

SJSU ChatGPT Tweets Analysis

1st Venkata Karthik Patralapati, 2nd Prasad Jayakumar, 3rd Anbu Valluvan Devadasan, 4rd Madhulika Dutta, 5rd Sai Krishna Bhamidipati

San Jose State University, San Jose, CA, United States of America

Department of Applied Data Science, Data Analytics

DATA 225 Lab-01, Group-06

Abstract—We all know that artificial intelligence is the future, and we are getting closer to that future with the new sensational discovery ChatGPT, which has both positive and negative outcomes. It is not within our purview to discuss its outcomes. However, People have been tweeting about ChatGPT, sharing their experiences with it and discussing articles or resources connected to it we have selected a data set that contains tweets about the ChatGPT on Twitter, messages containing the hashtag #chatgpt that were scraped from Twitter. The data set contains tweetid, user, context, and the number of replies, among other columns. We used Python to convert the data set into a data frame, conduct some DBMS normalization techniques, and clean the data before loading it into My SQL Database, which was connected to Amazon Web Services (AWS) via Python. To retrieve data, we used multiple SQL query techniques that includes DML, DDL, DCL commands, stored procedures, and triggers and presented our analysis.

Index Terms—tweet, user, hashtag, AWS, ChatGPT

I. PROBLEM STATEMENT

The huge popularity of social media sites makes them an important medium of information exchange in today's world, and valuable insights can be gained by analyzing participants' conversations around a topic on these platforms. Twitter is one such social media platform where users share their thoughts on a topic, mainly in the form of textual data called tweets. As part of this project, we analyzed a dataset containing tweets on ChatGPT to uncover trends in the discussion around this AI chatbot. We achieved this by building a database application to store the cleaned dataset in the form of tables and retrieving relevant data using structured query language commands. A database system has been used since it provides reliable data storage and retrieval along with secure user access control, thereby ensuring accurate analysis and results. As we are dealing with low volumes of mostly well-structured data, a relational database model was used to build the application to ensure higher data consistency.

II. SOLUTION REQUIREMENTS

The database solution we intend to propose must be able to handle the large amount of data from the twitter dataset consisting of tweets with the hashtag chatgpt and enable the data analysts to analyze the dataset. The solution must be accessible to different roles and enable multiple data analysts to collaborate together on the analytics. The application solution

must be able to process the data quickly and efficiently with all the access controls to different roles of users. The solution also must enable users and data scientists to understand and perform analytical processes about the tweets related to chatgpt. The system enables the users to query and analyze the twitter data based on various criteria such as date range, media, language, etc. The system will provide access to data that was scraped from twitter with hashtag chatgpt across the globe. The application will store the pre-processed data that is structured and normalized to perform analysis on the twitter dataset. Using this application, data scientists and analysts can perform analytical operations on the dataset to come up with insights and trends about the revolutionary product chatgpt. The application is limited by the dataset that was scraped from twitter and the tweet that contains chatgpt in it. The system doesn't provide real time monitoring and it is limited by the tweets within the time period that is available from the dataset. The application may be affected by the number of tweets that contain spam content or twitter bots. The system does not support sentiment analysis or visualization to report the analysis. The application solution helps researchers and analysts to search tweets containing the hashtag chatgpt and other viral hashtags about chatgpt and extract information and insights about topic, language, user engagement, shared content, media, device used. Businesses can use the system to monitor their brand reputation on Twitter, make informed decisions based on the insights gained and respond to customer feedback. Media outlets can use the system to track news stories and public opinion.

III. CONCEPTUAL DATABASE DESIGN

We designed a relational database using the Kaggle ChatGPT dataset, and we relied on Python to perform normalization techniques and divide the data into multiple entities such as count, hashtag, media, mentioned_users, outlinks, tweet, and user, where each entity is linked through a foreign key and each table has a primary key, with a few tables having composite primary keys.

A. *tweet*

This entity involves the various tweet attributes, with tweet_id serving as the primary key.

- **tweet_id:** This is the table's primary key, and it records the id of the specific tweet

- **timestamp:** It contains information about the time and date the tweet was tweeted.
- **conv_id:** It contains the tweet_id and is concerned with the reply thread of a certain tweet_id
- **user_id:** This is a new term in this context, referring to the id of the user who tweeted it
- **language:** It refers to the language in which the tweet was sent
- **source:** This column contains information about the device used to tweet
- **permalink:** This is a link that takes us to the tweet

B. user

This object contains information on the user who tweeted about the Chat GPT, and its main key is user_id.

- **user_id:** It is the primary key in this entity and features information on the id that an individual owns.
- **username:** It is identical to the name that a user chooses to tweet.
- **url:** It is a url that connects us to the user's Twitter account page, where you can discover more about the link

C. mentioned_user

This entity gathers information about people whose names are mentioned in the tweets of other users, leveraging tweet_id as a foreign key.

- **tweet_id:** It functions as the primary key in this object, which refers to the tweet table and stores the tweet's recognition
- **mentioned_user:** It returns the user's username who was referenced in a tweet

D. media

By using tweet_id as its foreign key, this object preserves information about the media with which the given tweet exists.

