DSC 450: Database Processing for Large-Scale Analytics
Take-home Final

Part 1

We will use one full day worth of tweets as our input (there are total of 4.4M tweets in this file, but we will intentionally use fewer tweets to run this final):
http://dbgroup.cdm.depaul.edu/DSC450/OneDayOfTweets.txt
Execute and time the following tasks with 130,000 tweets and 650,000 tweets:

a. Use python to download tweets from the web and save to a local text file (not into a database yet, just to a text file). This is as simple as it sounds, all you need is a for-loop that reads lines from the web and writes them into a file.

   NOTE: Do not call read() or readlines(). That command will attempt to read the entire file which is too much data. Clicking on the link in the browser would cause the same problem.

```
loading tweets file took (130k, 650k): (12.73785138130188, 87.981271982193)
def downloadTweets(url, count, file):
    webFd = urllib.request.urlopen(url)
    fd = open(file, 'w+', encoding="utf8")
    start = time.time()
    for i in range(count):
        line = webFd.readline()
        fd.write(str(line) + '\n')
        if i == 130000:
            end = time.time()
        print(i)
    end2 = time.time()
    fd.close()
    webFd.close()
    return end - start, end2-start
```

b. Repeat what you did in part 1-a, but instead of saving tweets to the file, populate the 3-table schema that you previously created in SQLite. Be sure to execute commit and verify that the data has been successfully loaded. Report loaded row counts for each of the 3 tables.

```
def tweetsToDb(url):
    createTableTweets = """
  CREATE TABLE Tweets
  (
    CreatedOn         VARCHAR2(50),
    TweetId           VARCHAR2(50) NOT NULL,
```

```
  UserId            NUMBER(50) NOT NULL,
  GeoId             VARCHAR2(50),
  Body              VARCHAR2(200),
  Source            VARCHAR2(200),
  ReplyToUser       VARCHAR2(50),
  ReplyToName       VARCHAR(100),
  ReplyToStatus     VARCHAR(100),
  Retweets          NUMBER(10),
  Contributors      VARCHAR(100),

  CONSTRAINT Tweets_PK
    PRIMARY KEY(TweetId),

  CONSTRAINT Tweets_FK
    FOREIGN KEY(UserId)
    REFERENCES User(Id),

  CONSTRAINT Tweets_FK2
    FOREIGN KEY(GeoId)
    REFERENCES Geo(Id)
);
"""
  createTableUsers = """
CREATE TABLE Users
(
  Id              NUMBER(50) NOT NULL,
  Name            VARCHAR2(100),
  ScreenName      VARCHAR2(100),
  Description     VARCHAR2(140),
  FriendsCount    NUMBER(50),


  CONSTRAINT Users_PK
    PRIMARY KEY(Id)
);
"""
  createTableGeo = """
CREATE TABLE Geo
(
  Id              VARCHAR2(50) NOT NULL,
  Type            VARCHAR2(100),
  Longitude       NUMBER(100),
  Latitude        NUMBER(140),
```

```
  CONSTRAINT Geo_PK
    PRIMARY KEY(Id)
);
"""

  dropTableTweets = "DROP TABLE IF EXISTS Tweets"
  dropTableUsers = "DROP TABLE IF EXISTS Users"
  dropTableGeo = "DROP TABLE IF EXISTS Geo"
  print('hi')
  conn = sqlite3.connect('final.db') # open db conection
  cursor = conn.cursor()

  cursor.execute(dropTableTweets)
  cursor.execute(dropTableUsers)
  cursor.execute(dropTableGeo)
  cursor.execute(createTableGeo)
  cursor.execute(createTableUsers)
  cursor.execute(createTableTweets)
  print('hi')
  #tweetData = url
  #webFD = urllib.request.urlopen(url)
  #tweets = webFD.readlines(

  ct = 1
  start = time.time()
  tweets = urllib.request.urlopen(url)

  for tweet in tweets:
    currGeo = []
    try:
      #print(tweet)
      tmp = json.loads(tweet.decode('utf-8'))

      if tmp['geo'] is not None:
        geoId = str(tmp['geo']['coordinates'][0]) + str(tmp['geo']['coordinates'][1])
        currGeo = [geoId, tmp['geo']['type'], tmp['geo']['coordinates'][0],
tmp['geo']['coordinates'][1]]
      else:
        geoId = None

      tmpDate = tmp["created_at"].split(' ')
      cleanDate = tmpDate[1] + ' ' + tmpDate[2]  + ' ' + tmpDate[5]
```

```
        currTweet = [cleanDate, tmp["id_str"], tmp['user']['id'], geoId, tmp["text"],
tmp["source"], tmp["in_reply_to_user_id"], tmp["in_reply_to_screen_name"],
tmp["in_reply_to_status_id"], tmp["retweet_count"], tmp["contributors"]]
        currUser = [tmp['user']['id'], tmp['user']['name'], tmp['user']['screen_name'],
tmp['user']['description'], tmp['user']['friends_count']]
        cursor.execute("INSERT OR IGNORE INTO Tweets VALUES(?,?,?,?,?,?,?,?,?,?,?)",
currTweet)
        print(ct)
        cursor.execute("INSERT OR IGNORE INTO Users VALUES(?,?,?,?,?)", currUser)
        if len(currGeo) > 0:
          cursor.execute("INSERT OR IGNORE INTO Geo VALUES(?,?,?,?)", currGeo)

      except ValueError:
        print('error')
      if ct == 130000:
        end130 = time.time()
      if ct == 650000:
        break
      print(ct)
      ct += 1
    end650 = time.time()

    test1 = cursor.execute("SELECT COUNT(*) FROM Tweets;").fetchall()
    test2 = cursor.execute("SELECT COUNT(*) FROM Users;").fetchall()
    test3 = cursor.execute("SELECT COUNT(*) FROM Geo;").fetchall()
    print(test1)
    print(test2)
    print(test3)

    return end130 - start, end650 - start
```

