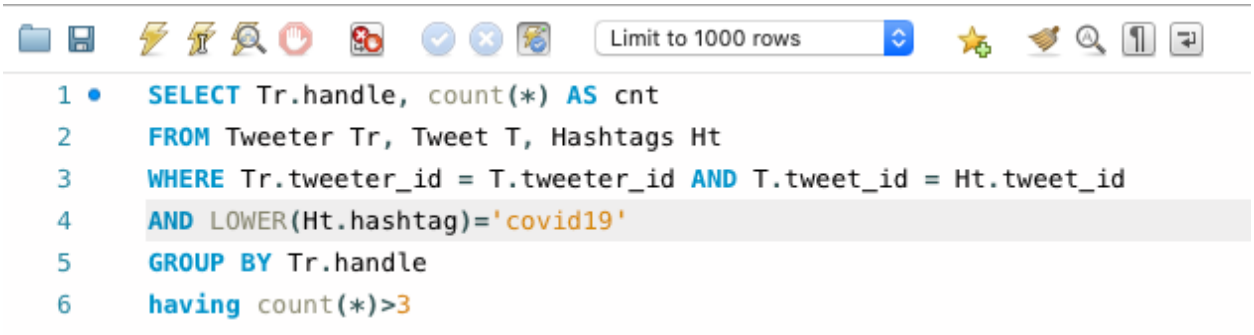
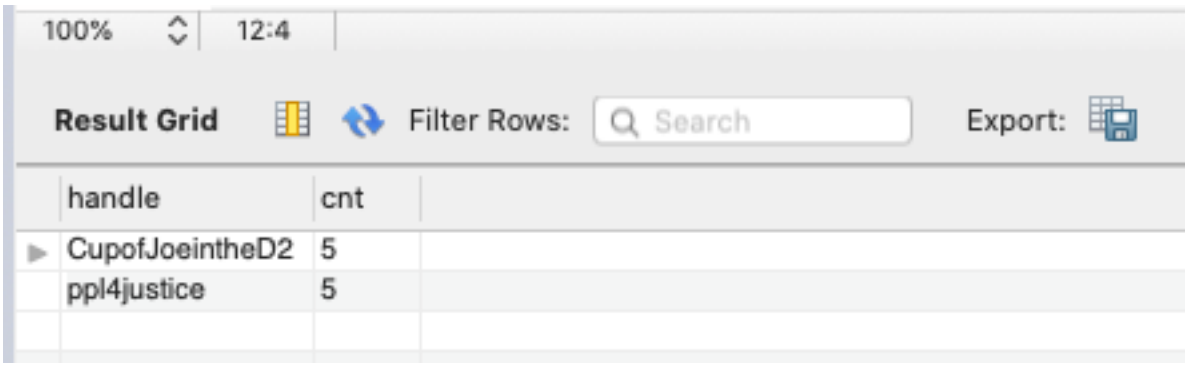
1. Return the handles of Tweeters and their number of Covid-tagged tweets if they've used the hashtag "**covid19**" more than 3 times. Your query should normalize the hashtags to lowercase (e.g., **Covid19** should be converted to **covid19** in order to properly consider all Covid-tagged tweets.

a) SQL Query:



b) Result (2 rows):