- **tweet_id:** It is used as a foreign key in this object, referencing the tweet table and featuring the tweet's identity.
- **media_type:** This pertains to the type of media in the tweet, such as a photo, a video, or a gif.
- **tweet_text:** It offers us with the context of a tweet.
- **quoted_tweet:** It describes a person's tweet that was quoted by a different user, who added some of his opinions to the tweet.

E. outlinks

This entity is about the outside links that a tweet contains, with tweet_id acting as both the primary and foreign key, and outlink acting as a composite primary key because each tweet can have many outlinks attached to it. As an outcome, we used the concept of composite primary for this entity to compensate.

- **tweet_id:** It serves as both a foreign key and a primary key in this object, referencing the tweet table and containing the tweet's identification.
- **outlinks:** This refers to the external source link that a tweet contains and it acts a composite primary key which combines with the tweet_id in the same table.

F. log_details

This entity is used for auditing purposes of the queries

- **log_id:** This is a primary key which records each log entries.
- **table_name:** It contains information about the table from which the trigger was activated.
- **column_name:** It contains information about the column in which modifications are being done.
- **old_value:** It contains information of the old value of the record.
- **new_value:** It contains information of the new value of the record that was updated.
- **changed_at :** This attribute record the time at which a record was updated.
- **changed_by:** This attribute records the user who updated the record.

G. hashtag

This entity is about the hashtags that a tweet contains, and because a tweet can have several hashtags, we have separated the hashtags from multi-valued attributes to single-valued attributes and loaded them into this entity.

- **tweet_id:** It acts a foreign key and primary key in this entity which references the tweet table and holds the identity of the tweet
- **hashtag:** Different hashtags a tweet is associated to
- **id:** It is an auto incremented primary key in the table which is used as an composite primary key along with tweet_id

H. count

This entity holds the information about the count details of the tweet with tweet_id acting as both primary and foreign key.

- **tweet_id:** It acts a foreign key and primary key in this entity which references the tweet table and holds the identity of the tweet
- **reply_count:** Number of replies a tweet got
- **retweet_count:** Count of the retweets for a tweet
- **like_counts:** Number of likes for a tweet
- **quote_count:** Number of times the tweet got quoted.

Entity Relationship Diagram for Twitter Dataset:

IV. FUNCTIONAL ANALYSIS

A. *The dataset that we are working with in this project is of twitter data – a collection of all the tweets about the Chatgpt for a period of 3 days i.e 01.22.2023, 01.23.2023 & 01.24.2023. We used python to clean the dataset and converted the data into a format suitable for analysis by segregating the entities based on their relationships into different entities. We have created the following entities accordingly to suit our analysis:*

- **tweet:** This table has all the information about the tweet like the 'tweet_id' which is the primary key for this entity along with other information like the 'user_id' from

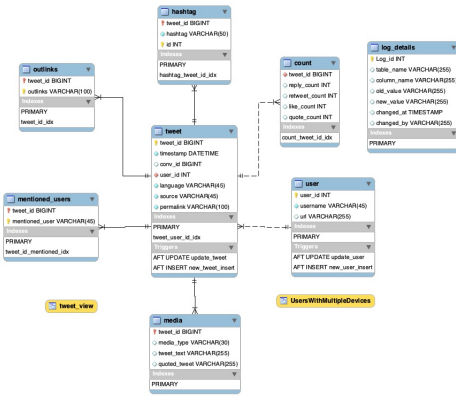


Fig. 1. Entity Relationship for Twitter Dataset

where the tweet was generated, the language in which the tweet was created, the time when the tweet was created etc.

- **user:** This table has all the information about the users who tweeted about Chatgpt in the given period. We use 'user_id' as the primary key in this table
- **mentioned_user:** This table has all the records where users have been mentioned by others in their tweets.
- **media:** Some of the tweets have used various kinds of media in their tweets which have been included in the media. We use the tweet_id which is a foreign key to identify the records in this table.
- **outlinks:** This contains the external links that are used in the tweet
- **log_details:** This is an independent entity that we use to keep log of all database operations for security and audit purpose.
- **hashtag:** The details about the hashtags used in the tweets are available in this table. We use the tweet_id from the tweet table to identify the records
- **count:** This table has all the quantitative information about the tweet like the number of times a tweet was liked, the number of times a tweet was retweeted etc. We use the tweet_id from the tweet table to identify the records.

B. AWS Migration

We are using the AWS instance to migrate the dataset into cloud after processing the data

C. Query Analysis

Using the SQL queries we are addressing the solution requirements and performing the analysis of the given data set. The design application in its simplicity can address the following and can be extended to understand consumer behavior or emotion about any particular product via their interaction on various social media platforms and also the overall perception of the product in the market. The design application has following components:

- **Top 3 Viral tweets and its details based on retweet counts**

This SQL query retrieves the top three viral tweets based on the number of retweets. Using the 'tweet_id' attribute, it joins four tables: the 'tweet', the 'user', the 'count', and the 'media'. The 'user' table stores user information, while the 'tweet' table stores tweet text. Retweet counts are listed in our count table; media is listed in our media table.