i. Report the row counts for all 3 tables

```
[(649708,)]
[(573179,)]
[(15694,)]
tweets db write took (130k, 650k): (22.86576223373413, 130.84315514564514)
```

NOTE: If your schema contains a foreign key in the Geo table or relies on TweetID as the primary key for the Geo table, you should change your schema. Geo entries should be identified based on the location they represent. The easiest way to create such an ID is by combining lon_lat into a primary key. There should not be any "blank" Geo entries such as (ID, None, None, None).

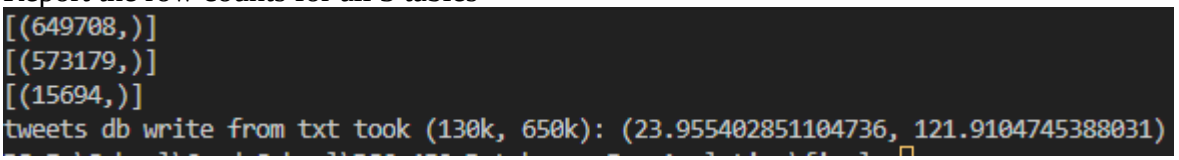
c. Use your locally saved tweet file to repeat the database population step from part-c. That is, load the tweets into the 3-table database using your saved file with tweets. This is the **same** code as in 1-b, but reading tweets from your file, not from the web.

```
def tweetsToDbfromTxt(file):
  createTableTweets = """
 CREATE TABLE Tweets
 (
  CreatedOn         VARCHAR2(50),
  TweetId           VARCHAR2(50) NOT NULL,
  UserId            NUMBER(50) NOT NULL,
  GeoId             VARCHAR2(50),
  Body              VARCHAR2(200),
  Source            VARCHAR2(200),
  ReplyToUser       VARCHAR2(50),
  ReplyToName       VARCHAR(100),
  ReplyToStatus     VARCHAR(100),
  Retweets          NUMBER(10),
  Contributors      VARCHAR(100),

  CONSTRAINT Tweets_PK
   PRIMARY KEY(TweetId),

  CONSTRAINT Tweets_FK
   FOREIGN KEY(UserId)
   REFERENCES User(Id),

  CONSTRAINT Tweets_FK2
   FOREIGN KEY(GeoId)
   REFERENCES Geo(Id)
 );
 """
  createTableUsers = """
 CREATE TABLE Users
 (
  Id              NUMBER(50) NOT NULL,
  Name            VARCHAR2(100),
  ScreenName      VARCHAR2(100),
  Description     VARCHAR2(140),
  FriendsCount    NUMBER(50),


  CONSTRAINT Users_PK
   PRIMARY KEY(Id)
```

```
);
"""
 createTableGeo = """
CREATE TABLE Geo
(
 Id              VARCHAR2(50) NOT NULL,
 Type              VARCHAR2(100),
 Longitude          NUMBER(100),
 Latitude          NUMBER(140),


 CONSTRAINT Geo_PK
  PRIMARY KEY(Id)
);
"""

 dropTableTweets = "DROP TABLE IF EXISTS Tweets"
 dropTableUsers = "DROP TABLE IF EXISTS Users"
 dropTableGeo = "DROP TABLE IF EXISTS Geo"
 print('hi')
 conn = sqlite3.connect('final.db') # open db conection
 cursor = conn.cursor()

 cursor.execute(dropTableTweets)
 cursor.execute(dropTableUsers)
 cursor.execute(dropTableGeo)
 cursor.execute(createTableGeo)
 cursor.execute(createTableUsers)
 cursor.execute(createTableTweets)
 print('hi')
 #tweetData = url
 #webFD = urllib.request.urlopen(url)
 #tweets = webFD.readlines(

 ct = 1
 start = time.time()
 tweets = open(file, 'r', encoding="utf")

 for tweet in tweets:
    currGeo = []
    try:
       #print(tweet)
       tmp = json.loads(tweet)
```

```
            if tmp['geo'] is not None:
                geoId = str(tmp['geo']['coordinates'][0]) + str(tmp['geo']['coordinates'][1])
                currGeo = [geoId, tmp['geo']['type'], tmp['geo']['coordinates'][0],
tmp['geo']['coordinates'][1]]
            else:
                geoId = None

            tmpDate = tmp["created_at"].split(' ')
            cleanDate = tmpDate[1] + ' ' + tmpDate[2]  + ' ' + tmpDate[5]
            currTweet = [cleanDate, tmp["id_str"], tmp['user']['id'], geoId, tmp["text"],
tmp["source"], tmp["in_reply_to_user_id"], tmp["in_reply_to_screen_name"],
tmp["in_reply_to_status_id"], tmp["retweet_count"], tmp["contributors"]]
            currUser = [tmp['user']['id'], tmp['user']['name'], tmp['user']['screen_name'],
tmp['user']['description'], tmp['user']['friends_count']]
            cursor.execute("INSERT OR IGNORE INTO Tweets VALUES(?,?,?,?,?,?,?,?,?,?,?)",
currTweet)
            print(ct)
            cursor.execute("INSERT OR IGNORE INTO Users VALUES(?,?,?,?,?)", currUser)
            if len(currGeo) > 0:
                cursor.execute("INSERT OR IGNORE INTO Geo VALUES(?,?,?,?)", currGeo)

        except ValueError:
            print('error')
        if ct == 130000:
            end130 = time.time()
        print(ct)
        ct += 1
    end650 = time.time()

    test1 = cursor.execute("SELECT COUNT(*) FROM Tweets;").fetchall()
    test2 = cursor.execute("SELECT COUNT(*) FROM Users;").fetchall()
    test3 = cursor.execute("SELECT COUNT(*) FROM Geo;").fetchall()
    print(test1)
    print(test2)
    print(test3)

    return end130 - start, end650 - start
```

