpoints, and actions of people discussing these models online. Twitter is a popular platform for discussing ChatGPT, and analyzing tweets related to this language model can provide valuable insights. Therefore, the problem statement is to develop an application that examines tweets containing the hashtag #chatgpt to understand the online discourse surrounding ChatGPT. The application should provide insights into the behaviors and patterns of those discussing ChatGPT on Twitter, by identifying the most active individuals, the most popular tweet, frequently used hashtags, and the tweet that received the most attention. Additionally, the application should provide a cloud-native solution for hosting the SJSU ChatGPT Tweets Analysis database.

To store the tweets and their associated metadata, as well as the results of our analysis, we will use a sophisticated database system that can handle large volumes of data and perform complex queries efficiently. The database will be designed to allow for easy updating and modification of the results of our analysis, as new data becomes available or as our research questions evolve.

SJSU CHATGPT TWITTER ANALYSIS

3. SOLUTION REQUIREMENTS

1. The proposed application system (fig-1) will extract Twitter data from the ChatGPT Twitter dataset and perform a comprehensive analysis of the dataset. The analyzed findings will be stored in a database, providing insights into various aspects of ChatGPT tweets.
2. The system's capabilities include finding the most active Twitter users talking about ChatGPT and the viral tweet based on retweets and figuring out the most talked-about tweet with the most replies and conversations , as well as the most recent tweets and likes.
3. The application is designed to locate frequently used hashtags in ChatGPT tweets and recognize trends or conversational themes based on hashtag usage.
4. The system can identify the tweet count based on the different devices and can determine the connection between hyperlinks in tweets and the websites or pages that are most frequently shared on Twitter. Moreover ,it can identify the relationship between hyperlinks in tweets and engagement indicators such as retweets and likes.
5. Additionally, the system will recognize the most active hour of the day for tweeting. Regarding the individuals , the people with influence or expertise who are frequently referenced in tweets can also be recognized by the system.
6. The system will also determine the commonly used media by popular accounts or topics in their tweets.

3.1.LIMITATIONS

However, The system's limitations should also be taken into account. It does not have sentiment analysis or topic modeling for more extensive insights. The system may not be able to capture Twitter data outside the ChatGPT dataset, limiting its scope.

Also, The system may not always accurately determine the commonly used media by popular accounts or topics in their tweets, as this can be influenced by various factors such as user preferences, platform algorithms, and trending topics. The ChatGPT Twitter dataset may not be a representative sample of the overall population, and it may not reflect the opinions and experiences of all Twitter users. Thus, the results of the analysis may not be generalized to the entire population.The system can currently only analyze the ChatGPT dataset provided in .csv file format, which may limit its ability to analyze other types of data, such as XML files. Additionally, the accuracy of the system's analysis may be affected by factors like Twitter API limitations or data quality issues. Thus, users of the system should be aware of these limitations and use the system's output accordingly. Overall, the system will be an essential tool for researchers and analysts seeking insights into ChatGPT tweets.

4. DATASET AND PRE-PROCESSING

There are 12,000 tweets altogether in the "ChatGPT Twitter Dataset"[1] dataset that is accessible on Kaggle. Tweets related to the ChatGPT language model are included in the dataset. Date time, tweet id, tweet text, Username, Permalink, User, Outlink, Countlinks, Reply Count, Retweet Count, Like Count, Quote Count, Conversation Count, Language, Source, Media, Quoted Tweet, Mentioned Users, hashtag, Hashtag Count are the attributes included in the collection. These characteristics offer insightful data on the users tweeting about ChatGPT and can be applied to various studies.

The multivalued attributes have been divided into single valued attributes in order to normalize the dataset. The normalized columns include countlinks, outlinks, mentioned users, and hashtags (fig-2).

5. FLOW CHART

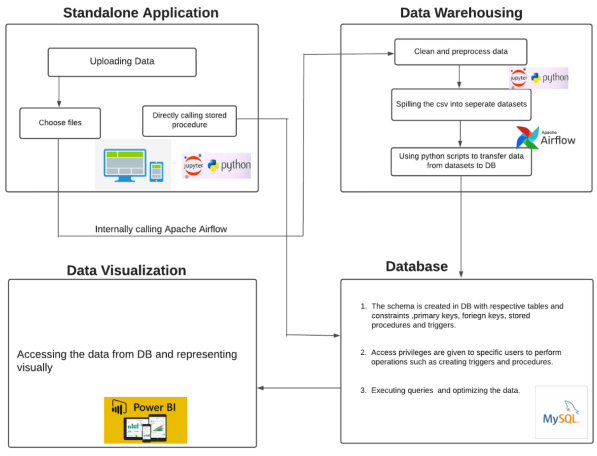


Fig - 1 Flowchart of the proposed application

6. CONCEPTUAL DATABASE DESIGN

1. Identifying the entities and relationships: Analyzed the dataset and identified the main entities that need to be stored in the database. For the Twitter dataset, entities could

include users, tweets,hashtags, mentions, and retweets. After identifying the entities, we have determined the relationships between them. For example, a tweet is created by a user, a tweet can contain hashtags, and a tweet can be retweeted by other users.

2. Normalizing the schema: Normalized the schema to reduce data redundancy and improve data consistency (fig-2). This involved breaking down large tables into smaller tables, eliminating repeating groups, and ensuring that each table has a primary key.

3. Determining data types: Determined the data types for each field in the tables (fig-3). For example, the"created_at" field in the tweets table is declared with datetime data type.

4. Creating the tables, constraints and indexes: Using the normalized schema and data types, created the tables in the database.Defined constraints on the tables, such as primary keys, foreign keys, and unique constraints and also Determined which columns will be frequently searched or sorted, and created indexes on those columns to improve query performance (fig-3).

5. Populating the tables: Imported the data from the Twitter dataset into the database tables (fig-26,26.1).

6. Testing the database: Tested the database to ensure that it is functioning correctly and that queries return the expected results.

7. Optimizing the database: Optimized the database by analyzing query performance, adjusting indexes, and optimizing SQL queries as needed.

8. Monitor and maintain the database: Regularly monitored the database to identify and fix any issues, and perform routine maintenance tasks such as backing up the database on aws rds.

9. Defining user roles and access levels: Identified the different types of users who will be accessing the system and the data. For our project we had set up access privileges for our group members who could access the database. We have granted specific privileges, such as select, insert, update, delete, etc. based on the role and responsibilities of group members.

10. Implement authentication: Require users to authenticate themselves before accessing the system. This so we have included requiring a username and password to login to the database(fig- 25).

6.1. NORMALIZATION STEPS

1. Removing repeating groups: The original dataset contained repeating groups of data, such as Outlink and Mentioned Users, which were removed and placed into separate tables.

2. Separating data into tables: The remaining data was separated into separate tables based on their relatedness, such as users, tweets, count_links, media, outlinks, trending_tweets, and tweet_counts.

3. Assigning primary keys: Each table was assigned a primary key, which uniquely identifies each record in the table. For example, the users table was assigned the primary key 'user_name', while the tweets table was assigned the primary key ‘tweet_id'.

4. Assigning foreign keys: Tables that were related to each other were linked using foreign keys. For example, the tweets table had foreign keys to link to the users table and to itself (to represent a conversation).

5. Normalizing columns: Columns in each table were normalized to ensure that each column represented only one type of data. For example, the count_links table had a column for 'count_links', which represented a count of links in a tweet, and the media table has columns for 'media_previewurl' and 'media_fullurl', which represented different aspects of the same data.

6. Resolving many-to-many relationships: Tables with many-to-many relationships were resolved using an intermediate table. For example, the tweet_counts table had a many-to-many relationship with the hashtag table, so an intermediate table was created to link the two tables.

