DSC 450: Database Processing for Large-Scale Analytics
Assignment Module 9

Part 1

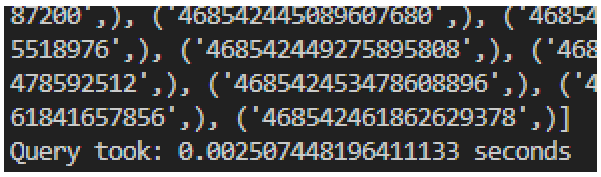
Using the same tweets from: https://dbgroup.cdm.depaul.edu/DSC450/Module7.txt

Create a third table, now incorporating the Geo table (in addition to tweet and user tables that you already have) and extend your schema accordingly. You do not need to use ALTER TABLE, it is sufficient to just re-create (drop and create) your schema.
You will need to generate an ID for the Geo table primary key (you may use any value or reasonable combination of values as long as it is unique) for that table and link it to the Tweet table (foreign key should be in the Tweet). However, that value should be related to the location (i.e., same value for same location) – do not use a simple incremental key or a random key.
In addition to the primary key column, the geo table should have the “type”, “longitude” and “latitude” columns. **NOTE**: that there are no longitude/latitude dictionary keys. Instead, the geo dictionary has a 2-value tuple which are the lon/lat coordinates.
Load the tweet data into your new tables.

a. Write and execute a SQL query to do the following. Time and report the runtime of your query.

   Find tweets where tweet id_str contains “89” or “78” anywhere in the column

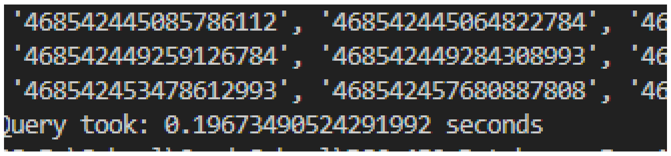
```
    #1a
    start = time.time()
    query = cursor.execute("SELECT TweetId FROM Tweets WHERE TweetId LIKE '%89%'
OR TweetId LIKE '%78%';").fetchall()
    end = time.time()
    print(query)
    print('Query took: ' + str(end - start) + ' seconds')
```



```
87200',), ('468542445089607680',), ('46854
5518976',), ('468542449275895808',), ('468
478592512',), ('468542453478608896',), ('4
61841657856',), ('468542461862629378',)]
Query took: 0.002507448196411133 seconds
```

b. Write the equivalent of the previous query in python (without using SQL) by reading it from the file. Time and report the runtime of your query.

```
#1b
start = time.time()
res = []
for tweet in tweets:
    try:
        tmp = json.loads(tweet.decode('utf8'))
        tweetId = tmp['id_str']
        if '89' in tweetId or '78' in tweetId:
            res.append(tweetId)
    except:
        fd.write("Error: " + str(tweet) + "\n")
print(res)
end = time.time()
print('Query took: ' + str(end - start) + ' seconds')
```

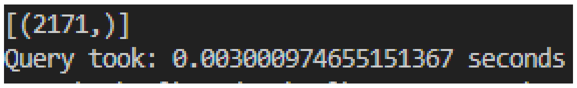


```
 '468542445085786112', '468542445064822784', '46
 '468542449259126784', '468542449284308993', '46
 '468542453478612993', '468542457680887808', '46
Query took: 0.19673490524291992 seconds
```

c. Write and execute a SQL query to do the following. Time and report the runtime of your query.

Find how many unique values are there in the "friends_count" column

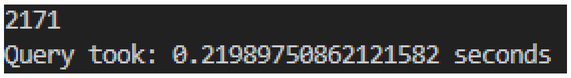
```
#1c
    start = time.time()
    query = cursor.execute("SELECT COUNT(DISTINCT FriendsCount) AS unique_fc FROM 
Users;").fetchall()
    end = time.time()
    print(query)
    print('Query took: ' + str(end - start) + ' seconds')
```



```
[(2171,)]
Query took: 0.003000974655151367 seconds
```