i. Report the row counts for all 3 tables



d. Repeat the same step with a batching size of 2,500 (i.e. by inserting 2,500 rows at a time with executemany instead of doing individual inserts). Since many of the tweets are missing a Geo location, its fine for the batches of Geo inserts to be smaller than 2,500.

```
def tweetsToDbBatch(url):
  createTableTweets = """
 CREATE TABLE Tweets
 (
  CreatedOn         VARCHAR2(50),
  TweetId           VARCHAR2(50) NOT NULL,
  UserId            NUMBER(50) NOT NULL,
  GeoId             VARCHAR2(50),
  Body              VARCHAR2(200),
  Source            VARCHAR2(200),
  ReplyToUser       VARCHAR2(50),
  ReplyToName       VARCHAR(100),
  ReplyToStatus     VARCHAR(100),
  Retweets          NUMBER(10),
  Contributors      VARCHAR(100),

  CONSTRAINT Tweets_PK
   PRIMARY KEY(TweetId),

  CONSTRAINT Tweets_FK
   FOREIGN KEY(UserId)
   REFERENCES User(Id),

  CONSTRAINT Tweets_FK2
   FOREIGN KEY(GeoId)
   REFERENCES Geo(Id)
 );
 """
  createTableUsers = """
 CREATE TABLE Users
 (
  Id              NUMBER(50) NOT NULL,
  Name            VARCHAR2(100),
  ScreenName      VARCHAR2(100),
  Description     VARCHAR2(140),
  FriendsCount    NUMBER(50),


  CONSTRAINT Users_PK
   PRIMARY KEY(Id)
```