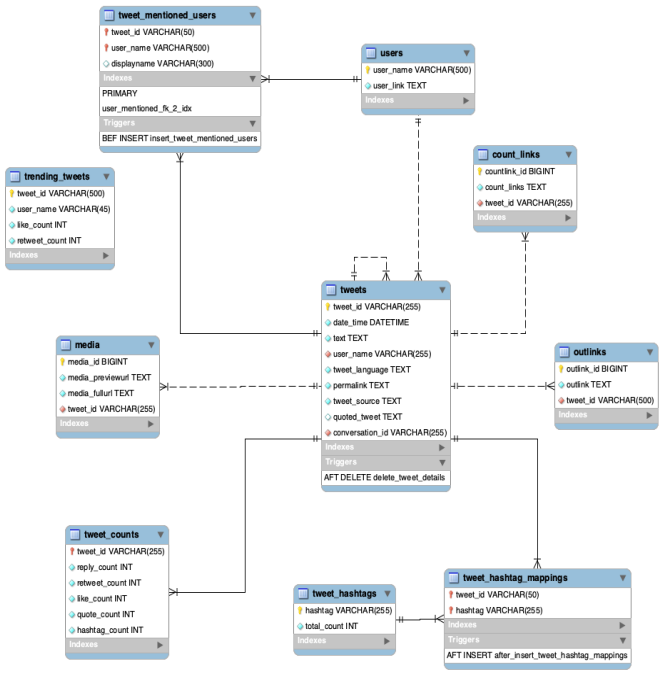


Fig - 2 ER Diagram

SJSU CHATGPT TWITTER ANALYSIS

Entity name	Attribute(Keys)	Description
users	user_name(pk),user_link	stores information about Twitter users, including their username and link to their profile
tweets	tweet_id(pk),user_name(fk),conversation_id(fk),date_time,text,tweet_language,permalink,tweet_source,quoteed_tweet	This table stores information about individual tweets, including the tweet ID, date and time of the tweet, the text of the tweet, the username of the user who tweeted it, the language of the tweet, the permalink to the tweet, the source of the tweet, the ID of any quoted tweet, and the conversation ID (if applicable). This table also has foreign key constraints that reference the users table for the user_name column and reference itself for the conversation_id column, which creates a hierarchical structure for storing tweets in conversations. Additionally, there is a trigger defined on this table that automatically generates a new tweet_id if a conversation_id is not provided
tweet_mentioned_users	tweet_id(pk),user_name(pk),displayname	This table stores information about Twitter users who are mentioned in tweets, including the ID of the tweet, the username of the mentioned user, and the display name of the mentioned user (if available). This table has foreign key constraints that reference the tweets table for the tweet_id column and the users table for the user_name column. The primary key of this table is a combination of the tweet_id and user_name columns to ensure uniqueness.
tweet_hashtags	hashtag(pk,total_count	This table stores information about hashtags used in tweets, including the hashtag itself and the total count of times it has been used in tweets. The hashtag column is set as the primary key for this table
outlinks	outlink_id(pk),tweet_id(fk,outlink	This table stores information about links included in tweets, including the link itself and the ID of the tweet that includes the link. This table has a foreign key constraint that references the tweets table for the tweet_id column.
count_links	countlinks_id(pk),tweet_id(fk),count_links	This table stores the count of links for each tweet, along with the ID of the tweet. This table also has a foreign key constraint that references the tweets table for the tweet_id column
media	media_id(pk),tweet_id(fk),media_previewurl,media_fullurl	This table stores information about media (images, videos, etc.) included in tweets, including the preview URL and the full URL of the media, along with the ID of the tweet that includes the media. This table has a foreign key constraint that references the tweets table for the tweet_id column
tweet_counts	tweet_id(pk),reply_count,retweet_count,lke_count,quote_count,hashtag_count	This table stores counts for different tweet actions (replies, retweets, likes, and quotes), along with the number of hashtags used in the tweet. This table has a foreign key constraint that references the tweets table for the tweet_id column
tweet_hashtag_mappings	tweet_id(pk),hashtag(pk)	This table is a mapping table that associates tweets with the hashtags used in them. It includes the tweet ID and hashtag, both of which are foreign keys that reference the tweets and tweet_hashtags tables, respectively. The primary key of this table is a combination of the tweet_id and hashtag columns to ensure uniqueness
trending_tweets	tweet_id(pk),user_name,like_count,retweet_count	A "trending tweet" refers to a tweet that is currently popular or receiving a high volume of engagement on Twitter. Twitter measures the popularity of a tweet by the number of retweets>1000, likes>1000 receives over a period of time

Fig - 3 Table that shows Entities, Attributes.

7. FUNCTIONAL ANALYSIS

Using the provided dataset, the suggested application system is a tool that will examine the Twitter discussion surrounding ChatGPT. The main objectives of the tool are to find users who are most engaged in ChatGPT discussions, the most popular tweet, the most popular hashtags used in ChatGPT tweets, and the most debated tweet. The application will also make it possible to analyze the Twitter conversations related to ChatGPT.

1. User analysis: This section will reveal which people are engaged in the most ChatGPT discussion. The number of times each user has tweeted about ChatGPT will be counted using the Tweet text and User entities. The top users who tweeted about ChatGPT will then be listed by the system.

2. Viral Tweet Analysis: This part will determine the ChatGPT tweet that has received the most retweets. It will output the tweet that has been retweeted the most using the Retweet Count entity to order tweets depending on the quantity of retweets.

3. Hashtag Analysis : This will reveal the popular hashtags that were used in tweets about ChatGPT. Using the Hashtag object, we can get a list of the top hashtags.

4. Conversation Analysis: Based on comments and conversation, this part will determine the ChatGPT tweet that has received the most attention. The tweet with the highest level of engagement will be generated after sorting tweets using the Reply Count and Conversation Count entities to account for the quantity of responses and conversations.

The dataset also includes several metadata fields for each tweet, including the tweet ID, the date and time of the tweet, the number of retweets and likes the tweet received. This metadata has been used to perform additional analysis or to filter the tweets based on specific criteria.

8. QUERIES

1. Users most discussing about it on twitter(tweets):
Query:

```
select user_name, count(*) as total_tweets from tweets
group by user_name
order by total_tweets desc limit 10;
```

Interaction with database: The group by clause groups the tweets by user_name, and the count function counts the number of tweets posted by each user. The order by clause sorts the results in descending order based on the total number of tweets, and the limit clause limits the results to the top 10 users.

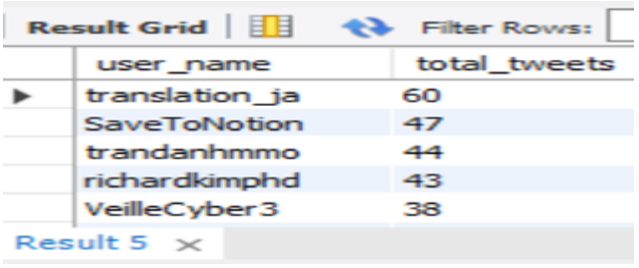


Fig-4 Output for Users most discussing about it on twitter(tweets)

2. Most viral tweet (wrt retweets) about the chatgpt(tweet_counts,tweets):
Query:

```
select t.tweet_id, t.text, tc.retweet_count from tweets as t
inner join tweet_counts as tc on t.tweet_id = tc.tweet_id
order by tc.retweet_count desc limit 1;
```

Interaction with database: The query uses an inner join to combine the 'tweets' and 'tweet_counts' tables based on their tweet id. The order by clause sorts the results in descending order based on the number of retweets, and the limit clause limits the results to the tweet and it gives the highest number of retweets.

