Problem Statement:

Recently, ChatGPT has been one of the most discussed topics. Twitter is a popular social media platform for people to connect and share their thoughts. Analyzing Twitter data provides deep insights into the way people think. By analyzing Twitter data on ChatGPT we can obtain valuable information. Hence, there is a need for a ChatGPT tweet analyzer application. A relational database provides many advantages like access, security, and integrity of the data compared to other earlier storage systems like files. Many users can concurrently access the database, the data is secured as access is required for data access. Any RDBMS promises data integrity with its ACID properties.

Functional Requirements:

Functional Requirements of our application from the given dataset:

1. Most active users from the dataset.
2. Most liked Tweet about ChatGPT.
3. Most used media in the Tweet.
4. Number of Hashtags used by user in the tweet.
5. Most discussed tweet with respect to replies on conversation ID
6. Most used language to express about ChatGPT.
7. Most retweeted tweet in each language.
8. Top Ten users who have more outlinks.
9. Conversations with most number of media shares.
10. Number of Tweets on each day on ChatGPT.
11. Get the most viral tweet with respect to retweet, quotes, likes.
12. Most used hashtag counts.
13. Most used source for tweeting.
14. Most mentioned user in the tweets.
15. Automated update of hashtag table after insertion to the tweet table.
16. Uniform tweet length maintenance restriction.
17. Auditing any deleted tweets .

**SPECIFICATIONS:**

My SQL Workbench 8.0.30

MySQL Libraries

Python Libraries: Pymysql, Sqlalchemy

AWS-RDS DB Instance

Diagram

Description automatically generated

SJSU ChatGPT Analyzer application has a two-tier architecture. The Analyzer app acts as a client to the AWS RDS MySQL Instance which acts as a server. AWS RDS provides a cloud-based database environment that caters to the data need of our application. The raw data which is in the form of a .csv file is extracted using a python program and then cleaned and transformed into the required format and then loaded into the ‘twitter’ database in AWS RDS MySQL Instance. The application runs on a command line interface and the user of our application can select options to view a few of the interesting metrics that can be obtained from the application. The developer can connect to AWS RDS MySQL Instance endpoints either through a MySQL client or through the python application.

ER DIAGRAM

Diagram

Description automatically generated

**TABLE STRUCTURE**

**/\*------- CREATING DATABASE -----------\*/**

create database twitter;

use twitter;

**/\*------- CREATING TABLES ----\*/**

**/\* Users Table \*/**

create table users (

username varchar(50) not null primary key,

User\_url varchar (50),

user\_info varchar(1000)

);

**/\* Media table \*/**

create table media(

media\_id bigint not null primary key,

media\_details text not null,

media\_type varchar(10)

);

**/\* Tweet Table \*/**

create table tweets

(

Tweetid bigint not null primary key,

username varchar(20) not null,

tweettext text not null,

createdtime datetime not null,

lang varchar(10) not null,

conversation\_id bigint not null,

media\_id bigint default null,

foreign key (username) references users(username) on delete cascade,

foreign key (media\_id) references media(media\_id)

);

**/\* Public Metrics Table \*/**

create table public\_metrics (

tweetid bigint not null primary key,

reply\_count int not null default 0,

retweet\_count int not null default 0,

like\_count int not null default 0,

quote\_count int not null default 0,

foreign key (tweetid) references tweets(tweetid) on delete cascade

);

**/\* Hash tag Table \*/**

create table hash\_tags

(

tweetid bigint not null,

hash\_tag varchar(1000) not null,

hastag\_count int,

foreign key (tweetid) references tweets(tweetid) on delete cascade

);

**/\* Url Table \*/**

create table tweet\_links

(

tweetid bigint not null,

permalink\_url varchar(100) not null,

source\_url varchar(200) not null,

count\_link text not null,

out\_link text not null,

foreign key (tweetid) references tweets(tweetid) on delete cascade

);

**/\* Quoted Tweet table \*/**

create table quoted\_tweets(

tweetid bigint not null,

quoted\_tweet varchar(100) not null,

**foreign key** (tweetid) references tweets(tweetid) on delete cascade

);

**/\* User Mentions Table \*/**

create table user\_mentions (

tweetid bigint not null,

username text not null,

**foreign key** (tweetid) references tweets(tweetid) on delete cascade

);

**Usage of triggers and Procedures:**

**Trigger:**

A Database trigger is automated execution of the procedural code in response to an event on a particular table or view in database.Triggers are mainly used for maintaining the integrity of the information for a database. For example, whenever the new record(from existing user) is inserted into the tweet table the trigger should automatically fire and the hash tag table should get updated and triggers can also be used for auditing.