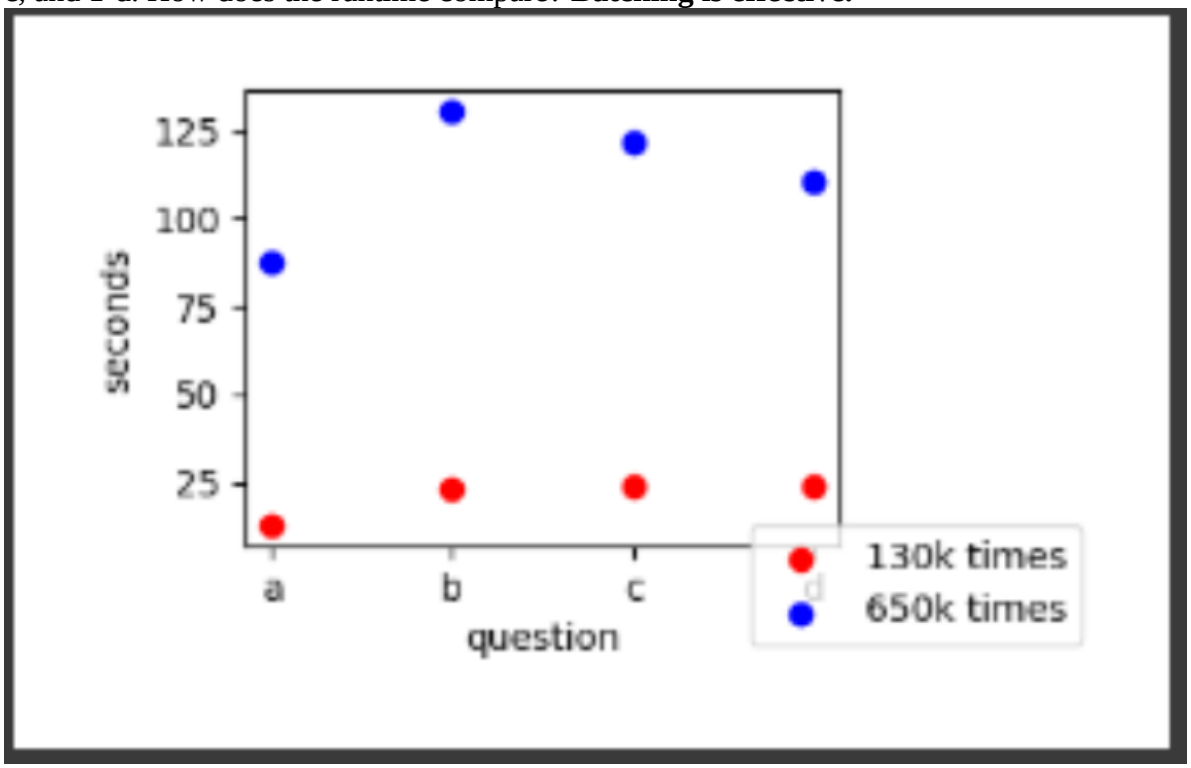
```
);
"""
 createTableGeo = """
CREATE TABLE Geo
(
 Id              VARCHAR2(50) NOT NULL,
 Type             VARCHAR2(100),
 Longitude          NUMBER(100),
 Latitude          NUMBER(140),


 CONSTRAINT Geo_PK
  PRIMARY KEY(Id)
);
"""

 dropTableTweets = "DROP TABLE IF EXISTS Tweets"
 dropTableUsers = "DROP TABLE IF EXISTS Users"
 dropTableGeo = "DROP TABLE IF EXISTS Geo"
 print('hi')
 conn = sqlite3.connect('final.db') # open db conection
 cursor = conn.cursor()

 cursor.execute(dropTableTweets)
 cursor.execute(dropTableUsers)
 cursor.execute(dropTableGeo)
 cursor.execute(createTableGeo)
 cursor.execute(createTableUsers)
 cursor.execute(createTableTweets)
 print('hi')
 #tweetData = url
 #webFD = urllib.request.urlopen(url)
 #tweets = webFD.readlines(

 ct = 1
 start = time.time()
 tweets = urllib.request.urlopen(url)

 batch_size = 2500
 tweet_batch = []
 user_batch = []
 geo_batch = []

 for tweet in tweets:
```

```
        try:
            #print(tweet)
            tmp = json.loads(tweet.decode('utf-8'))

            if tmp['geo'] is not None:
                geoId = str(tmp['geo']['coordinates'][0]) + str(tmp['geo']['coordinates'][1])
                geo_batch.append([geoId, tmp['geo']['type'], tmp['geo']['coordinates'][0],
tmp['geo']['coordinates'][1]])
            else:
                geoId = None

            tmpDate = tmp["created_at"].split(' ')
            cleanDate = tmpDate[1] + ' ' + tmpDate[2]  + ' ' + tmpDate[5]
            tweet_batch.append([cleanDate, tmp["id_str"], tmp['user']['id'], geoId,
tmp["text"], tmp["source"], tmp["in_reply_to_user_id"],