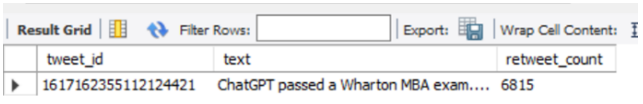


Fig-5 Output for Most viral tweet about the chatgpt

SJSU CHATGPT TWITTER ANALYSIS

3. Hashtags used prominently for the tweets about chatgpt(tweet_hashtags, tweet_hashtag_mappings):
Query:

```
select th.hashtag, th.total_count from tweet_hashtags as th
inner join tweet_hashtag_mappings as thm on
th.hashtag = thm.hashtag where lcase(th.hashtag) not like
'%chatgpt' group by th.hashtag
order by th.total_count desc limit 10;
```

Interaction with database: The query uses an inner join to combine the 'tweet_hashtags' and 'tweet_hashtag_mappings' tables based on the hashtag name. The group by clause groups the hashtags by their name, and the order by clause sorts the results in descending order based on the count. The limit clause gives the results of the top 10 hashtags.

hashtag	total_cou...
AI	1958
OpenAI	772
ArtificialIntelligence	539
Microsoft	451
IA	238
Google	224
chatgpt3	201
MachineLearning	198
cybersecurity	127

Fig-6 Output for hashtags used prominently for the tweets about chatgpt.

4. Most discussed tweet (wrt replies and conversation) (tweet_counts, tweets):
Query:

```
select t.tweet_id, t.text, tc.reply_count +
count(distinct tm.user_name) as discussion_count from
tweets as t inner join tweet_counts as tc on t.tweet_id =
tc.tweet_id left join tweet_mentioned_users as tm on
t.tweet_id = tm.tweet_id group by t.tweet_id
order by discussion_count desc limit 5;
```

Interaction with database: The query uses an inner join to combine the 'tweets' and 'tweet_counts' tables based on their tweet id. it also uses a left join to combine the 'tweets' and 'tweet_mentioned_users' tables based on their tweet id. The group by clause groups the tweets by their id, and the order by clause sorts the results in descending order based on the discussion count.

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetcl

tweet_id	text	discussion_count
1617161446202245120	ChatGPT is Free. But most people don't know t...	3099

Fig-7 Output for Most discussed tweet.

5. Latest tweet(tweet) :
Query:

```
select t.tweet_id, t.text as most_recent_tweet , t.user_name,
t.date_time as latest_time from tweets t inner join ( select
tweet_id, max(date_time) as max_datetime from tweets
group by tweet_id) t2 on t.tweet_id = t2.tweet_id and
t.date_time = t2.max_datetime
order by latest_time desc limit 10;
```

Interaction with database: The query uses inner join to join the 'tweets' table with a subquery that retrieves the maximum date_time for each tweet_id. It then orders the results by the latest_time in descending order and returns the top 10 latest tweets.

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

tweet_id	most_recent_tweet	user_name	latest_time
1617778733355790342	Ahora sueño con el día en que Amazon integre ...	AmericoSD_69	2023-01-24 06:58:01+00:00
1617778731678044162	Portland Shop Uses ChatGPT To Tell Family Stori...	EuniceNyandat	2023-01-24 06:58:01+00:00
1617778728481992705	5 minuti di #chatGPT e ho capito che apprende i...	marcopiccinini	2023-01-24 06:58:00+00:00
1617778726393249792	@r8r Ich hab mal die AI dazu befragt (ChatGPT...	werpu	2023-01-24 06:57:59+00:00
1617778712082096128	#ChatGPT ist ein #Chatbot, der durch künstlich...	HorstKrieger	2023-01-24 06:57:56+00:00

Fig-8 Output for Latest tweet.

6. Number of likes(tweet,tweet_count):
Query:

```
select tweets.tweet_id, concat(text, '\n') as tweet ,
tweet_counts.like_count as 'number of likes' from tweets
join tweet_counts on tweets.tweet_id =
tweet_counts.tweet_idlimit 10;
```

Interaction with database: The query uses join to join the 'tweets' table with 'tweet_counts' table on tweet_id. It then limits the results to 10 .

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

tweet_id	tweet	Number of Likes
1617156270871699456	ChatGPTで遊ぶの忘れてた！！ 書類作るコード...	5
1617156291046133761	@AlexandrovnaIng Prohibition of ChatGPT has ...	5
1617156308926349312	Schaut Euch an, was @fobizz @DianaKnodel all...	4
1617156332297256961	Bow down to chatGPT ?..... https://t.co/ENTSzi...	2

Fig-9 Output for number of likes.

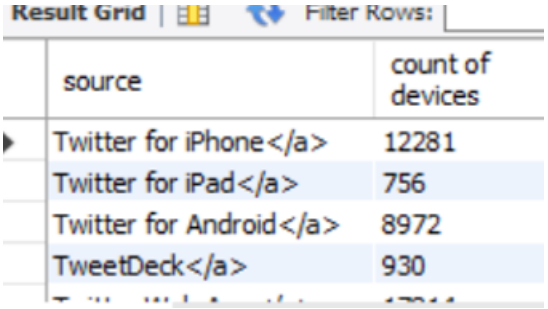
7. Count of tweets from different devices(tweets):
Query:

```
select replace(substring_index(substring_index

(tweet_source, '>',-2), '<', -4), '</a>', '') as source,
count(tweet_source) as 'count of devices' from tweets
group by tweet_source
order by 'count of devices' desc;
```

Interaction with database: The query uses the replace, sub_string_index and group by functions to extract the source of the tweet from the tweet_source column. It then

groups the results by the source and orders them by the count of devices in descending order.



source	count of devices
Twitter for iPhone	12281
Twitter for iPad	756
Twitter for Android	8972
TweetDeck	930

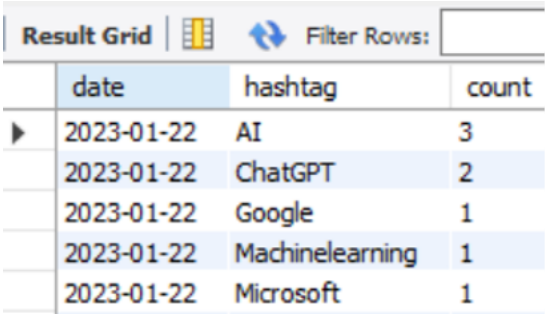
Fig-10 Output for count of tweets from different devices.

8. Identifying trends or topics of conversation based on hashtag usage(tweets ,tweet_hashtag_mappings,):

Query:

```
select date(date_time) as date, hashtag, count(*) as count
from tweet_hashtag_mappings join tweets on
tweet_hashtag_mappings.tweet_id = tweets.tweet_id where
date_time between '2023-01-01' and '2023-03-31'
group by date, hashtag
order by date, count desc;
```

Interaction with database: The query uses join to join the 'tweet_hashtag_mappings' table with 'tweets' table on the basis of tweet_id. It then filters the results to select only those tweets that were posted between 1st January 2023 and 31st March 2023. Finally, it groups the results by date and hashtag and orders them by date and the count of times.



date	hashtag	count
2023-01-22	AI	3
2023-01-22	ChatGPT	2
2023-01-22	Google	1
2023-01-22	Machinelearning	1
2023-01-22	Microsoft	1

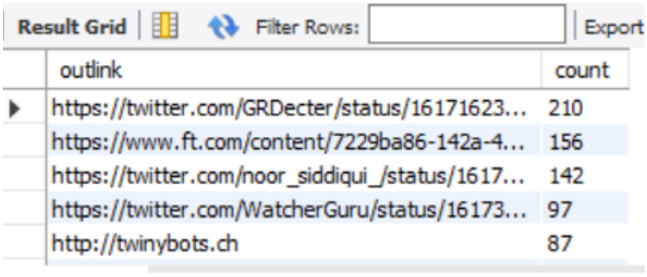
Fig-11 Output for identifying trends or topics of conversation based on hashtag usage.