Observation:

Among the results, the tweet by GRDecter with the text 'ChatGPT passed a Wharton MBA exam. Time to overhaul education' was the most viral tweet with the highest retweets of 6815.

- **Most favorable media (photo/video/gif) used in chat gpt tweets based on likes, retweets and quotes**

This SQL query retrieves the most favoured media amongst all the users in their tweet based on the number of likes, retweets and quotes. Using the 'tweet_id' attribute we are joining three tables 'tweet', 'media' and 'count' we are getting the desired result.

Observation:

We found out that 'Photo' media types received the highest number of likes, retweets, and quotes.

- **Most shared material / website / platform discussed about chat gpt (using outlink):**

This SQL query retrieves the most shared material or website or platform in the tweets recorded in given time frame. We are using the outlink table to obtain the report.

Observation:

Based on the results, 'we observe that the most shared content was an article from financial times website about Chatgpt. <https://www.ft.com/content/7229ba86-142a-49f6-9821-f55c07536b7c>'.

- **The top 5 languages based on tweets about chat_gpt.**

This SQL gives us the report of 5 of the most used languages in all the tweets that were recorded in the given time frame. We use the 'tweet' entity to obtain the results. The top 5 languages in which users tweeted are

EN : ENGLISH – No of tweets : 32076

JA : JAPANESE – No of tweets : 5046/

ES : SPANISH - No of tweets : 3315

FR : FRENCH – No of tweets : 2492

DE : GERMAN – No of tweets : 1207

- **Top 10 hashtags used for chatgpt tweets and their counts from the dataset**

This SQL query retrieves the top 10 hashtags based on their count from the 'hashtag' table.

Observation:

The query results reveal that, in addition to the hashtag '#chatgpt', the hashtags '#ai' and '#openai' are the next most frequently used hashtags in the list.

- **Information about the longest tweet text about chat gpt from the dataset**

The SQL query retrieves the longest tweet from a user using the 'user', 'tweet' and 'media' tables

Observation:

The query results indicate that the longest tweet, posted

from an iPhone by the user 'mochico0123', contained 737 letters.

- **Tweet lengths of different languages showing the minimum, maximum and average text length for each language**

We are generating a report language wise showing the minimum, maximum and average text length for all the 61 languages used to tweet in the time frame.

We are using the tables 'tweet' and 'media' to obtain the required report. **Observation:**

Among all languages, it is noteworthy that the 'Tamil language' has the highest average tweet length compared to all other languages.

- **Total tweets about chatgpt on each day from the dataset**

This query generates the report of total number of tweets per day and we tried to correlate if any event influenced the spike in the number of tweets.

Observation:

According to the query results, the day of January 23, 2023, had the highest number of tweets. It is notable that this spike in activity may be due to the announcement that 'Microsoft plans' to invest \$10 billion in OpenAI, the creator of ChatGPT.

- **The top 10 most mentioned twitter users in chatgpt related tweets**

We are using the 'user' table and 'mentioned_users' table to obtain the information about the top 10 most mentioned users in chatgpt related tweets.

Observation:

Based on the query results, it is clear that OpenAI is the most frequently mentioned user in the dataset, which may be due to their significant presence in the field of artificial intelligence. It is worth noting that the Twitter URL is only displayed for users who have tweeted within the given time period covered by the dataset.

- **Hashtag statistics - The minimum, maximum and average number of hashtags used in all the tweets from the dataset**

We are using the 'tweet' table and 'hashtag' table to generate a statistician report of all the hashtags used in all the tweets.

Observation:

The query calculates the minimum, maximum, and average number of hashtags used in tweets from the dataset. The result shows that the minimum number of hashtags used is 0, the maximum is 28, and the average number is approximately 0.77.

- **The top 5 most preferred sources of twitter to tweet during the time period from the dataset.**

We are using the tweet table to find out the 5 most preferred sources of twitter to tweet during the given time period.

Observation:

The query results reveal the top sources that were used to tweet, which include Web, iPhone, and Android. These results suggest that there are more users who are actively using web applications to tweet more than smartphone

applications.

- **The top 5 most quoted tweets about chatgpt, by the user of the tweet and its text.**

We are using the entities 'user', 'tweet' and 'media' to find out the top 5 most quoted tweets about Chatgpt.

Observation:

The result shows that the tweet by GRDecter - 'ChatGPT passed a Wharton MBA exam. Time to overhaul education.' has been the most quoted text within the dataset.

- **Peak hours of activity**

We are using the tweet table and trying to find out the peak hours of activity.

Observation:

The query results provide information on the peak hours of activity for tweets in the dataset. The results show the top 10 hours with the highest number of tweets, based on the count of tweet IDs. Highest is 3297 tweets at 15:00:01 on 22nd Jan of 2023.

D. Views

- **View that shows users who have atleast used 3 devices, along with the number of devices used and the names of the devices**

Observation:

The code creates a view called 'UsersWithMultipleDevices' that displays users who have used at least 3 devices, along with the number of devices used and the names of the devices. The query joins the 'tweet' and 'user' tables and groups the data by user ID to count the number of unique devices used by each user. The resulting view will only show users who meet the criteria and will be sorted based on the number of devices used.