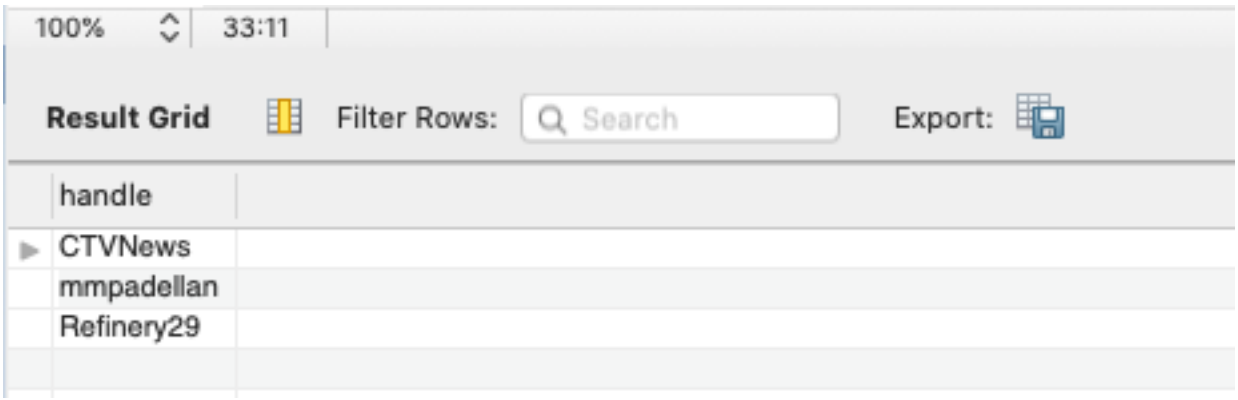


2. Return the handles of Tweeters who have a followers count greater than 500,000 and who have posted a tweet that contains one or more of the top ten most popular hashtags. (Note: You can break popularity ties arbitrarily.)

a) SQL Query:

```
1  WITH TrendingTags AS (SELECT Ht.hashtag
2                        FROM Hashtags Ht
3                        GROUP BY Ht.hashtag
4                        ORDER BY count(*) DESC
5                        LIMIT 10)
6       SELECT DISTINCT Tr.handle
7       FROM TrendingTags TT, Tweeter Tr, Hashtags Ht, Tweet T
8       WHERE Tr.followers_count > 500000
9       AND T.tweeter_id= Tr.tweeter_id
10      AND T.tweet_id=Ht.tweet_id
11      AND Ht.hashtag=TT.hashtag;
```

b) Result (3 rows):



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:

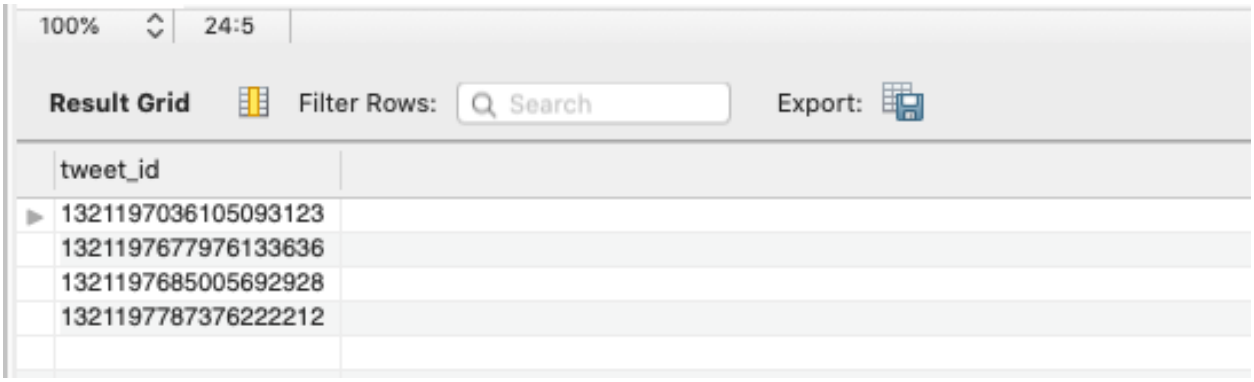
Popularity=0.4 (Number of quotes)+0.6 (Number of replies)

a) SQL Query:

```sql
WITH VerifiedTweets AS (SELECT T.tweet_id
FROM Tweet T, Verification V, VerifiedUsing VU
WHERE T.tweet_id=V.tweet_id AND VU.ver_id=V.ver_id
GROUP BY T.tweet_id
Having count(*) >= 2),
PopularTweets AS (SELECT T2.tweet_id, (
SELECT count(*) FROM Tweet T4 WHERE T4.quoted_tweet=T2.tweet_id) AS qt_cnt,
 (SELECT count(*) FROM Tweet T3 WHERE T3.replied_to_tweet=T2.tweet_id) AS rep_cnt
FROM Tweet T2
WHERE  0.4 * (SELECT count(*) FROM Tweet T4 WHERE T4.quoted_tweet=T2.tweet_id) +
 0.6 * (SELECT count(*) FROM Tweet T3 WHERE T3.replied_to_tweet=T2.tweet_id) > 2.4
ORDER BY 0.4 * qt_cnt+ 0.6 * rep_cnt)

SELECT VT.tweet_id FROM VerifiedTweets VT , PopularTweets PT
WHERE VT.tweet_id = PT.tweet_id
```

b) Result (4 rows):