9. Analyzing which websites or pages are most commonly shared on twitter(outlines):

Query:

```
select distinct outlink, count(*) as count from outlinks
where outlink is not null
group by outlink
order by count desc limit 10;
```

Interaction with database: The query performs a group by on the 'outlink' column and then sorts the results by count in descending order.



outlink	count
https://twitter.com/GRDecter/status/16171623...	210
https://www.ft.com/content/7229ba86-142a-4...	156
https://twitter.com/noor_siddiqui_/status/1617...	142
https://twitter.com/WatcherGuru/status/16173...	97
http://twinybots.ch	87

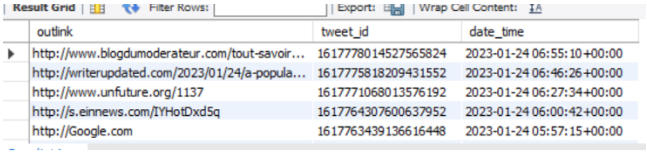
Fig-12 Output for Analyzing which websites or pages are most commonly shared on twitter.

10. Websites which are not secure in outlinks (outline, tweets):

Query:

```
select distinct outlink, t.tweet_id, t.date_time from outlinks
o join tweets t on o.tweet_id = t.tweet_id where outlink like
'http:%' and outlink not like 'https:%' and (outlink like
'%.com%' or outlink like '%.org%'  or outlink like '%.net%')
order by t.date_time desc;
```

Interaction with database: The query performs a join operation between the 'outlinks' and 'tweets' table on the tweet_id column. It then uses where condition to check if the outlink starts with 'http:' and does not start with 'https:'. It then checks if the outlink contains a .com, .org, or .net domain. Finally, it sorts the results by date-time in descending order.



outlink	tweet_id	date_time
http://www.blogdumoderateur.com/tout-savoir...	1617778014527565824	2023-01-24 06:55:10+00:00
http://writerupdated.com/2023/01/24/a-popula...	1617775818209431552	2023-01-24 06:46:26+00:00
http://www.unfuture.org/1137	1617771068013576192	2023-01-24 06:27:34+00:00
http://s.einnews.com/IYHotDxd5q	1617764307600637952	2023-01-24 06:00:42+00:00
http://Google.com	1617763439136616448	2023-01-24 05:57:15+00:00

Fig-13 Output for Websites which are not secure in outlinks.

11. Analyzing how the use of links in tweets correlates with engagement metrics like retweets and likes(tweets):

This query calculates the presence of a link in the tweet text by counting the number of occurrences of the string "http" in the text column. If this count is greater than zero, the tweet is considered to have a link (has_link = 1), otherwise it does not have a link (has_link = 0). We then calculate the average retweet and like counts for each group, grouping by the has_link column. The results of this query can help us understand whether tweets with links tend to have higher or

lower engagement than tweets without links. If we find that tweets with links have higher engagement, we may want to encourage the use of links in our Twitter strategy. On the other hand, if we find that tweets without links have higher engagement, we may want to focus on creating more engaging content that doesn't rely on external links.

Query:

```
select if(length(text) - length(replace(text, 'http', '')) > 0, 1,
0) as has_link, avg(retweet_count) as avg_retweets,
avg(like_count) as avg_likes from tweets join tweet_counts
on tweets.tweet_id = tweet_counts.tweet_id
group by has_link;
```

Interaction with database: The query performs a join operation between the 'tweets' and 'tweet_counts' table on the tweet_id column. It then checks if the tweet contains a link by counting the number of times of 'http' in the tweet text. It groups the results based on whether the tweet contains a link or not. Finally, it gives the average retweet count and average like count for each group.

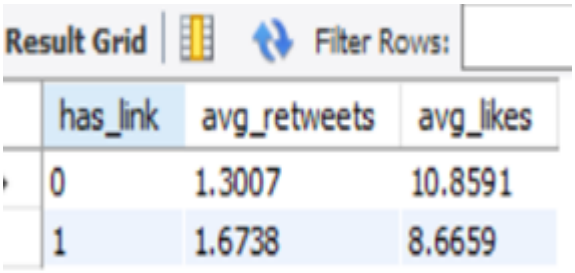


Fig-14 Output for Analyzing how the use of links in tweets correlates with engagement metrics like retweets and likes.

12. Analyzing the types of media that are most commonly included in tweets(media):

Query:

```
select media_previewurl, count(*) as count from media
group by media_previewurl
order by count desc limit 10;
```

Interaction with database: The query uses the group by clause to group the tweets by media_previewurl and the count function to count the number of tweets that contain that media file. It also uses the order by clause to sort the results in descending order based on the count of tweets and the limit clause to limit the results to the top 10 media files.

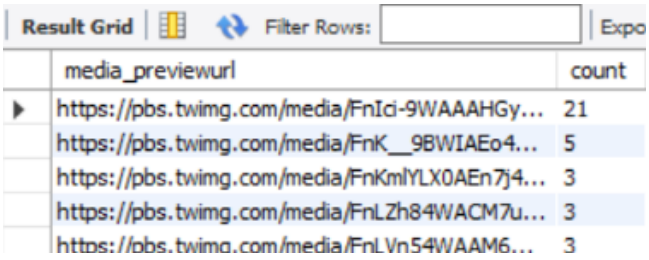


Fig-15 Output for Analyzing the types of media that are most commonly included in tweets

13. Identifying popular accounts or topics that frequently use media in their tweets(tweets, media):

Query:

```
select tweets.user_name, count(*) as count from tweets join
media on tweets.tweet_id = media.tweet_id
group by tweets.user_name
order by count desc limit 10;
```

Interaction with database: The query performs a join operation between the 'tweets' and 'media' table on the tweet_id column. It then groups the result by the user_name column and counts the number of media used in each tweet. Finally, it gives the top 10 usernames with the highest count of media used in their tweets.

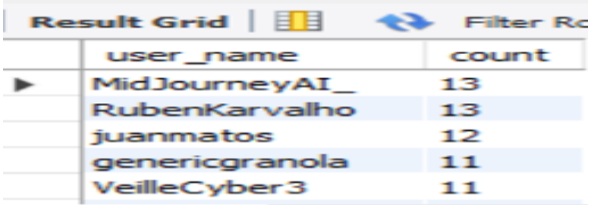


Fig-16 Output for Identifying popular accounts or topics that frequently use media in their tweets.

14. Examining the impact of visual content (media) on engagement metrics like retweets and likes(tweets, media):

Query:

```
select ifnull(media.media_id, 0) as has_media,
avg(tweet_counts.retweet_count) as avg_retweets,
avg(tweet_counts.like_count) as avg_likes from tweets
left join media on tweets.tweet_id = media.tweet_id
left join tweet_counts on tweets.tweet_id =
tweet_counts.tweet_id
group by has_media
order by avg_retweets,avg_likes desc;
```

Interaction with database: : The query uses left to join the 'tweets' table with the 'media' and 'tweet_counts' tables. The AVG function is used to calculate the average number of

retweets and likes. It uses the ifnull function to assign a value of 0 to tweets without media files. The group by function groups the tweets based on whether they have media or not. The order by function sorts the results in ascending order based on the average number of retweets and in descending order based on the average number of likes.

has_media	avg_retweets	avg_likes
6295	0.0000	2.0000
13490	0.0000	2.0000
25822	0.0000	2.0000
13830	0.0000	2.0000
30007	0.0000	2.0000

Fig-17 Output for Examining the impact of visual content (media) on engagement metrics like retweets and likes.