E. Stored Procedures

- **Stored Procedure to Display the Statistics for a Given Day**

We call the stored procedure and enter a date of our choice to get all the tweet stats for the day.

Observation: The result shows the statistics for the day '2023-01-22', including the total number of distinct users who tweeted, the total number of distinct tweets, and the total number of distinct hashtags used on that day. The results are sorted first by the total number of distinct tweets in descending order, then by the total number of distinct users in ascending order, and lastly by the hashtag in descending order.

- **Stored Procedure to Retrieve User Statistics We call the stored procedure to get all the user statistics for a given user. Observation:** The result shows the statistics for the user with the username 'GRDecter', including the total number of tweets made by the user, the total number of replies received by the user's tweets, the total number of retweets of the user's tweets, the total number of likes received by the user's tweets, and the total number of times the user's tweets were quoted.
- **Stored Procedure to Check if a User Was Mentioned in a Tweet**

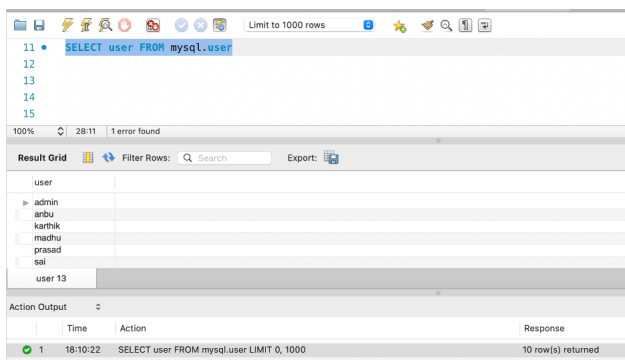
H. Access privileges:

MySQL provides Data Control Language (DCL) through GRANT and REVOKE statements for controlling access to specific roles or users. In our application, all group members have access to all Data Manipulation Language (DML) statements, while a restricted user is used to demonstrate access privileges.

- **We can create users**

```
CREATE USER user@%
IDENTIFIED BY 'password';
```

Snippet:



The screenshot shows a MySQL query execution window. The query entered is `SELECT user FROM mysql.user`. The result is displayed in a table with one column, 'user', and 10 rows. The rows are: admin, anbu, karthik, madhu, prasad, sai, and user 13. Below the table, an 'Action Output' section shows the query execution details: '1 18/10/22 SELECT user FROM mysql.user LIMIT 0, 1000' with a response of '10 row(s) returned'.

| user |
|---------|
| admin |
| anbu |
| karthik |
| madhu |
| prasad |
| sai |
| user 13 |

Fig. 3. Example log for executing a query

Observation

To create users, we can use the CREATE USER statement, and to list all users, we can use a SELECT statement on the mysql.user table. We can use GRANT statements to provide restricted access to a specific table or view for a particular user, limiting their access to only the granted privileges.

- **We grant restricted access to select particular table and view for specific user 'Prasad'**

```
GRANT SELECT ON tweet_ver_1.tweet_view TO 'prasad'@'%';
GRANT SELECT ON tweet_ver_1.tweet TO 'prasad'@'%';
```

Snippet

Observation

In these two statements and the result snippet, we are granting restricted access to a specific user named 'Prasad' to select from two tables: 'tweet_ver_1.tweet_view' and 'tweet_ver_1.tweet'. This means that the user 'Prasad' will only be able to perform SELECT queries on these two tables and won't be able to perform any other operations on them. The '@' symbol with '%' indicates that this user can access these tables from any host.

- **The access privilege restricts the user to access anything other than the granted.**

Observation

This is an access privilege that restricted access to the user prasad as the command has been denied to the user

V. SQL QUERIES

We used the structured query language to build queries, stored procedures, views and triggers in order to access the required data from tables relevant to our analysis and the commands used are detailed in the following section.

A. Queries

1. Top 3 Viral tweets and its details based on retweet counts. This also shows the media shared in the tweet and if it was quoted or an original tweet.

```
SELECT
u.username AS 'User', m.tweet_text AS 'Tweet',
m.media_type AS 'Media Shared', m.quoted_tweet AS 'Quoted', c.retweet_count AS 'Retweets'
FROM tweet t
JOIN user u USING (user_id)
JOIN count c USING (tweet_id)
JOIN media m USING (tweet_id)
GROUP BY tweet_id
ORDER BY retweets DESC
LIMIT 3
```

2. The most favorable media (photo/video/gif) used in chat gpt related tweets based on likes, retweets and quotes.

```
SELECT
m.media_type AS 'Media', SUM(c.like_count) AS 'Likes',
SUM(c.retweet_count) AS 'Retweets', SUM(c.quote_count) AS 'Quoted'
FROM tweet
JOIN media m USING (tweet_id)
JOIN count c USING (tweet_id)
WHERE m.media_type = ANY (SELECT DISTINCT media_type
FROM media WHERE media_type IS NOT NULL)
GROUP BY m.media_type
ORDER BY Likes DESC , Retweets DESC , Quoted DESC
```