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…;

```
CREATE VIEW ElectionTweets(tweeter_id, handle, followers_count, verified, tweet_id, tweet_text, popularity, quality) AS
SELECT DISTINCT Tr.tweeter_id, Tr.handle, Tr.followers_count, Tr.verified, T.tweet_id, T.tweet_text,
 (0.4 * (SELECT count(*) FROM Tweet T4 WHERE T4.quoted_tweet=T.tweet_id)
 + 0.6 * (SELECT count(*) FROM Tweet T3 WHERE T3.replied_to_tweet=T.tweet_id)),
 (SELECT count(*) FROM Verification V, VerifiedUsing VU WHERE V.ver_id=VU.ver_id AND T.tweet_id=V.tweet_id)
FROM Tweet T, Tweeter Tr, Hashtags Ht
WHERE T.tweeter_id=Tr.tweeter_id AND Ht.tweet_id = T.tweet_id
AND LOWER(Ht.hashtag) IN ('election2020','trump','biden','bidenharris2020','trumppence2020','pennsylvania','northcarolina','wisconsin','michigan');
SELECT count(*) FROM ElectionTweets;
```

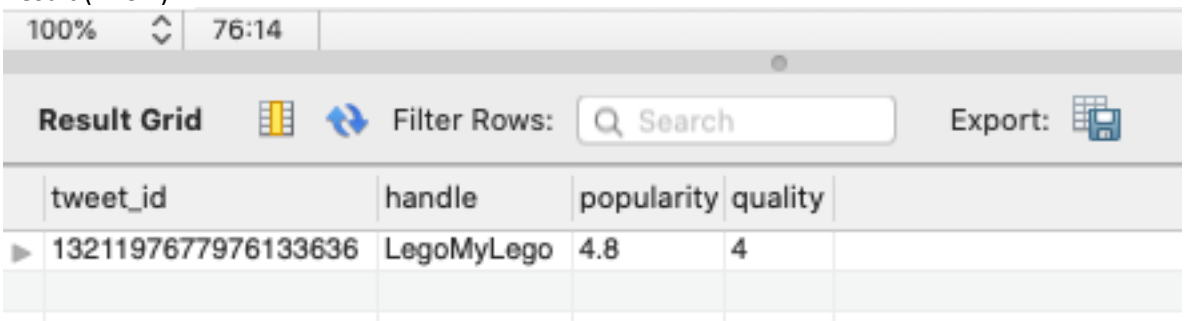
(Hint: your view should have 699 rows)

tweeter_id	handle	followers_count	verified	tweet_id	tweet_text	popularity	quality
3424914034	jtksandstormer	2	0	1321194058656681986	SCOTUS rules Wisconsin ball…	0.0	3
2251868828	BelleBelle410	40	0	1321194075517693952	After just filling out my ballot, I…	1.2	2
1314354148234625024	Emile_L_Tellah	54	0	1321194079246581766	Why didnt 80 million of us thin…	0.6	3
15096075	idealist	94401	1	1321194099588956166	68% of adults say that the 22	0.6	0

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.

```
SELECT ET.tweet_id, ET.handle, ET.popularity, ET.quality
FROM ElectionTweets ET
WHERE ET.popularity = (SELECT Max(ET2.popularity) FROM ElectionTweets ET2);
```

Result (1 row):



