Maryam Adebayo

Part 1

1. 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.
    a. Code:
        i.
```
# A
import urllib.request
tweetData = "http://dbgroup.cdm.depaul.edu/DSC450/OneDayOfTweets.txt"
webFD = urllib.request.urlopen(tweetData)


f = open("OneDayOfTweets.txt", "w")

import json
for i in range(50000):
    content = webFD.readline()
    f.write(f"{content}")

f.close()
```

2. 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 by running a SELECT DISCINT of the primary key of each table. Additionally, report the runtime of finding the number of rows for each table.
    a. Code:
        i.
```
start1 = time.time()
print(cursor.execute("SELECT COUNT(DISTINCT id) FROM Users").fetchall())
print(cursor.execute("SELECT COUNT(DISTINCT id_str) FROM Tweets").fetchall())
print(cursor.execute("SELECT COUNT(DISTINCT geo_id) FROM Geo").fetchall())
end1 = time.time()
print("Elapsed time:" + str(end1-start1) + 'seconds')
```
    b. Output:
        i. 110,000:

```
[(69884,)]
[(77503,)]
[(75987,)]
Elapsed time for b110:163.08311915397644seconds
```

ii. 550,000:

```
[(83384,)]
[(93642,)]
[(91779,)]
Elapsed time:0.069100856781005 86seconds
```

3. 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. Time the code used to run this step and report. Specifically, have the code print out the time.

   a. 110,000:

      i. `Elapsed time for c110:1.3031480312347412seconds`

   b. 550000

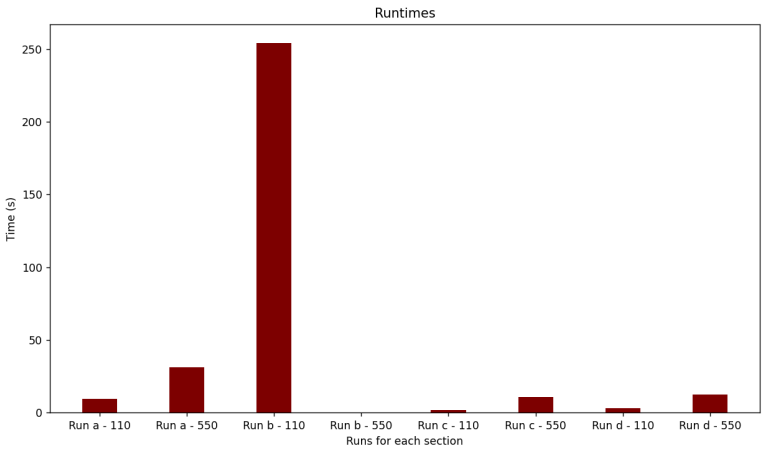
      i. `Elapsed time for c550:6.4927918910980225seconds`

4. 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.

   a.

```
Elapsed time for d110:1.1128082275390625seconds
Elapsed time for d550:4.853403091430664seconds
```

5. 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?

a. 

b. Across all the sections, the runtime seems to be the smallest in section c where we are parsing through the file instead of through the web. Also, in each section, we notice that the run times are significantly smaller when parsing through 110,000 tweets vs 550,000.

6. What would you recommend in a large-scale production environment?
    a. Based on the results, I would recommend first writing the data to a file and parsing through the file to add data to the database. There is a significant difference in time when parsing directly from the website vs parsing from a file.

Part 2

1. Write and execute a SQL query to find the minimum and average longitude and latitude value for each user ID. 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,

MIN(longitude), MAX(latitude) FROM Tweet, Geo WHERE Tweet.GeoFK = Geo.GeoID GROUP BY UserID;

a. Code:

i.
```
cursor.execute("""
SELECT
    Tweets.user_id,
    MIN(Geo.longitude) AS MinLongitude,
    AVG(Geo.longitude) AS AvgLongitude,
    MIN(Geo.latitude) AS MinLatitude,
    AVG(Geo.latitude) AS AvgLatitude
FROM
    Tweets
JOIN
    Geo ON Tweets.geo_id = Geo.geo_id
GROUP BY
    Tweets.user_id;
""")
```

2. Re-execute the SQL query in part 2-a 5 times and 25 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 25X as much time?) What is the average runtime of each individual run? What is the minimum time of each run?

a. Code:

```
start1 = time.time()
for i in range(1,5):
    cursor.execute("""
    SELECT
        Tweets.user_id,
        MIN(Geo.longitude) AS MinLongitude,
        AVG(Geo.longitude) AS AvgLongitude,
        MIN(Geo.latitude) AS MinLatitude,
        AVG(Geo.latitude) AS AvgLatitude
    FROM
        Tweets
    JOIN
        Geo ON Tweets.geo_id = Geo.geo_id
    GROUP BY
        Tweets.user_id;
    """)
end1 = time.time()
print("Elapsed time for executed query 5x:" + str(end1-start1) + 'seconds')
```

```
start2 = time.time()
for i in range(1, 25):
    cursor.execute("""
    SELECT
        Tweets.user_id,
        MIN(Geo.longitude) AS MinLongitude,
        AVG(Geo.longitude) AS AvgLongitude,
        MIN(Geo.latitude) AS MinLatitude,
        AVG(Geo.latitude) AS AvgLatitude
    FROM
        Tweets
    JOIN
        Geo ON Tweets.geo_id = Geo.geo_id
```

i.

b. Output:

i.
```
Elapsed time for executed query 5x:1.4493134021759033seconds
Elapsed time executed query 25x:8.61534571647644seconds
```

c. It seems the runtime scales exponentially, instead of linearly. The average run time for each run for the query that was run 5 times was .2898 second. For the query that was run 25 times was .3446.