3. Most shared material / website / platform discussed about chat gpt (using outlook)

```
SELECT
outlinks 'Content', COUNT(outlinks) AS 'Count'
FROM outlinks
WHERE outlinks NOT LIKE '%twitter.com%'
GROUP BY outlinks
ORDER BY count DESC
LIMIT 3
```

4. The top 5 languages that discussed based on tweets about chat gpt

```
SELECT
language 'Language', COUNT(language) AS '# of Tweets'
FROM tweet
GROUP BY language
```



```
ORDER BY COUNT(language) DESC
LIMIT 5
```

5. Top 10 hashtags used for chat gpt related tweets and their counts from the dataset.

```
SELECT
hashtag 'Hashtag', COUNT(hashtag) AS 'Count'
FROM hashtag
GROUP BY hashtag
ORDER BY COUNT(hashtag) DESC
LIMIT 10
```

6. Information about the longest tweet text about chat gpt from the dataset.

```
SELECT
u.username 'User', tweet_text 'Tweet', t.language 'Language',
t.source 'Device',
MAX(LENGTH(tweet_text)) 'Length' FROM media m
JOIN tweet t USING (tweet_id)
JOIN user u USING (user_id)
```

7. Tweet lengths of different languages showing the minimum, maximum and average text length for each language.

```
SELECT
t.language 'Language',
MIN(LENGTH(mtweet_text)) 'Minimum',
MAX(LENGTH(mtweet_text)) 'Maximum',
AVG(LENGTH(mtweet_text)) 'Average' FROM media
m
JOIN tweet t ON t.tweet_id = m.tweet_id
GROUP BY t.language
ORDER BY AVG(LENGTH(mtweet_text)) DESC
```

8. Total tweets about chat gpt on each day from the dataset

```
SELECT
DATE_FORMAT(timestamp, '%d-%m-%Y') 'Date',
COUNT(DATE_FORMAT(timestamp, '%d-%m-%Y')) '
of tweets' FROM tweet
GROUP BY DATE_FORMAT(timestamp, '%d-%m-%Y')
ORDER BY DATE_FORMAT(timestamp, '%d-%m-%Y')
```

9. The top 10 most mentioned twitter users in chat gpt related tweets.

```
SELECT
m.mentioned_user 'User', u.url 'Twitter URL',
COUNT(m.mentioned_user) 'Mentions' FROM
mentioned_users m
LEFT JOIN
user u ON m.mentioned_user = u.username GROUP BY
mentioned_user
ORDER BY mentions DESC
LIMIT 10;
```

10. Hashtag statistics - The minimum, maximum and average number of hashtags used in all the tweets from the dataset

```
SELECT
MIN(h.h_cnt) 'Min', MAX(h.h_cnt) 'Max', AVG(h.h_cnt)
'Avg' FROM (SELECT
COUNT(hashtag) h_cnt FROM tweet t LEFT JOIN hashtag
h USING (tweet_id) GROUP BY tweet_id) AS h
```

11. The top 5 most preferred sources of twitter to tweet during the time period from the dataset.

```
SELECT
Source, COUNT(source) 'Count' FROM tweet
GROUP BY source
ORDER BY COUNT(source) DESC
LIMIT 5;
```

12. The top 5 most quoted tweets about chatgpt, the user of the tweet and its text

```
SELECT
qt.qt_twt 'Quoted Tweet', qt.cnt 'Quoted Times', u.username
'User', m.tweet_text 'Tweet' FROM (SELECT
t.tweet_id AS id, t.user_id AS usr_id, m.quoted_tweet qt_twt,
t.permalink perma, COUNT(quoted_tweet) cnt FROM media
m
LEFT JOIN tweet t ON m.quoted_tweet = t.permalink
GROUP BY quoted_tweet
ORDER BY COUNT(quoted_tweet) DESC
LIMIT 10) AS qt LEFT JOIN
media m ON qt.id = m.tweet_id LEFT JOIN
user u ON qt.usr_id = u.user_id LIMIT 5;
```

13. Peak hours of activity

```
SELECT
timestamp, COUNT(tweet_id) AS Tweets FROM tweet
GROUP BY HOUR(timestamp)
ORDER BY COUNT(tweet_id) DESC
LIMIT 10;
```

B. Stored Procedures

1. Stored Procedure to display the stat for the day

```
Delimiter **
CREATE PROCEDURE Stat(IN Input_date date)
BEGIN
SELECT DATE(tweet.timestamp), COUNT(DISTINCT
(tweet.user_id)) AS 'Total Users for the day',
COUNT(DISTINCT (tweet.tweet_id)) AS 'Total tweets
for the day', COUNT(DISTINCT (hashtag.hashtag)) AS
'Total Hashtags for the day' FROM tweet
JOIN hashtag ON hashtag.tweet_id = tweet.tweet_id
WHERE DATE(tweet.timestamp) = Input_date
GROUP BY DATE(tweet.timestamp)
ORDER BY COUNT(DISTINCT (tweet.tweet_id)) DESC
, COUNT(DISTINCT (tweet.user_id)) , (hashtag.hashtag)
DESC;
END **
Delimiter ;
```

```
call stat('2023-01-22')
```

