1. Return the handles of Tweeters and their number of Covid-tagged tweets if they’ve used the hashtag “**covid19**” more than 3 times. Your query should normalize the hashtags to lowercase (e.g., **Covid19** should be converted to **covid19** in order to properly consider all Covid-tagged tweets.

a) SQL Query:

Graphical user interface, text, application

Description automatically generated

b) Result (2 rows):

Graphical user interface, application

Description automatically generated

2. Return the handles of Tweeters who have a followers count greater than 500,000 and who have posted a tweet that contains one or more of the top ten most popular hashtags. (Note: You can break popularity ties arbitrarily.)

a) SQL Query:

Graphical user interface, text, application

Description automatically generated

b) Result (3 rows):

Graphical user interface, text, application, email

Description automatically generated

3. Find the tweet ids for tweets that have been **verified** using at least two **different** pieces of evidence and that have a popularity greater than 2.4. Remember from HW1 (😷  ) that the popularity of a tweet can be computed using the formula:

Popularity=0.4 (Number of quotes)+0.6 (Number of replies)

a) SQL Query:

Graphical user interface, text, application, email

Description automatically generated

b) Result (4 rows):

Graphical user interface, text, application

Description automatically generated

4. **Views**

Congratulations! For obvious reasons, the CTO of ***CheckedTweets.org*** is setting up a data science team to analyze election tweets that contain one or more of the following hashtags: **"election2020", "trump", "biden", "bidenharris2020", "trumppence2020", "pennsylvania", "northcarolina", "wisconsin", "michigan".**  (*You will need to normalize the hashtags to lowercase*.)  The CTO has made you the head of that team. As the team leader, you have been asked to create a SQL view so that the rest of the team can simply look at the data and draw meaningful conclusions without having to deal with all of its underlying complexity.

The view should provide simple tabular access to a combination of the following pieces of information:

* Tweeter info (tweeter\_id, handle, followers\_count, verified)
* Tweet info (tweet\_id, tweet\_text, *popularity*, *quality*)

**Remember** that tweet *popularity* and *quality* are derived attributes and can be computed as follows:

Popularity=0.4 (Number of quotes)+0.6 (Number of replies)

Quality=Amount of associated evidence used for verification

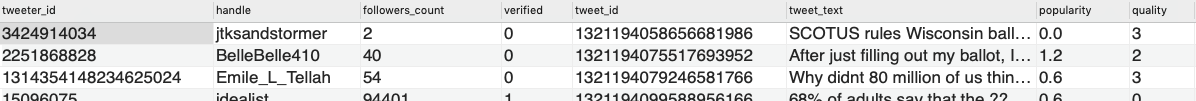
a) Create the desired view (ElectionTweets) by writing an appropriate CREATE VIEW statement.

CREATE VIEW ElectionTweets…;

Graphical user interface, text, application

Description automatically generated

*(Hint: your view should have 699 rows)*



b) [5 pts] Show the usefulness of your view by writing a SELECT query against the view that prints the Tweet id, the Tweeter’s handle, and the popularity and quality of tweets that have the maximum popularity.

Text

Description automatically generated with medium confidence

Result (1 row):

Table

Description automatically generated

5. **Stored Procedures**

a) Create and exercise a SQL stored procedure called RegisterChecker(…) that the application can use to add a brand new checker with an office phone to the database. You **may not** change the signature of this procedure. Hint: To get the current time, use the [NOW()](https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html#function_now) function.

|  |
| --- |
| DELIMITER // **CREATE** **PROCEDURE** RegisterChecker(     user\_id integer,     name\_first varchar(50),     name\_last varchar(50),      email varchar(100),     password varchar(30),     profile\_pic varchar(500),     address\_country varchar(30),     address\_state varchar(30),     address\_city varchar(30),     office\_number varchar(20) ) **BEGIN**      ... **END**;  // DELIMITER ; |

Graphical user interface, text, application, email

Description automatically generated

b) Verify that your new stored procedure works properly by calling it as follows to add a new checker and then running a SELECT query to show the stored procedure’s after-effects:

|  |
| --- |
| **CALL** RegisterChecker (3000, "Peter", "Anteater",      "peter-anteater2020@gmail.com", "pretend-this-is-hashed",     null, "USA", "California", "Irvine", "(949) 824-5011");  **SELECT** U.user\_id, U.email, U.profile\_pic, C.checker\_since, P.number, P.kind **FROM** **User** U, Checker C, Phone P **WHERE** U.user\_id = C.user\_id **AND**        P.user\_id = C.user\_id **AND**  U.user\_id = 3000; |

Result (1 row):

Graphical user interface, application, table

Description automatically generated

6. **Alter Table**

As your schema currently stands, evidence can only be submitted in the form of URLs to websites. Your boss would like to enrich the Evidence entity by also allowing books (specifically, 13-character ISBNs) to be used as evidence. This changes your ER model in two ways: 1) URL now becomes an optional field in Evidence, and 2) ISBN is now an additional optional field in Evidence.