15. Identifying influencers or thought leaders who are frequently mentioned in tweets(tweet_mentioned_users, users):
Query:

```
select u.user_name, count(*) as count from
tweet_mentioned_users u join users i on
u.user_name=i.user_name
group by u.user_name
order by count desc limit 10;
```

Interaction with database: : The query uses join operation to join the 'tweet_mentioned_users' table with the 'users' table based on the user_name. The group by function groups the tweets by the user_name, and the count function counts the number of tweets in which each user is mentioned. The order by clause sorts the results in descending order based on the count of tweets and the limit clause limits the output to the top 10 users.

Result Grid | Filter

user_name	count
GRDecter	359
chatgpt_issac	191
sejournal	102
noor_siddiqui_	99
ShiLLin_ViLLian	92

Fig-18 Output for Identifying influencers or thought leaders who are frequently mentioned in tweets.

16. The most active hour of the day for tweeting:
Query:

```
select hour(date_time) as hour, count(*) as total_tweets
from tweets
group by hour(date_time)
order by total_tweets desc limit 1;
```

Interaction with database: : The query uses the hour function to extract the hour from the date_time column. The group by clause groups the tweets by the hour, and the count function counts the number of tweets during each hour. The order by clause sorts the results in descending order based on the total number of tweets, and the limit clause limits the results to the hour with the highest number of tweets.

Result Grid | Filte

hour	total_tweets
15	3297

Fig-19 Output for The most active hour of the day for tweeting.

17. Most used tweet language on the platform:
Query:

```
select tweet_language, count(*) as total_tweetsfrom tweets
group by tweet_language
order by total_tweets desc limit 1;
```

Interaction with database: : The query groups the tweets by tweet_language and counts the number of tweets in each language. Finally, it gives the language with the highest count.

Result Grid | Filter Rows:

tweet_language	total_tweets
en	32076

Fig-20 Output for Most used tweet language on the platform.

8.1 TRIGGERS

1.To update the hashtagcount(attribute) and entity whenever there is new hashtag or to change the count for new hashtag:
Trigger:

```
create definer=`root`@`%` trigger
`after_insert_tweet_hashtag_mappings` after insert on
`tweet_hashtag_mappings`
for each row begin
declare hashtag_exists integer;
set hashtag_exists = (select count(*) from tweet_hashtags
where hashtag = new.hashtag);
if hashtag_exists = 0 then insert into
tweet_hashtags(hashtag) values (new.hashtag);end if;
```

```
update tweet_hashtags set total_count = total_count + 1
where hashtag =new.hashtag; update tweet_counts set
hashtag_count = hashtag_count + 1 wheretweet_id =
new.tweet_id;
end
```

2.To update tweet and user entities if a new record is not present in one of the table:

Trigger:

```
create definer=`root`@`%` trigger
`insert_tweet_mentioned_users` before insert on
`tweet_mentioned_users` for each row begin
-- check if user exists in users table, if not, insert new
record.
if not exists (select * from users where user_name =
new.user_name) then insert into users (user_name,
user_link) values (new.user_name, ''); end if;
-- check if tweet exists in tweets table, if not, insert new
record

if not exists (select * from tweets where tweet_id =
new.tweet_id) then insert into tweets (tweet_id, date_time,
text, user_name, tweet_language, permalink, tweet_source,
quoted_tweet_id, conversation_id) values (new.tweet_id,
now(), '', new.user_name, '', '', '', '');end if; end
```

3.Trigger that will delete all details associated with a tweet whenever that tweet is deleted from the tweets table:

Trigger:

```
delimiter //
create trigger delete_tweet_details
after delete on tweets
for each row begin -- delete from outlinks table
delete from outlinks where tweet_id = old.tweet_id;
-- delete from count_links,media, table ,
tweet_mentioned_users,tweet_hashtag_mappings
delete from count_links  where tweet_id = old.tweet_id;
delete from media where tweet_id = old.tweet_id;
delete from tweet_mentioned_users  where tweet_id =
old.tweet_id;
delete from tweet_hashtag_mappings where tweet_id =
old.tweet_id;
-- delete from tweet_counts table
delete from tweet_counts  where tweet_id = old.tweet_id;
end // delimiter ;
```

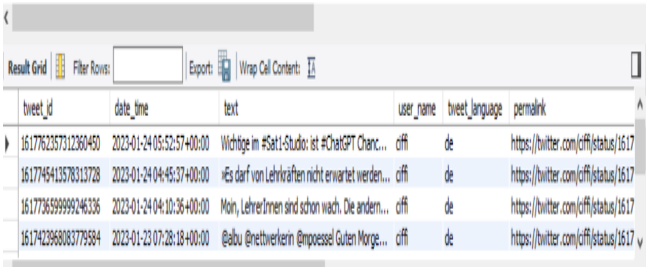
8.2 STORED PROCEDURES

1.Stored procedure for number of tweets based on particular user:

Procedure:

```
create definer=`root`@`%` procedure
`get_tweets_and_mentioned_users`(in p_username
varchar(255))
begin select t.tweet_id, t.date_time,  t.text, t.user_name,
t.tweet_language,t.permalink,t.tweet_source,t.quoted_tweet
,mu.user_name as mentioned_user_name,mu.displayname
as mentioned_displayname from tweets t
left join tweet_mentioned_users mu on t.tweet_id =
mu.tweet_id where t.user_name = p_username order by
t.date_time desc;end
```

Calling the stored procedure: call twitter.get_tweets_and_mentioned_users('ciffi');



tweet_id	date_time	text	user_name	tweet_language	permalink
1617762357312360450	2023-01-24 05:52:57+00:00	Wichtige im #Sat1-Studio: ist #ChatGPT Chanc...	ciffi	de	https://twitter.com/ciffi/status/1617
1617745413578313728	2023-01-24 04:45:37+00:00	»Es darf von Lehrkräften nicht erwartet werden...	ciffi	de	https://twitter.com/ciffi/status/1617
1617736599999246336	2023-01-24 04:10:36+00:00	Moin, Lehrer:innen sind schon wach. Die andern...	ciffi	de	https://twitter.com/ciffi/status/1617
1617429680837795841	2023-01-23 07:28:18+00:00	@albu @nettwerkerin @mpoessel Guten Morge...	ciffi	de	https://twitter.com/ciffi/status/1617

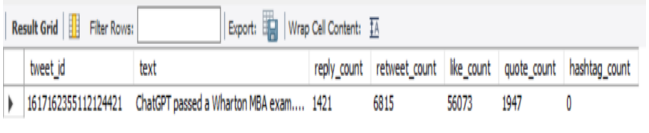
Fig-21 Output for number of tweets based on particular user.

2.Stored procedure for the most popular tweet between the range of dates:

Procedure:

```
create definer=`root`@`%`
procedure`most_popular_tweet_between_days`(in
start_date date,  in end_date date) begin select  t.tweet_id,
t.text, tc.reply_count,
tc.retweet_count,tc.like_count,tc.quote_count,tc.hashtag_co
unt from tweets t join tweet_counts tc on t.tweet_id =
tc.tweet_id where  t.date_time between start_date and
end_date order by tc.like_count desc, tc.retweet_count desc
limit 1 ;
end
```

Calling the stored procedure: call twitter.most_popular_tweet_between_days('2023-01-22', '2023-01-24');



tweet_id	text	reply_count	retweet_count	like_count	quote_count	hashtag_count
1617162355112124421	ChatGPT passed a Wharton MBA exam....	1421	6815	56073	1947	0

Fig-22 Output for the most popular tweet between the range of dates.