2. User statistics

```
Delimiter $$
```

```
Create Procedure user_stat( IN uname VARCHAR(20) )
BEGIN
  SELECT u.username, COUNT(t.tweet_id)
  AS 'Tweets', SUM(c.reply_count) AS 'Replies',
  SUM(c.retweet_count) AS 'Retweets', SUM(c.like_count)
  AS 'Likes', SUM(c.quote_count) AS 'Times Quoted' FROM
  user u
  JOIN tweet t USING (user_id)
  JOIN count c USING (tweet_id)
  WHERE u.username = uname;
END $$
Delimiter ;
```

```
call user_stat('GRDecter')
```

3. Stored Procedure To check whether a user was mentioned or not in a tweet:

```
DROP PROCEDURE IF EXISTS user_mentioned;
DELIMITER //
CREATE PROCEDURE user_mentioned(IN p_username
  VARCHAR(20))
BEGIN
  SELECT tweet_text as 'Message', timestamp as
  'Tweet posted_time', user_id as 'Userid', user_name as
  'User who mentioned' FROM ( SELECT media.tweet_text
  AS tweet_text, tweet.timestamp as timestamp, user.user_id as
  user_id, user.username as user_name FROM media
  JOIN tweet ON tweet.tweet_id = media.tweet_id
  JOIN mentioned_users ON mentioned_users.tweet_id =
  tweet.tweet_id
  JOIN user ON user.user_id = tweet.user_id WHERE
  mentioned_users.mentioned_user = p_username
  UNION ALL
  SELECT CONCAT(p_username, ' was not mentioned.') AS
  tweet_text, NULL as timestamp, NULL as user_id, NULL as
  user_name FROM dual
  WHERE NOT EXISTS(SELECT mentioned_user FROM
  mentioned_users WHERE mentioned_user = p_username) ) t;
END //
DELIMITER ;
```

```
CALL user_mentioned('sama');
CALL user_mentioned('gusthema');
```

4. Stored Procedure to retrieve most popular/liked tweet for a given language:

```
DELIMITER //
CREATE PROCEDURE most_pop_tweet(IN lang
  varchar(20))
BEGIN
  SELECT tweet.language as 'Language', media.tweet_text as
  'Most Liked Tweet', count.like_count as 'Likes' FROM tweet
  JOIN media USING (tweet_id)
  JOIN count USING (tweet_id)
```

```
WHERE tweet.language = lang
ORDER BY like_count DESC
LIMIT 1;
END //
DELIMITER ;
```

```
call most_pop_tweet('en');
```

C. Views

View that shows users who have at least used 3 devices, along with the number of devices used and the names of the devices

```
CREATE VIEW UsersWithMultipleDevices AS
SELECT
  tweet.user_id,
  user.username,
  COUNT(DISTINCT tweet.source) AS device_count,
  GROUP_CONCAT(DISTINCT tweet.source ORDER BY
  tweet.source SEPARATOR ', ') AS devices_used
FROM tweet
INNER JOIN user ON tweet.user_id = user.user_id
GROUP BY tweet.user_id
HAVING COUNT(DISTINCT tweet.source) >= 3
ORDER BY device_count DESC;
select * from UsersWithMultipleDevices
```

D. Triggers:

1. If any update happened to tweet, Updating like count in the count table if any incoming user likes the tweet,

```
DELIMITER $$
CREATE TRIGGER update_like_count
AFTER UPDATE ON tweet
FOR EACH ROW
BEGIN
  IF NEW.like_count != OLD.like_count THEN
    UPDATE count SET like_count = NEW.like_count WHERE
    tweet_id = OLD.tweet_id;
  END IF;
END $$
DELIMITER ;
(OR)
```

Can do it as a whole to update whole count table when a tweet is updated.

```
DELIMITER $$
CREATE TRIGGER UpdateCountOnTweetUpdate
AFTER UPDATE ON tweet
FOR EACH ROW
BEGIN
  UPDATE count SET
  reply_count = NEW.reply_count,
  retweet_count = NEW.retweet_count,
  like_count = NEW.like_count,
  quote_count = NEW.quote_count
  WHERE tweet_id = NEW.tweet_id;
END $$
```


DELIMITER ;

2.Delete Related Rows Trigger: This trigger deletes all related rows from remaining tables whenever a tweet is deleted.

```
DELIMITER $$
CREATE TRIGGER Delete_Related_Rows
AFTER DELETE ON tweet
FOR EACH ROW
BEGIN
DELETE FROM hashtag WHERE tweet_id = OLD.tweet_id;
DELETE FROM media WHERE tweet_id = OLD.tweet_id;
DELETE FROM mentioned_users WHERE tweet_id =
OLD.tweet_id;
DELETE FROM outlinks WHERE tweet_id = OLD.tweet_id;
DELETE FROM count WHERE tweet_id = OLD.tweet_id;
DELETE FROM user WHERE user_id = OLD.user_id
AND NOT EXISTS (SELECT 1 FROM tweet WHERE
user_id = OLD.user_id);
END $$
DELIMITER ;
```