d. Write the equivalent of the previous query in python (without using SQL) by reading it from the file. Time and report the runtime of your query.

```
#1d
  start = time.time()
  ufc = []
  for tweet in tweets:
    try:
      tmp = json.loads(tweet.decode('utf8'))
      tweetId = tmp['id_str']
      curr = tmp['user']['friends_count']
      if curr not in ufc:
        ufc.append(curr)
    except:
      fd.write("Error: " + str(tweet) + "\n")
  end = time.time()
  print(len(ufc))
  print('Query took: ' + str(end - start) + ' seconds')
```



```
2171
Query took: 0.21989750862121582 seconds
```

e. Use python to plot the lengths of first 60 tweets (only 60, not all of the tweets) versus the length of the username for the user on a graph. Create a scatterplot. Submit both your python code and the resulting graph file.

Matplotlib broken ☹

Part 2

a. Create an index on userid in Tweet table in SQLite (submit SQL code for this question). These questions are as straightforward as they appear, you just need to create an index.

```
CREATE INDEX UserIdIndex
        ON Tweets(UserId);
```

b. Create a composite index on (friends_count, screen_name) in User table (submit SQL code for this question)

```
CREATE INDEX FcSnIndex
        ON Users(FriendsCount, ScreenName);
```

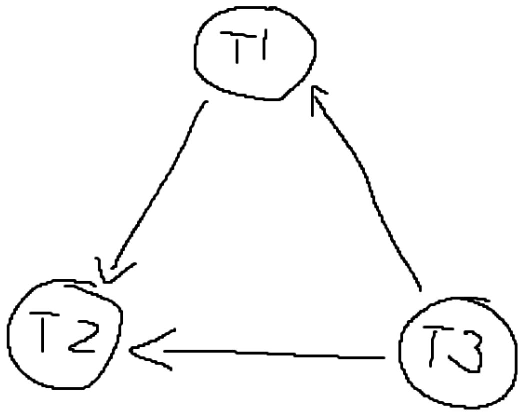
c. Create a materialized view (using CREATE TABLE AS because SQLite does not have full support for MVs) that answers the query in Part-1-a. Submit your SQL code.

```
CREATE TABLE IdFilterMV AS
SELECT TweetId FROM Tweets
WHERE TweetId LIKE '%89%' OR TweetId LIKE '%78%';
```

Part 3

a. Draw a precedence graph.
Is the schedule serializable? If not, say "no". If it is, name at least one equivalent serial schedule (e.g., <T1, T2, T3> if the execution is equivalent to individual operations executing in that order)

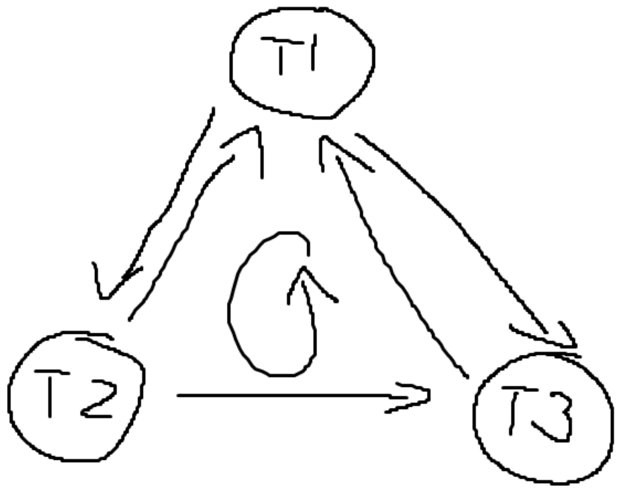
T1	T2	T3
		W(A)
W(A)		
	W(C)	
		R(E)
W(B)		
	W(C)	


$$\begin{tikzcd} & T1 \arrow[dl] & \\ T2 & & T3 \arrow[ul] \arrow[ll] \end{tikzcd}$$

Yes <T3, T2, T1>

b. Draw a precedence graph.
Is the schedule serializable? If not, say "no". If it is, name at least one equivalent serial schedule (e.g., <T1, T2, T3> if the execution is equivalent to individual operations executing in that order)

T1	T2	T3
		R(X)
	R(X)	
		W(X)
R(X)		
W(X)		
		R(X)



No