tmp["in_reply_to_screen_name"], tmp["in_reply_to_status_id"],
tmp["retweet_count"], tmp["contributors"]])
            user_batch.append([tmp['user']['id'], tmp['user']['name'],
tmp['user']['screen_name'], tmp['user']['description'], tmp['user']['friends_count']])

            if ct % batch_size == 0:
                cursor.executemany("INSERT OR IGNORE INTO Tweets
VALUES(?,?,?,?,?,?,?,?,?,?,?)", tweet_batch)
                cursor.executemany("INSERT OR IGNORE INTO Users VALUES(?,?,?,?,?)",
user_batch)
                cursor.executemany("INSERT OR IGNORE INTO Geo VALUES(?,?,?,?)",
geo_batch)

                tweet_batch = []
                user_batch = []
                geo_batch = []

        except ValueError:
            print('error')
        if ct == 130000:
            end130 = time.time()
        if ct == 650000:
            break
        print(ct)
        ct += 1

    if tweet_batch:
        cursor.executemany("INSERT OR IGNORE INTO Tweets
VALUES(?,?,?,?,?,?,?,?,?,?,?)", tweet_batch)
```

```
    if user_batch:
        cursor.executemany("INSERT OR IGNORE INTO Users VALUES(?,?,?,?,?)",
user_batch)
    if geo_batch:
        cursor.executemany("INSERT OR IGNORE INTO Geo VALUES(?,?,?,?)", geo_batch)

    end650 = time.time()

    test1 = cursor.execute("SELECT COUNT(*) FROM Tweets;").fetchall()
    test2 = cursor.execute("SELECT COUNT(*) FROM Users;").fetchall()
    test3 = cursor.execute("SELECT COUNT(*) FROM Geo;").fetchall()
    print(test1)
    print(test2)
    print(test3)

    return end130 - start, end650 - start
```

```
[(649708,)]
[(573179,)]
[(15694,)]
tweets db write (batches) took (130k, 650k): (23.80016541481018, 110.46281695365906)
```