3.Trigger using logs:

We can create a log table - for every update on a data, we capture the update on the logs table - this user had performed this

- Creating table log_details to capture and have the track of changes made to user Tweet Tables.

```
CREATE TABLE log_details (
Log_id INT AUTO_INCREMENT PRIMARY KEY,
table_name VARCHAR(255),
column_name VARCHAR(255),
old_value VARCHAR(255),
new_value VARCHAR(255),
changed_at TIMESTAMP DEFAULT CUR-
RENT_TIMESTAMP,
changed_by VARCHAR(255)
);
```

- Insert Trigger on USER table when we have a new row on the table.

```
DELIMITER $$
CREATE TRIGGER new_user_insert
AFTER INSERT ON user
FOR EACH ROW
BEGIN
INSERT INTO log_details (table_name, column_name,
old_value, new_value, changed_by)
VALUES ('user', 'user_id', NULL, NEW.user_id,
USER());
INSERT INTO log_details (table_name, column_name,
old_value, new_value, changed_by)
VALUES ('user', 'username', NULL, NEW.username,
USER());
END $$
DELIMITER ;
```

- Update Trigger on USER table to update a new value replacing the old value.

```
DELIMITER $$
CREATE TRIGGER update_user AFTER UPDATE ON
user
FOR EACH ROW
BEGIN
IF OLD.user_id != NEW.user_id THEN
INSERT INTO log_details (table_name, column_name,
old_value, new_value, changed_by)
VALUES ('user', 'user_id', OLD.user_id, NEW.user_id,
USER());
END IF;
IF OLD.username != NEW.username THEN
INSERT INTO log_details (table_name, column_name,
old_value, new_value, changed_by)
VALUES ('user', 'username', OLD.username,
NEW.username, USER());
END IF;
END $$
DELIMITER ;
```

- Log the new value in the log_details table when we have a new value to get insert in the tweet table row.

```
DELIMITER $$
CREATE TRIGGER new_tweet_insert
AFTER INSERT ON tweet
FOR EACH ROW
BEGIN
INSERT INTO log_details (table_name, column_name,
old_value, new_value, changed_by)
VALUES ('tweet', 'tweet_id', NULL, NEW.tweet_id,
USER());
INSERT INTO log_details (table_name, column_name,
old_value, new_value, changed_by)
VALUES ('tweet', 'timestamp', NULL, NEW.timestamp,
USER());
INSERT INTO log_details (table_name, column_name,
old_value, new_value, changed_by)
VALUES ('tweet', 'conv_id', NULL, NEW.conv_id,
USER());
INSERT INTO log_details (table_name, column_name,
old_value, new_value, changed_by)
VALUES ('tweet', 'user_id', NULL, NEW.user_id,
USER());
INSERT INTO log_details (table_name, column_name,
old_value, new_value, changed_by)
VALUES ('tweet', 'language', NULL, NEW.language,
USER());
INSERT INTO log_details (table_name, column_name,
old_value, new_value, changed_by)
VALUES ('tweet', 'source', NULL, NEW.source,
USER());
INSERT INTO log_details (table_name, column_name,
old_value, new_value, changed_by)
VALUES ('tweet', 'permalink', NULL, NEW.permalink,
```

```
USER());
END$$
DELIMITER ;
```

- Update a old value with new value and having a log about it in the log_details table.

```
DELIMITER $$
CREATE TRIGGER update_tweet
AFTER UPDATE ON tweet
FOR EACH ROW
BEGIN
IF OLD.tweet_id != NEW.tweet_id THEN
INSERT INTO log_details (table_name, column_name,
old_value, new_value, changed_by)
VALUES ('tweet', 'tweet_id',
OLD.tweet_id, NEW.tweet_id, USER());
END IF;
IF OLD.timestamp != NEW.timestamp THEN
INSERT INTO log_details (table_name, column_name,
old_value, new_value, changed_by)
VALUES ('tweet', 'timestamp',
OLD.timestamp, NEW.timestamp, USER());
END IF;
IF OLD.conv_id != NEW.conv_id THEN
INSERT INTO log_details (table_name, column_name,
old_value, new_value, changed_by)
VALUES ('tweet', 'conv_id',
OLD.conv_id, NEW.conv_id, USER());
END IF;
IF OLD.user_id != NEW.user_id THEN
INSERT INTO log_details (table_name, column_name,
old_value, new_value, changed_by)
VALUES ('tweet', 'user_id',
OLD.user_id, NEW.user_id, USER());
END IF;
IF OLD.language != NEW.language THEN
INSERT INTO log_details (table_name, column_name,
old_value, new_value, changed_by)
VALUES ('tweet', 'language',
OLD.language, NEW.language, USER());
END IF;
IF OLD.source != NEW.source THEN
INSERT INTO log_details (table_name, column_name,
old_value, new_value, changed_by)
VALUES ('tweet', 'source',
OLD.source, NEW.source, USER());
END IF;
IF OLD.permalink != NEW.permalink THEN
INSERT INTO log_details (table_name, column_name,
old_value, new_value, changed_by)
VALUES ('tweet', 'permalink',
OLD.permalink, NEW.permalink, USER());
END IF;
END$$
DELIMITER ;
```