Note: *The current datatype for URL is VARCHAR(500).*

a) Write and execute the ALTER TABLE statement(s) needed to modify the Evidence table to reflect the new requirements above. (Hint: Refer to the MySQL documentation online if you need more information about how to use the ALTER TABLE statement.)

Graphical user interface

Description automatically generated with low confidence

b) Execute the following INSERT and SELECT statements to show the effect of your change. Report the results (just the counts) for each SELECT statement.

|  |
| --- |
| **INSERT** **INTO** Evidence (ev\_id, url, isbn) **VALUES** (2000, NULL, "0-1306-3278-3");  **SELECT** **COUNT**(\*) **AS** url\_evidence  **FROM** Evidence **WHERE** url **IS NOT NULL**;  **SELECT COUNT**(\*) **AS** book\_evidence  **FROM** Evidence  **WHERE** isbn **IS NOT NULL**; |

Result:

1706 URLs, 1 book

7. **Triggers**

To help tie your newfound SQL knowledge back to the seemingly mysterious initial ER model, you are tasked with defining a trigger called update\_tweet\_info(…). When raw tweets are deposited into the database by your application, this trigger will insert tuples into the Tweet, Tweeter, and Hashtags tables using the information found in the newly deposited raw tweets. *If a Tweeter already exists at the time of a deposit, you should only update their follower count, display name, and handle.* To specify an update action for an INSERT statement when you have a duplicate primary key, i.e., when an object with that key already exists, see [here](https://dev.mysql.com/doc/refman/8.0/en/insert-on-duplicate.html).

Hint 1: To get all tweeter-associated information for an arbitrary single raw tweet, we can perform the query:

|  |
| --- |
| **SELECT** JSON\_UNQUOTE(JSON\_EXTRACT(**content**, '$.user.screen\_name')) **AS** display\_name,     JSON\_UNQUOTE(JSON\_EXTRACT(**content**, '$.user.followers\_count')) **AS** followers\_count,     JSON\_UNQUOTE(JSON\_EXTRACT(**content**, '$.user.name')) **AS** handle,     JSON\_UNQUOTE(JSON\_EXTRACT(**content**, '$.user.id\_str')) **AS** tweeter\_id,     **CASE** **WHEN** JSON\_EXTRACT(**content**, '$.user.verified') **THEN** 1 **ELSE** 0 **END** **AS** verified **FROM** RawTweet T **LIMIT** 1; |

Hint 2: To get all tweet-associated information for an arbitrary single raw tweet, we can perform the query:

|  |
| --- |
| **SELECT** JSON\_UNQUOTE(JSON\_EXTRACT(T.content, '$.created\_at')) **AS** posting\_datetime,      JSON\_EXTRACT(T.content, '$.geo.coordinates[0]') **AS** posting\_location\_latitude,     JSON\_EXTRACT(T.content, '$.geo.coordinates[1]') **AS** posting\_location\_longitude,      JSON\_EXTRACT(T.content, '$.quoted\_status\_id') **AS** quoted\_tweet,      JSON\_EXTRACT(T.content, '$.in\_reply\_to\_status\_id') **AS** replied\_to\_tweet,      JSON\_UNQUOTE(JSON\_EXTRACT(T.content, '$.id')) **AS** tweet\_id,     JSON\_UNQUOTE(JSON\_EXTRACT(T.content, '$.text')) **AS** tweet\_text,     JSON\_UNQUOTE(JSON\_EXTRACT(T.content, '$.user.id\_str')) **AS** tweeter\_id **FROM** RawTweet T **LIMIT** 1; |

Hint 3: To **update** the Hashtag table for a particular raw tweet, we can call the following stored procedure that we have provided for you in the updated load script:

|  |
| --- |
| **CALL** UpdateHashtags(tweet\_id); |

|  |
| --- |
| DELIMITER // **CREATE** **TRIGGER** update\_tweet\_info ... **FOR** **EACH** **ROW** **BEGIN** ... **END**; // DELIMITER ; |

Graphical user interface, application

Description automatically generated