e. Plot the resulting runtimes (# of tweets versus runtimes) using matplotlib for 1-a, 1-b, 1-c, and 1-d. How does the runtime compare? **Batching is effective.**



```
import matplotlib.pyplot as plt
import numpy as np

xlst = ['a', 'b', 'c', 'd']
ylst130 = [12.73, 22.86, 23.95, 23.80]
ylst650 = [87.98, 130.84, 121.91, 110.46]


fig = plt.figure()
fig.set_size_inches(10,5)
sp1 = fig.add_subplot(2,3,3)

sp1.scatter(xlst, ylst130, color = 'r', label = '130k times' )
sp1.scatter(xlst, ylst650, color = 'b', label = '650k times' )
plt.ylabel("seconds")
plt.xlabel("question")
fig.legend(loc="right")

plt.show()
```

Part 2

a. Write and execute a SQL query to find the average latitude value for each user ID, using both AVG and SUM/COUNT. This query does not need the User table because User ID is a foreign key in the Tweet table. E.g., something like *SELECT UserID, AVG(latitude), SUM(latitude)/COUNT(latitude) FROM Tweet, Geo WHERE Tweet.GeoFK = Geo.GeoID GROUP BY UserID;*

```
a = cursor.execute("""
  SELECT
    UserId, AVG(Latitude) AS AvgLatitude, SUM(Latitude)/COUNT(Latitude) AS
AvgLatitudeUsingSumCount
  FROM
    Tweets
  JOIN
    Geo ON Tweets.GeoId = Geo.Id
  GROUP BY
    UserId;
  """).fetchall()
  print(a)
```

b. Re-execute the SQL query in part 2-a 5 times and 20 times and measure the total runtime (just re-run the same exact query multiple times using a for-loop, it is as simple as it looks). Does the runtime scale linearly? (i.e., does it take 5X and 20X as much time?)

```
query executed 1 time: 0.10854268074035645
query executed 5 times: 0.500053882598877
query executed 20 times: 2.0139636993408203
```
Yes.

c. Write the equivalent of the 2-a query in python (without using SQL) by reading it from the file with 650,000 tweets.

```
def partTwoC(file):
    tweets = open(file, 'r', encoding="utf")
    dict = {}
    start = time.time()
    ct = 1
    for tweet in tweets:
        try:
            tmp = json.loads(tweet)
            if tmp['geo'] is not None:
                if tmp['user']['id'] in dict:
                    dict[tmp['user']['id']].append(tmp['geo']['coordinates'][0])
                else :
                    dict[tmp['user']['id']] = [tmp['geo']['coordinates'][0]]
        except ValueError:
            print('error')
    end = time.time()
    for key, val in dict.items():
        print(key, (sum(val) / len(val)))
    end = time.time()
    print('time:', str(end - start))
```

d. Re-execute the query in part 2-c 5 times and 20 times and measure the total runtime. Does the runtime scale linearly?

```
Times (1, 5, 20): (21.356687545776367, 103.80078315734863, 412.8703627586365)
```
Yes