- Updating user and tweet tables and checking the record with log_details table.

```
update user set username='karthik' where user_id=1;
update user set username='mochico123' where
user_id=1;
```

```
update tweet set source='some phone' where user_id=1;
update tweet set source='Twitter for iPhone' where
user_id=1;
```

E. Data Control Statements

1.We can create users

```
CREATE USER user@%
IDENTIFIED BY 'password';
```

2.List all users.

```
SELECT user FROM mysql.user
```

3.We grant restricted access to select particular table and view for specific user 'Prasad'

```
GRANT SELECT ON tweet_ver_1.tweet_view TO
'prasad'@'%';
GRANT SELECT ON tweet_ver_1.tweet TO 'prasad'@'%';
```

VI. AWS MIGRATION

Most organizations prefer saving their data on the cloud nowadays, as it offers higher scalability and storage capacity. We have thus migrated our MySQL application to Amazon Web Services for making it ready to use in the cloud environment. The figure below illustrates the database instance created on AWS that works using embedded sql.

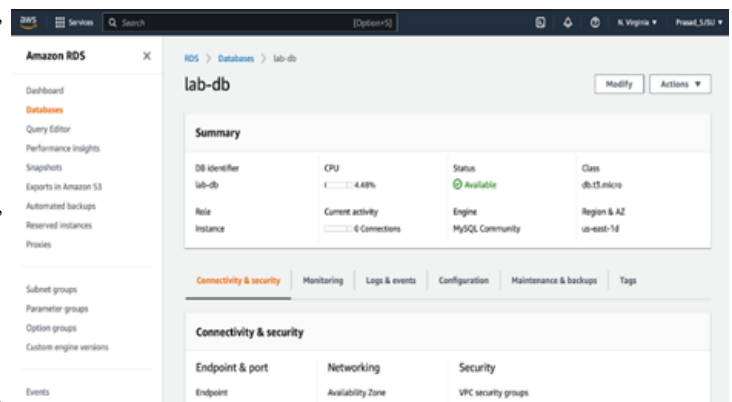


Fig. 4. AWS Instance DB Connection

VII. PERFORMANCE MEASUREMENT IN DB AND AWS

Different types of queries such as Data Definition Language(DDL),Data Manipulation Language(DML),Data Control Language(DCL),Stored Procedures,Triggers are used to retrieve the data.Out of all these queries DML commands will

was also made cloud ready by being migrated to AWS. Through this project we were thus able to demonstrate how to leverage a database management system, in combination with Python programming and cloud instantiating, for building an application relevant to a present day data analysis use case.

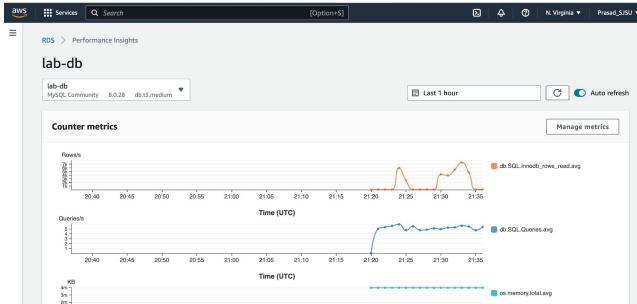


Fig. 5. Performance Metrics in AWS

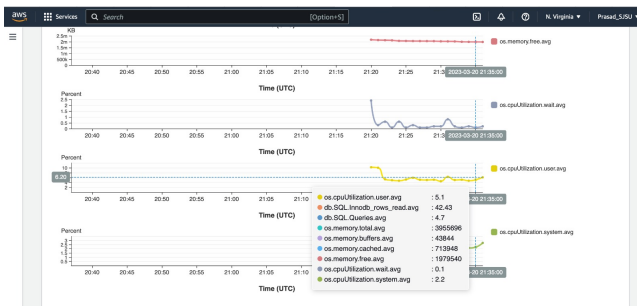


Fig. 6. Performance Metrics in AWS

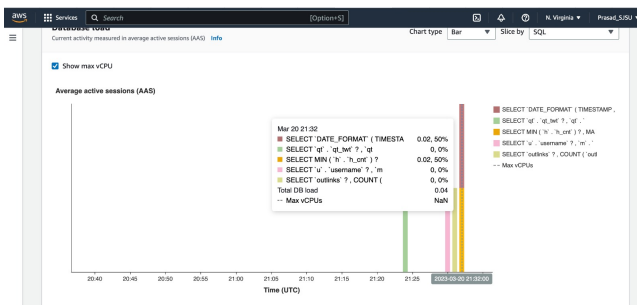


Fig. 7. Performance Metrics in AWS

VIII. CONCLUSION

We were able to successfully build and use our database application for analyzing the ChatGPT Twitter dataset. Using a relational database system required some data preprocessing to make it more structured for entering into the schema format. By analyzing this data through sql commands, we could generate insights into user statistics, popularity trends, and site usage patterns. Using a relational model allowed the application to have higher data consistency, which is appropriate for handling low volume of sensitive user data. The application