SJSU CHATGPT TWITTER ANALYSIS

3.Stored procedure to know who's tweet was most reposted:

Procedure:

```
create definer=`root`@`%` procedure
`most_quoted_tweet`()
begin select substring_index(t.quoted_tweet, '/', -1) as
quoted_tweet_id,substring_index(substring_index(t.quoted
_tweet, '/', -3), '/', 1) as
quoted_tweet_username,count(t.quoted_tweet) as count
from tweets t where t.quoted_tweet is not null group by
t.quoted_tweet order by count desc limit 5 ; end
```

Calling the stored procedure : call twitter.most_quoted_tweet();

Result Grid | Filter Rows: | Export:

quoted_tweet_id	quoted_tweet_username	count
1617162355112124421	GRDecter	209
1617194845810077697	noor_siddiqui_	141
1617386607006584832	WatcherGuru	99
1617715712461766657	GRDecter	70
1616845386903347200	nabeelqu	69

Fig-23 Output that shows who's tweet was most reposted.

4.Stored procedure to get most trending tweets

Procedure:

```
create definer=`root`@`%` procedure
`trending_tweets_sp`() begin insert into trending_tweets
(tweet_id, user_name, like_count, retweet_count)
select tweets.tweet_id, tweets.user_name,
tweet_counts.like_count, tweet_counts.retweet_count from
tweets join tweet_counts on tweets.tweet_id =
tweet_counts.tweet_id where tweet_counts.like_count >
100 and tweet_counts.retweet_count > 100; end
```

Calling the stored procedure: call twitter.trending_tweets_sp();-- Details will be updated in trending tweets table.

Check the table: select * from twitter.trending_tweets;

tweet_id	user_name	like_count	retweet_count
1617156454141534212	Veskii_	9125	542
1617161446202245120	hasantoxr	5682	1094
1617162355112124421	GRDecter	56073	6815
1617194845810077697	noor_siddiqui_	12557	2627
1617235666580144128	GrindMarket	4413	713

Fig-24 Output that shows most trending tweets.

9. ACCESS PRIVILEGES

The rights given to individuals or groups of individuals to view and modify data kept in a database are referred to as Access Privileges. Access privileges in AWS are controlled by Identity and Access Management (IAM) policies, which can be used to specify what actions are permitted for a particular user or group. For instance, for our project we had set up access privileges for our group members (fig-25) who could access the database. We have granted specific privileges, such as select, insert, update, delete, etc., based on the role and responsibilities of group members. The security and integrity of the data contained in the database are dependent on access privileges. For instance, important operations like altering the schema or removing data must be restricted to authorized users only. It is very crucial and important to have access privileges as it maintains the confidentiality, integrity, and availability of data in the database.

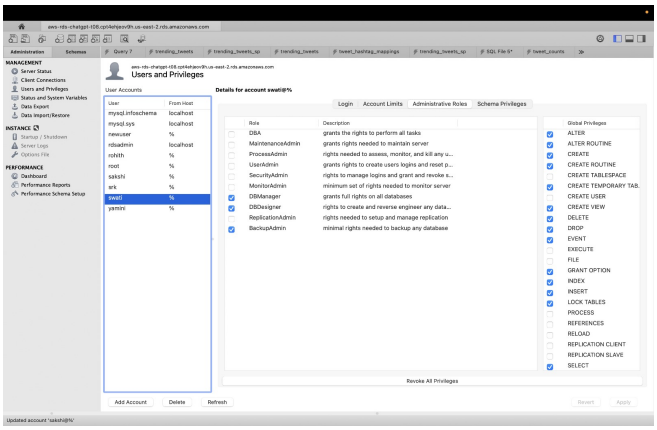


Fig-25 Access privileges of the team mates

9. DATA CONNECTIVITY TO DATABASE

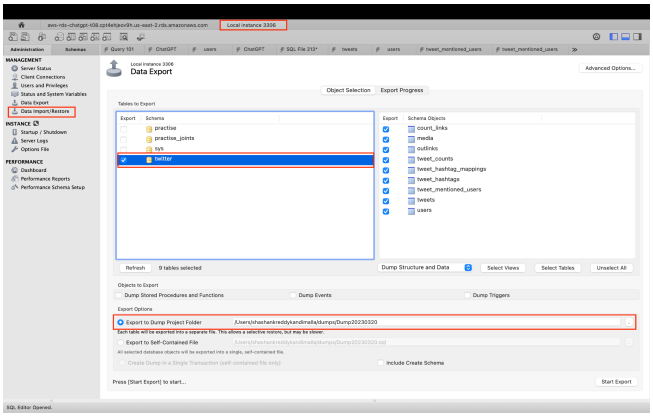


Fig-26 Data Connectivity to database.

SJSU CHATGPT TWITTER ANALYSIS

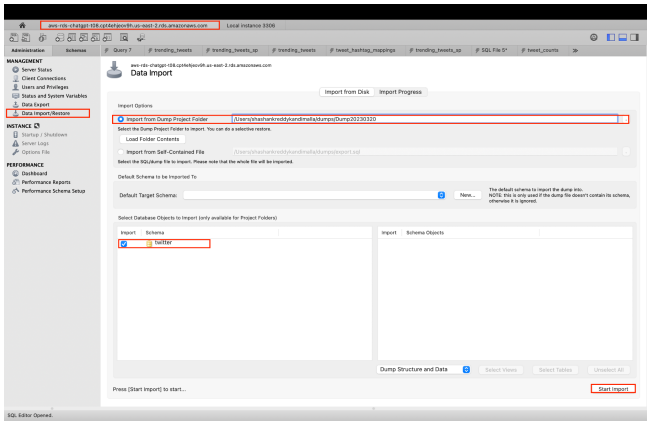


Fig-26.1 Data Connectivity to database.

10. CONNECTIVITY TO AWS

This Python Program(fig-27) uses the mysql.connector module to establish a connection to a MySQL database running on an Amazon Web Services (AWS) server and then uses the cursor.callproc() method to invoke the 'most_quoted_tweet' stored procedure.

Following the stored procedure call, the script uses the cursor.stored results() method to obtain the outcomes before printing them to the console with the print() function.

Lastly, the script uses the cnx.close() method to terminate the database connection.

```
In [3]: import mysql.connector

        hostname = "aws-rds-chatgpt-t08.cpt4ehjeov9h.us-east-2.rds.amazonaws.com"
        username = "root"
        password = "Team8nohate"
        database = "twitter"

        # Connect to the database
        cnx = mysql.connector.connect(host=hostname, user=username, password=password, database=database)

        # Invoke the stored procedure
        cursor = cnx.cursor()
        cursor.callproc('most_quoted_tweet')

        # Get the results of the stored procedure
        for result in cursor.stored_results():
            print(result.fetchall())

        # Close the database connection when you're done
        cnx.close()

[('1617162355112124421', 'GRDecter', 209), ('1617194845810077697', 'noor_siddiqui_', 141), ('1617386607006584832', 'WatcherGuru', 99), ('1617715712461766657', 'GRDecter', 70), ('1616845386903347200', 'nabeelqu', 69)]
```

Fig-27 Script to connect to AWS.

11. LOGGING OF DB

Amazon RDS log files can offer important insights into the operation and use of your database.