e. Write the equivalent of the 2-a query in python by using regular expressions instead of json.loads(). Do not use json.loads() here. Note that you only need to find userid and geo location (if any) for each tweet, you don't need to parse the whole thing.

```
def partTwoE(file):
    tweets = open(file, 'r', encoding="utf")
    userid_pattern = re.compile(r'"user":\{"id":(\d+),')
    geo_pattern =
re.compile(r'"geo":\{"type":"Point","coordinates":\[(\d+\.\d+),\d+\.\d+\]\}')
    dict = {}
    start = time.time()
    ct = 1
    for tweet in tweets:
        try:
            userid_match = re.search(userid_pattern, tweet)
            userid = int(userid_match.group(1))
            geo_match = re.search(geo_pattern, tweet)
            if geo_match:
                lat = float(geo_match.group(1))
                if userid in dict:
                    dict[userid].append(lat)
                else:
                    dict[userid] = [lat]
        except ValueError:
            print('error')
    end = time.time()
    for key, val in dict.items():
        print(key, (sum(val) / len(val)))
    end = time.time()
    #print(dict)
    print('time:', str(end - start))
```

f. Re-execute the query in part 2-e 5 times and 20 times and measure the total runtime. Does the runtime scale linearly?

```
Times (1, 5, 20): (10.869965314865112, 56.131858587265015, 219.4923746585846)
```
yes

Part 3

a. Using the database with 650,000 tweets, create a new table that corresponds to the join of all 3 tables in your database, including records without a geo location. This is the equivalent of a materialized view but since SQLite does not support MVs, we will use CREATE TABLE AS SELECT (instead of CREATE MATERIALIZED VIEW AS SELECT).


```
def partThreeA():
    conn = sqlite3.connect('final.db') # open db conection
    cursor = conn.cursor()

    dropTable = "DROP TABLE IF EXISTS PseduoMV"
    cursor.execute(dropTable)

    pseudoMv = """
    CREATE TABLE PseduoMV AS
    SELECT
        T.CreatedOn, T.TweetId, T.UserId, T.GeoId, T.Body, T.Source, T.ReplyToUser,
T.ReplyToName, T.ReplyToStatus, T.Retweets, T.Contributors,
        U.Name, U.ScreenName, U.Description, U.FriendsCount,
        G.Type, G.Longitude, G.Latitude
    FROM
        Tweets AS T
    LEFT JOIN
        Users AS U ON T.UserId = U.Id
    LEFT JOIN
        Geo AS G ON T.GeoId = G.Id;
    """

    cursor.execute(pseudoMv)
    conn.commit()
```

b. Export the contents of 1) the Tweet table and 2) your new table from 3-a into a new JSON file (i.e., create your own JSON file with just the keys you extracted). You do not need to replicate the structure of the input and can come up with any reasonable keys for each field stored in JSON structure (e.g., you can have longitude as "longitude" key when the location is available).
How do the file sizes compare to the original input file?

483 MB for new json file vs. 1.90 GB for original text fil!

```
def partThreeB():
    conn = sqlite3.connect('final.db')
    cursor = conn.cursor()

    select = "SELECT * FROM PseduoMV;"
    cursor.execute(select)
    rows = cursor.fetchall()

    column_names = [description[0] for description in cursor.description]

    dict_rows = [dict(zip(column_names, row)) for row in rows]

    with open('tweets.json', 'w', encoding='utf-8') as json_file:
        json.dump(dict_rows, json_file, ensure_ascii=False, indent=4)
```

c. Export the contents of 1) the Tweet table and 2) your table from 3-a into a .csv (comma separated value) file.
How do the file size compare to the original input file and to the files in 3-b?

49.6 MB for new csv file vs. 1.90 GB for original text file!

```
def partThreeC():
    conn = sqlite3.connect('final.db')
    cursor = conn.cursor()

    select = "SELECT * FROM PseduoMV;"

    df = pd.read_sql(select, conn)

    df.to_csv('tweets.csv', index=False)
```