**Stored Procedure**

A stored procedure is similar to a function in python or C.It is a subroutine used to access a RDS.

These stored procedure is sequence of SQL statements whichis stored in server and used whenever required by, calling the procedures.

**RESULTS**

1..

**Query:**

/\* Active Users tweeted in given time in descending order \*/

SELECT username as active\_users,COUNT(tweetid) as tweet\_count

FROM tweets

GROUP BY username

ORDER BY COUNT(tweetid) DESC limit 10;

Graphical user interface, text, application, email

Description automatically generated

2.

Query:

/\* Most Liked Tweet about ChatGPT \*/

select tweetid,username,tweettext,createdtime,like\_count from tweets join public\_metrics using (tweetid)

order by like\_count desc;

Graphical user interface, text, application, email

Description automatically generated

3.

**Query**

/\* Most Used Media Type \*/

select media\_type, count(\*)as media\_used from media

group by media\_type

order by media\_used desc;

Graphical user interface, text, application

Description automatically generated

4

**Query**

/\*User who have used more Hashtags \*/

select username,tweetid,hash\_tag,hastag\_count from tweets join hash\_tags using (tweetid) order by hastag\_count desc;

Graphical user interface, text, application, email

Description automatically generated

5.

**Query**

/\* Most Discussed tweet with respect to replies based on conversation id in the given data \*/

SELECT tweetid,username,conversation\_id,COUNT(\*) as reply\_to\_tweet

FROM tweets

GROUP BY conversation\_id

ORDER BY reply\_to\_tweet DESC limit 15;

Graphical user interface, text, application, email

Description automatically generated

6.

Query

/\* Most used Language to express about ChatGPT \*/

select lang,count(\*)as total\_count from tweets group by lang order by total\_count desc limit 10;Graphical user interface, text, application

Description automatically generated

7.

-- Most retweeted tweet in each language

with langquery as (

select tweetid, `lang`, username, tweettext, retweet\_count,

row\_number() over (partition by `lang` order by retweet\_count desc) as rownum

from tweets join public\_metrics using (tweetid))

Select \* from langquery

where rownum =1

order by retweet\_count desc;Graphical user interface, text, application, email

Description automatically generated

8.

-- top 10 Users who have more outlinks

select username, count(out\_link) as `Number of links`

from tweets join tweet\_links using (tweetid)

group by username

order by count(out\_link) desc

limit 10;

Graphical user interface, text, application, email

Description automatically generated

9.

-- conversations with most number of media shares

select conversation\_id, media\_type, count(media\_id) as `number of media shared`

from tweets join media using (media\_id)

group by conversation\_id, media\_type

order by count(media\_id) desc

limit 10;

Graphical user interface, text, application, email

Description automatically generated

10.

-- Number of tweets each day on chatGPT

select date(createdtime) as Tweeted\_date, count(tweetid) as tweet\_count

from tweets

group by Tweeted\_date;

Graphical user interface, text, application, email

Description automatically generated

**Stored\_procedure**

1.viral tweet

**Query**

DROP PROCEDURE get\_most\_viral\_tweet;

DELIMITER //

CREATE PROCEDURE get\_most\_viral\_tweet()

BEGIN

SELECT t.username,t.tweetid,t.tweettext, t.createdtime,

COALESCE(pm.retweet\_count, 0) + COALESCE(pm.quote\_count, 0)+COALESCE(pm.like\_count, 0) AS total\_count

FROM tweets t

LEFT JOIN public\_metrics pm ON t.tweetid = pm.tweetid

ORDER BY total\_count DESC

LIMIT 15;