General Logs :

When clients connect or disconnect, it logs every piece of information to this log (fig-28), as well as every SQL statement that clients send. When you think a client might have made a mistake and want to know exactly what the client sent, the general query log can be quite helpful. The 'general_log' parameter has been set to '1' to view the logs for every query run on AWS - RDS.

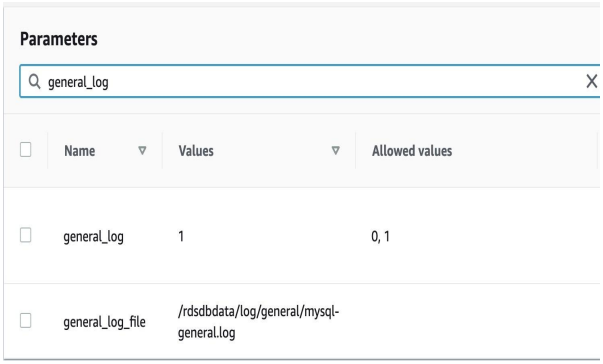


Fig-28 General logs.

AWS gives us an option to limit the number of logs we want to view at once. We can select the log group and also the time for which we want to view the logs (for example, 5 minutes, 30 minutes and so on..)

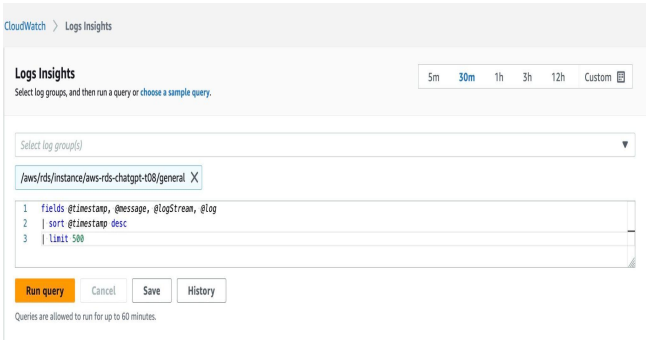


Fig-29 Selecting logs specific to time.

We can see the various queries, stored procedures and triggers created for this project in the above log file generated by the AWS database. The log file (fig-30) shows the timestamp of the query execution, details of the query which are mentioned in the message and the log stream (AWS db instance).

SJSU CHATGPT TWITTER ANALYSIS

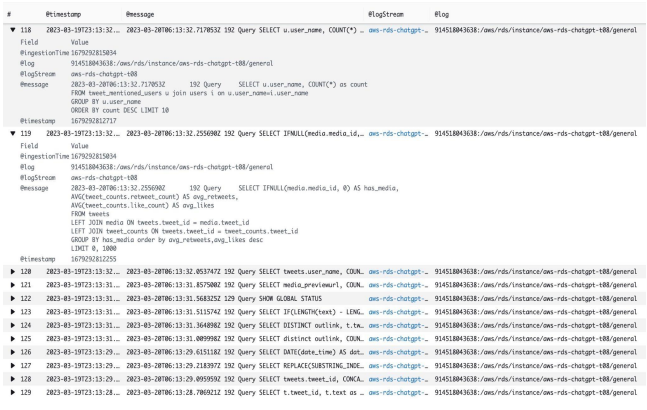


Fig-30 Queries with timestamp

12. PERFORMANCE MEASUREMENT

Performance reports can be viewed using a feature in MySQL Workbench. Several metrics can be used to display performance statistics, including memory usage, hot spots for I/O, expensive SQL statements, and user resource usage. The Performance dashboard (fig-31) can be generated from the administration section under the performance statistics. It shows us the following things

1. Network Status : The network traffic sent and received by the MySQL server across client connections is highlighted in these statistics.

2. MySQL Status : The key performance and activity data for the MySQL server are highlighted here.

3. InnoDB Status : This is a summary of the disk activity produced by the InnoDB storage engine's InnoDB Buffer Pool. (We can hover over a graph to see more details)

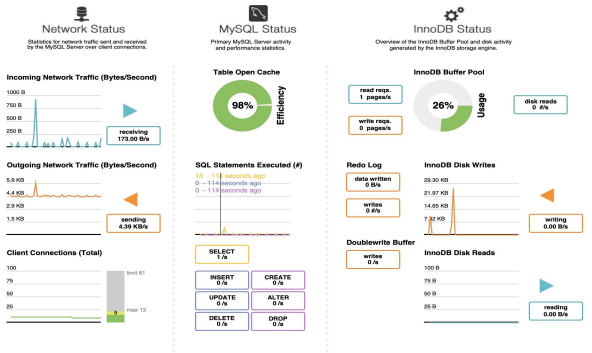
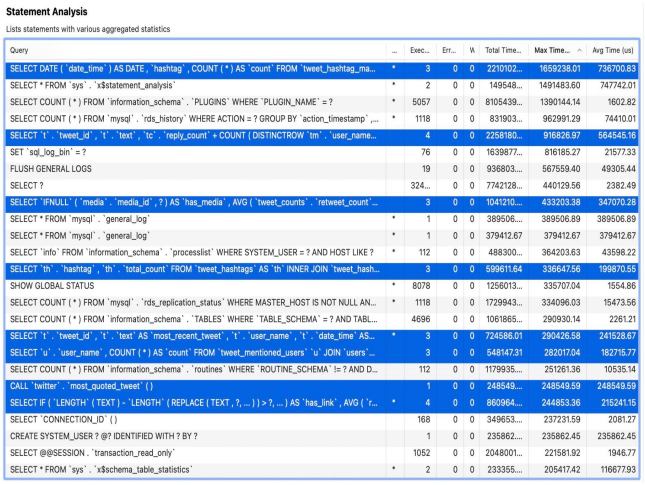


Fig-31 Performance Dashboard for ChatGPT Twitter Model.

13. STATEMENT ANALYSIS

This lists the statements with aggregated statistics.



Statement Analysis

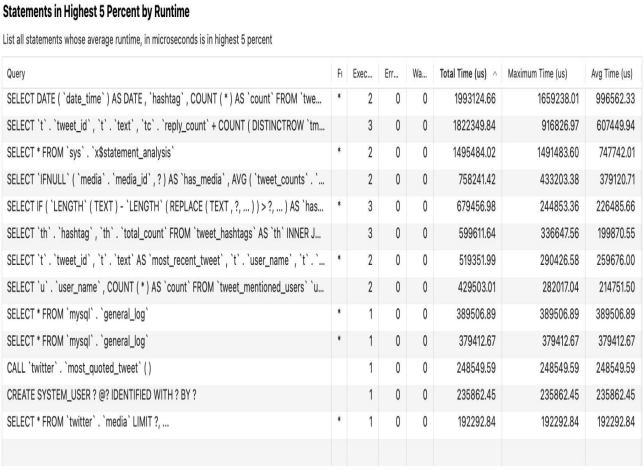
Lists statements with various aggregated statistics

