**Syed Noor Razi Ali**
**2070326**

# 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 "889" or "777" anywhere in the column.

In [10]:
```
# A.

import time

start_timer = time.time()
query = cursor.execute('SELECT * FROM tweets WHERE id_str LIKE "%777%" OR id_str LIKE "%889%";').fetchall()
finish_timer = time.time()

print("\n1.a It took ", finish_timer - start_timer, "seconds to run this query\n")
```

```
1.a It took  0.013962507247924805 seconds to run this query
```

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.

In [9]:
```
# B.

start_timer = time.time()
raw_tweets = urllib.request.urlopen("http://dbgroup.cdm.depaul.edu/DSC450/Module7.txt")

tweets = raw_tweets.readlines()

results = []

for x in tweets:
    try:
        id_str = json.loads(x)["id_str"]
        if "777" in id_str or "889" in id_str:
            results.append(x)

    except:
        None

finish_timer = time.time()

print("\n1.b It took ", finish_timer - start_timer, "seconds to run this python script\n")
```

```
1.b It took  6.800382852554321 seconds to run this python script
```

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

In [12]:
```
# C.

start_timer = time.time()

query = cursor.execute("SELECT COUNT(DISTINCT friends_count) FROM users;").fetchall()

finish_timer = time.time()

print("\n1.c It took ", finish_timer - start_timer, "seconds to run this query\n")
```

```
1.c It took  0.012941598892211914 seconds to run this query
```

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.

In [14]:

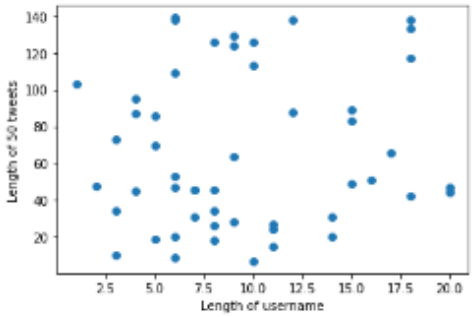
```
# D.

start_timer = time.time()
raw_tweets = urllib.request.urlopen("http://dbgroup.cdm.depaul.edu/DSC450/Module7.txt")

users = raw_tweets.readlines()

#set elements are unique
results = set()

for x in users:
    try:
        friends_count = json.loads(x)["friends_count"]
        if friends_count != None:
            results.append(friends_count)

    except:
        continue

finish_timer = time.time()

print("\n1.d It took ", finish_timer - start_timer, "seconds to run this python script\n")
```

```
1.d It took  9.361337661743164 seconds to run this python script
```

e. Use python to plot the lengths of first 50 tweets (only 50, 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.

In [15]:

```
# E.

import matplotlib.pyplot as plt

raw_tweets = urllib.request.urlopen("http://dbgroup.cdm.depaul.edu/DSC450/Module7.txt")

tweets = raw_tweets.readlines()

len_tweet = []
len_name = []

for x in tweets:
    try:
        #get the username and the full text
        username = json.loads(x)["user"]["name"]
        text = json.loads(x)["text"]

        #stop appending when it reaches a length of 50
        if len(len_name) < 51 and len(len_tweet) < 51:
            len_name.append(len(username))
            len_tweet.append(len(text))
    except:
        None

#create the scatter plot
plt.scatter(len_name, len_tweet)

#add x and y labels
plt.xlabel("Length of username")
plt.ylabel("Length of 50 tweets")

#display
plt.show()
```




## 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.

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

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.

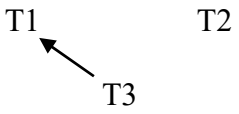
## Part 2

```
In [16]: # A.
         cursor.execute("CREATE INDEX UserID_index ON tweets(user_id);")
Out[16]: <sqlite3.Cursor at 0x1eeff495b20>

In [17]: # B.
         cursor.execute("CREATE INDEX compos_index ON Users(friends_count, screen_name);")
Out[17]: <sqlite3.Cursor at 0x1eeff495b20>

In [18]: # C.
         cursor.execute("CREATE TABLE material_view AS SELECT * FROM tweets WHERE id_str LIKE '%7777%' OR id_str LIKE '%88%';")
Out[18]: <sqlite3.Cursor at 0x1eeff495b20>
```

## 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) | |

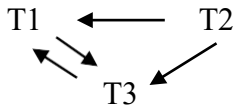



Since there is no cycle, this is conflict serializable. Equivalent serial schedule: < T2, T3, T1>

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) |




Since there is a cycle between T1 and T3, this is NOT conflict serializable.