END; //

CALL get\_most\_viral\_tweet();

Graphical user interface, text, application, email

Description automatically generated

2.most used hashtag

Query

/\*-----------Most used Hashtags------------\*/

DROP PROCEDURE IF EXISTS GET\_HASHTAG\_COUNT;

DELIMITER //

CREATE PROCEDURE GET\_HASHTAG\_COUNT()

BEGIN

DECLARE val TEXT DEFAULT NULL;

DECLARE firstVal TEXT DEFAULT NULL;

DECLARE firstValLen INT DEFAULT NULL;

DECLARE tempVal TEXT DEFAULT NULL;

DECLARE finished INT DEFAULT 0;

DECLARE CUR CURSOR FOR SELECT HASH\_TAG FROM hash\_tags;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;

DROP TABLE IF EXISTS HASHTAG\_SPLIT\_TABLE;

CREATE TABLE HASHTAG\_SPLIT\_TABLE(HASHTAG TEXT);

OPEN CUR;

get\_hashtag: LOOP

FETCH CUR INTO val;

IF finished = 1 THEN

LEAVE get\_hashtag;

END IF;

iterator: LOOP

IF LENGTH(val) = 0 OR val IN ('[]') OR val IS NULL THEN

LEAVE iterator;

END IF;

IF val NOT LIKE '[%]' THEN

SET val = trim(trailing '\"' from trim(leading '\"' from val));

END IF;

IF val LIKE '[%]' THEN

SET val = trim(trailing ']' from trim(leading '[' from val));

END IF;

SET firstVal = SUBSTRING\_INDEX(VAL,',',1);

SET firstValLen = LENGTH(firstVal);

SET tempVal = trim(trailing '\'' from trim(leading '\'' from TRIM(firstVal)));

SET tempVal = trim(trailing '\'' from trim(leading '\'' from TRIM(tempVal)));

IF LENGTH(tempVal) = 0 OR tempVal IS NULL THEN

LEAVE iterator;

END IF;

INSERT INTO HASHTAG\_SPLIT\_TABLE (HASHTAG) VALUES (tempVal);

SET val = INSERT(val,1,firstValLen + 1,'');

END LOOP;

END LOOP;

CLOSE CUR;

SELECT HASHTAG, COUNT(HASHTAG)as used\_count FROM HASHTAG\_SPLIT\_TABLE GROUP BY HASHTAG ORDER BY COUNT(HASHTAG) DESC LIMIT 10;

END //

DELIMITER ;

CALL GET\_HASHTAG\_COUNT();

Graphical user interface, text, application

Description automatically generated

3.

Query

/\*------------- Most used Source -----\*/

drop procedure if exists most\_used\_source;

DELIMITER //

CREATE PROCEDURE most\_used\_source()

BEGIN

SELECT SUBSTRING\_INDEX(SUBSTRING\_INDEX(source\_url, '>', -2), '<', 1) AS used\_source,

COUNT(\*) AS most\_used\_source

FROM tweet\_links

WHERE

source\_url LIKE '%twitter.com%'

GROUP BY used\_source

ORDER BY most\_used\_source DESC

LIMIT 10;

end //

call most\_used\_source()

Graphical user interface, text, application

Description automatically generated

4.

Query

/\* Most Mentioned Use \*/

drop procedure if exists most\_mentioned\_user ;

DELIMITER //

CREATE PROCEDURE most\_mentioned\_user()

BEGIN

SELECT mentioned\_user, COUNT(\*) AS mention\_count

FROM (SELECT regexp\_SUBSTR(tweettext, '@[[:alnum:]\_]+') AS mentioned\_user

FROM tweets) AS t

WHERE mentioned\_user != ''

GROUP BY mentioned\_user

ORDER BY mention\_count DESC limit 20;

end //

call most\_mentioned\_user()

Graphical user interface, text, application, Teams

Description automatically generated

**Triggers**

1.most viral tweet

Pending

**1**

**2.**

**Query**

/\*-------Trigger to not update if tweet text length is > 280.......\*/

drop trigger if exists trigger\_tweet\_length;