Query	...	Exec...	Err...	W	Total Time...	Max Time...	Avg Time (us)
SELECT DATE (`date_time`) AS DATE , `hashtag` , COUNT (*) AS `count` FROM `tweet_hashtag_ma...	*	3	0	0	2210102...	1659238.01	736700.83
SELECT * FROM `sys` . `x$statement_analysis`	*	2	0	0	149548...	1491483.60	747742.01
SELECT COUNT (*) FROM `information_schema` . `PLUGINS` WHERE `PLUGIN_NAME` = ?	*	5057	0	0	8105439...	1390144.14	1602.82
SELECT COUNT (*) FROM `mysql` . `rds_history` WHERE ACTION = ? GROUP BY `action_timestamp` ,...	*	1118	0	0	831903...	962991.29	74410.01
SELECT `t` . `tweet_id` , `t` . `text` , `tc` . `reply_count` + COUNT (DISTINCTROW `tm` . `user_name...		4	0	0	2258180...	916826.97	564545.16
SET `sql_log_bin` = ?		76	0	0	1639877...	816185.27	21577.33
FLUSH GENERAL LOGS		19	0	0	936803...	567559.40	49305.44
SELECT ?		324...	0	0	7742128...	440129.56	2382.49
SELECT `IFNULL` (`media` . `media_id` , ?) AS `has_media` , AVG (`tweet_counts` . `retweet_count`...		3	0	0	1041210...	433203.38	347070.28
SELECT * FROM `mysql` . `general_log`	*	1	0	0	389506....	389506.89	389506.89
SELECT * FROM `mysql` . `general_log`	*	1	0	0	379412.67	379412.67	379412.67
SELECT `info` FROM `information_schema` . `processlist` WHERE SYSTEM_USER = ? AND HOST LIKE ?	*	112	0	0	488300...	364203.63	43598.22
SELECT `th` . `hashtag` , `th` . `total_count` FROM `tweet_hashtags` AS `th` INNER JOIN `tweet_hash...		3	0	0	599611.64	336647.56	199870.55
SHOW GLOBAL STATUS	*	8078	0	0	1256013...	335707.04	1554.86
SELECT COUNT (*) FROM `mysql` . `rds_replication_status` WHERE MASTER_HOST IS NOT NULL AN...	*	1118	0	0	1729943...	334096.03	15473.56
SELECT COUNT (*) FROM `information_schema` . `TABLES` WHERE `TABLE_SCHEMA` = ? AND TABL...		4696	0	0	1061865...	290930.14	2261.21
SELECT `t` . `tweet_id` , `t` . `text` AS `most_recent_tweet` , `t` . `user_name` , `t` . `date_time` AS...	*	3	0	0	724586.01	290426.58	241528.67
SELECT `u` . `user_name` , COUNT (*) AS `count` FROM `tweet_mentioned_users` `u` JOIN `users`...		3	0	0	548147.31	282017.04	182715.77
SELECT COUNT (*) FROM `information_schema` . `routines` WHERE `ROUTINE_SCHEMA` != ? AND D...		112	0	0	1179935...	251261.36	10535.14
CALL `twitter` . `most_quoted_tweet` ( )		1	0	0	248549....	248549.59	248549.59
SELECT IF (`LENGTH` (TEXT) - `LENGTH` (REPLACE (TEXT , ?, ...)) > ?, ...) AS `has_link` , AVG (`r...	*	4	0	0	860964....	244853.36	215241.15
SELECT `CONNECTION_ID` ( )		168	0	0	349653....	237231.59	2081.27
CREATE SYSTEM_USER ? @? IDENTIFIED WITH ? BY ?		1	0	0	235862....	235862.45	235862.45
SELECT @@SESSION . `transaction_read_only`		1052	0	0	2048001...	221581.92	1946.77
SELECT * FROM `sys` . `x$schema_table_statistics`	*	2	0	0	233355....	205417.42	116677.93

Fig-32 Statement Analysis

We sorted the statements by the maximum time taken in the descending order. Queries with multiple where conditions, joins and many row retrievals take more time compared to other relatively smaller queries. Heavy stored procedures also and their calling also takes more time and memory space in the db.

Statements in the Highest 5 percent by Runtime - Lists all statements whose average runtime (measured in microseconds) falls inside the top 5%.



Statements in Highest 5 Percent by Runtime

List all statements whose average runtime, in microseconds is in highest 5 percent

Query	Fi	Exec...	Err...	Wa...	Total Time (us)	Maximum Time (us)	Avg Time (us)
SELECT DATE (`date_time`) AS DATE , `hashtag` , COUNT (*) AS `count` FROM `twe...	*	2	0	0	1993124.66	1659238.01	996562.33
SELECT `t` . `tweet_id` , `t` . `text` , `tc` . `reply_count` + COUNT (DISTINCTROW `tm...		3	0	0	1822349.84	916826.97	607449.94
SELECT * FROM `sys` . `x$statement_analysis`	*	2	0	0	1495484.02	1491483.60	747742.01
SELECT `IFNULL` (`media` . `media_id` , ?) AS `has_media` , AVG (`tweet_counts` . `...		2	0	0	758241.42	433203.38	379120.71
SELECT IF (`LENGTH` (TEXT) - `LENGTH` (REPLACE (TEXT , ?, ...)) > ?, ...) AS `has...	*	3	0	0	679456.98	244853.36	226485.66
SELECT `th` . `hashtag` , `th` . `total_count` FROM `tweet_hashtags` AS `th` INNER J...		3	0	0	599611.64	336647.56	199870.55
SELECT `t` . `tweet_id` , `t` . `text` AS `most_recent_tweet` , `t` . `user_name` , `t` . `...	*	2	0	0	519351.99	290426.58	259676.00
SELECT `u` . `user_name` , COUNT (*) AS `count` FROM `tweet_mentioned_users` `u...		2	0	0	429503.01	282017.04	214751.50
SELECT * FROM `mysql` . `general_log`	*	1	0	0	389506.89	389506.89	389506.89
SELECT * FROM `mysql` . `general_log`	*	1	0	0	379412.67	379412.67	379412.67
CALL `twitter` . `most_quoted_tweet` ( )		1	0	0	248549.59	248549.59	248549.59
CREATE SYSTEM_USER ? @? IDENTIFIED WITH ? BY ?		1	0	0	235862.45	235862.45	235862.45
SELECT * FROM `twitter` . `media` LIMIT ?, ...	*	1	0	0	192292.84	192292.84	192292.84

Fig-33 Statements in the Highest 5 percent by Runtime.

14. CONCLUSION AND FUTURE WORK

In order to gain insight into the online discussion surrounding this language model on Twitter, we have proposed an app to analyze tweets regarding ChatGPT. The program is successful in generating all the solutions that are required. Functional analysis was used to identify the key functional elements needed for the application and to determine the functional requirements. We also spoke about how the SJSU ChatGPT Tweets Analysis database was moved to AWS using a cloud-native strategy.

Although our application offers insightful analyses of the online discourse surrounding ChatGPT, there is a need for future study to enhance its capabilities. Sentiment analysis, for instance, may be included to comprehend the general sentiments regarding ChatGPT reflected in tweets. To acquire a better understanding of the bigger picture of natural language processing, the application might also be expanded to analyze tweets concerning different language models.

Future studies can also investigate the application of machine learning models to enhance the precision of identifying the most popular tweets, users, and hashtags. In order to provide a more thorough examination of the online discussion surrounding ChatGPT and other language models, the program might also be coupled with other social media platforms.

Overall, this application serves as a foundation for further study and advancement in the fields of social media analysis and natural language processing.

15. REFERENCES

[1] chatgpt twitter data analysis
https://www.kaggle.com/datasets/tariqsays/chatgpt-twitter-dataset.

[2] Normalization
https://www.splunk.com/en_us/blog/learn/data-normalization.html

[3] Creation of db in AWS
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_CreateDBInstance.html

[4] ER Diagram Understanding
https://www.lucidchart.com/pages/er-diagrams

[5] Fundamentals of Database Systems, 7th edition, by Elmasri and Navathe, Addison Wesley Publishing Company, Menlo Park, CA.

[6] Jeffrey A. Hoffer, Ramesh Venkataraman, and Heikki Topi. Modern Database Management.Pearson. 13th Edition. ISBN-13: 978-1292263359