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() 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 ;
```

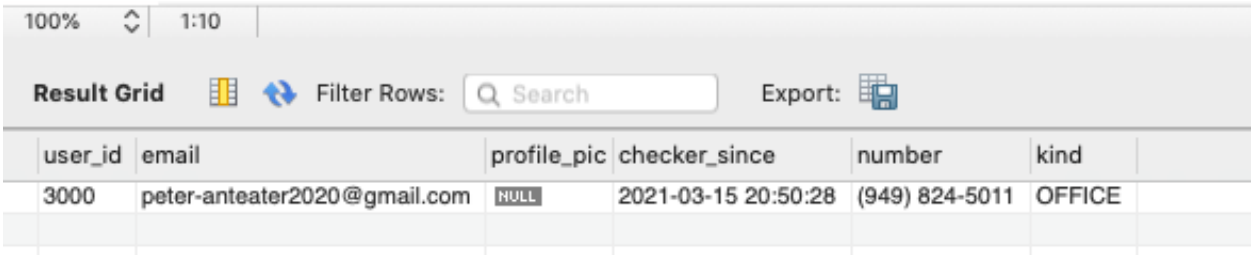
```
use cs122a_fall20;
DELIMITER //
CREATE PROCEDURE RegisterChecker(
    user_id integer,
    name_first varchar(50),
    name_last varchar(50),
    email varchar(100),
    password varchar(30),
    profile_pic varchar(500),
    address_country varchar(30),
    address_state varchar(30),
    address_city varchar(30),
    office_number varchar(20)
)
BEGIN
    SET @user_since = NOW();
    INSERT INTO User(user_id, name_first, name_last, email, password, user_since, profile_pic, address_country, address_state, address_City)
    VALUES(user_id, name_first, name_last, email, password, @user_since, profile_pic, address_country, address_state, address_city);
    INSERT INTO Checker(user_id, checker_since)
    VALUES(user_id, @user_since);
    INSERT INTO Phone(user_id, kind, number)
    VALUES(user_id, 'Office', office_number);
END;  //
DELIMITER ;
```

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):



user_id	email	profile_pic	checker_since	number	kind
3000	peter-anteater2020@gmail.com	NULL	2021-03-15 20:50:28	(949) 824-5011	OFFICE

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.)

```
ALTER TABLE Evidence MODIFY url varchar(500),
                     ADD isbn varchar(13);
```

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.

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 ;
```

```
DELIMITER $$

CREATE TRIGGER update_tweet_info
AFTER INSERT ON RawTweet FOR EACH ROW
BEGIN
    INSERT INTO Tweeter(display_name, followers_count, handle, tweeter_id, verified)
    VALUES(
    JSON_UNQUOTE(JSON_EXTRACT(NEW.content, '$.user.screen_name')),
    JSON_UNQUOTE(JSON_EXTRACT(NEW.content, '$.user.followers_count')),
    JSON_UNQUOTE(JSON_EXTRACT(NEW.content, '$.user.name')),
    JSON_UNQUOTE(JSON_EXTRACT(NEW.content, '$.user.id_str')),
    CASE WHEN JSON_EXTRACT(NEW.content, '$.user.verified') THEN 1 ELSE 0 END)
    ON DUPLICATE KEY UPDATE
    display_name=JSON_UNQUOTE(JSON_EXTRACT(NEW.content, '$.user.screen_name')),
    followers_count=JSON_UNQUOTE(JSON_EXTRACT(NEW.content, '$.user.followers_count')),
    handle=JSON_UNQUOTE(JSON_EXTRACT(NEW.content, '$.user.name'));
    INSERT INTO Tweet(posting_datetime, posting_location_longitude, posting_location_latitude, quoted_tweet, replied_to_tweet, tweet_id, tweet_text, tweeter_id)
    VALUES(
    JSON_UNQUOTE(JSON_EXTRACT(NEW.content, '$.created_at')),
    JSON_EXTRACT(NEW.content, '$.geo.coordinates[0]'),
    JSON_EXTRACT(NEW.content, '$.geo.coordinates[1]'),
    JSON_EXTRACT(NEW.content, '$.quoted_status_id'),
    JSON_EXTRACT(NEW.content, '$.in_reply_to_status_id'),
    JSON_UNQUOTE(JSON_EXTRACT(NEW.content, '$.id')),
    JSON_UNQUOTE(JSON_EXTRACT(NEW.content, '$.text')),
    JSON_UNQUOTE(JSON_EXTRACT(NEW.content, '$.user.id_str')));



    CALL UpdateHashtags(tweet_id);
END;
$$

DELIMITER ;
```