DELIMITER //

CREATE TRIGGER trigger\_tweet\_length

BEFORE insert ON tweets

FOR EACH ROW

BEGIN

IF LENGTH(NEW.tweettext) > 280 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Tweet text cannot be longer than 280 characters';

END IF;

END;//

insert into tweets (tweetid,tweettext) values (161050057205020000,'aaaaaaaaaaaaaaaaaaaaaaaaa

aaaaaajjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

jjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaajjjjjjjj

jjjjjjjjjjjjjjjjjjjjjjjjjjaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaajjjjjjjjjjjjjjjj

jjjjjjjjjjjjjjjjjjaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaajjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjaaaaaaaaaa

aaaaaaaaaaaaaaaaaaaaajjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjj');

A picture containing text

Description automatically generated

**3.**

CREATE TABLE log (

delete\_timestamp datetime primary key,

db\_user varchar(50),

tweet\_id bigint not null,

username varchar(50) not null

);

DELIMITER //

CREATE TRIGGER log\_before\_delete

Before delete ON tweets

FOR EACH ROW

BEGIN

INSERT INTO log values (now(),user(),old.tweetid,old.username);

END;//

delete from tweets where username='priyanka';

Log Table Before data was Deleted

Graphical user interface, text, application

Description automatically generated

Log Table After data was Deleted

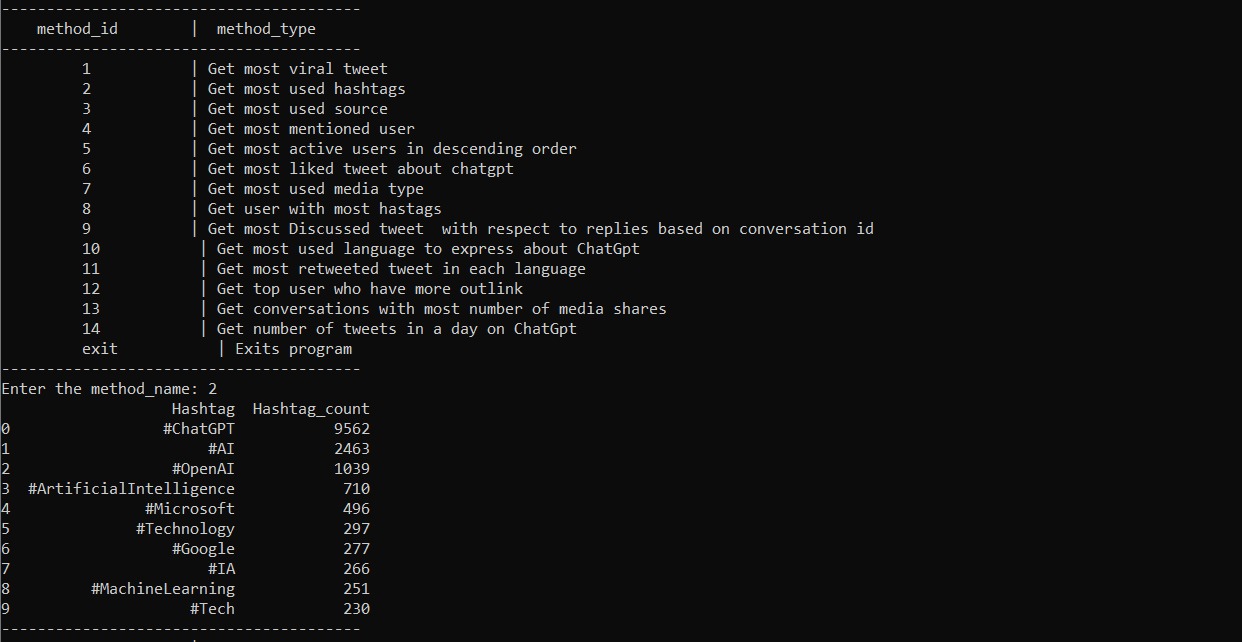
**USER INTERFACE**

User can select pre defined sql queries to obtain the required results in the application.

**User Interface**

Text

Description automatically generated

**Query giving most used hashtags in user interface**

**Most used source result in user interface**

Text

Description automatically generated