The query that was run 5 times was faster on each individual run versus the query that was run 25 times.

3. Write the equivalent of the 2-a query in python (without using SQL) by reading it from the file with 550,000 tweets.

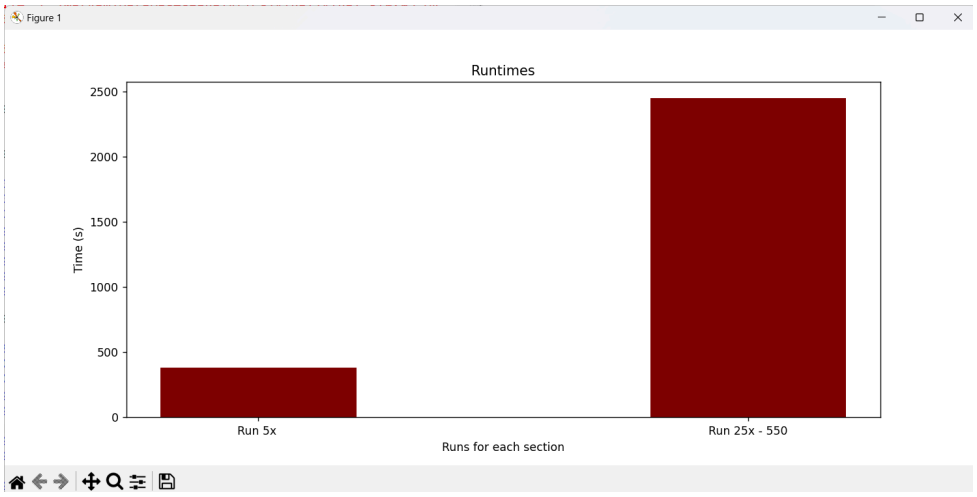
a.
```
#C
from statistics import mean
start3 = time.time()
lst1 =[]
for val in tweets:
  if val in lst1:
    continue
  else:
    if val['geo'] is not None:
        long = [val['geo']['coordinates'][0]]
        lat = [val['geo']['coordinates'][1]]
        lst1.append((val['user']['id'], min(long), min(lat), mean(long), mean(lat)))
end3 = time.time()
print("Elapsed time:" + str(end3-start3) + 'seconds')
```

4. Re-execute the query in part 2-c 5 times and 25 times and measure the total runtime. Does the runtime scale linearly? What is the average runtime of each individual run? What is the minimum time of each run?

a.
```
Elapsed time for python code to run 5x:383.78329968452454seconds
Elapsed time for python code to run 25x:2082.2003803253174seconds
```

b. The run time does scale approximately linearly. The average time for each individual run of the python code that ran 5x was 76.8 seconds. For the code that ran 25x took 83 seconds. The code ran 25x took 6 seconds more than the code ran 5x.

5. Create a visual using matplotlib of 2d showing the distribution of the runtimes (using the average). This should be a single visual that is able to compare the different approaches in a sample manner.

a.


6. Provide a short (i.e., 3-6 sentences) of how you would explain this relationship to a non-technical stakeholder.
    a. I wrote Python code to parse through 550,000 tweets and query each user's minimum and average longitude and latitude. I measured how long it took to run this code 5 times and 25 times. When the code was run 5 times, it took about 500 seconds. When it was run 25 times, it took about 2,500 seconds. If we take the time it took to run 5 times and multiply it by 5, we get the time it took to run it 25 times. This means that the runtimes for this code are scaled linearly.

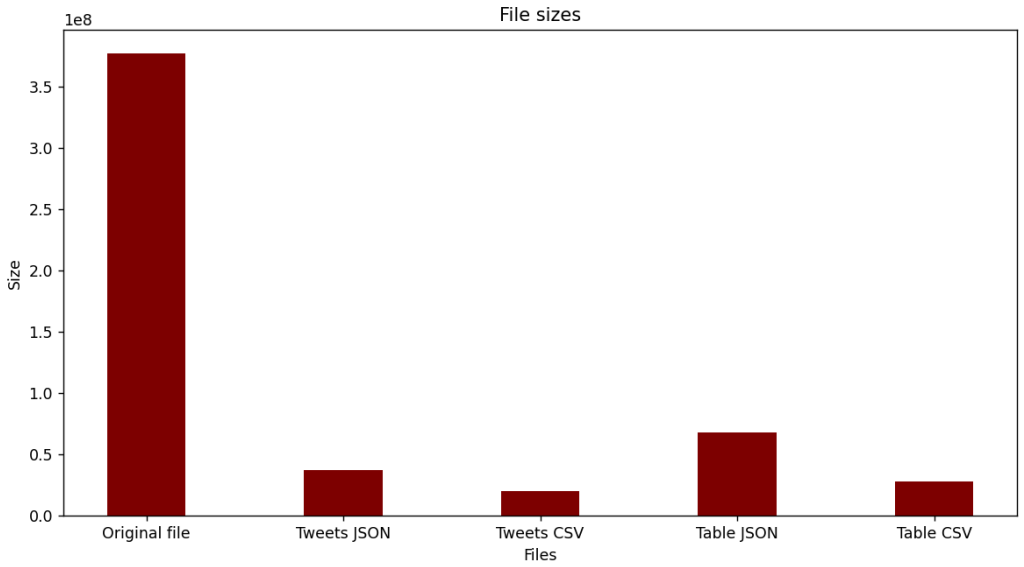
Part 3

1. Using the database with 550,000 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). How many rows are in this table? How long did this take to run? Why would you choose to use a materialized view instead of running the SELECT every time you wanted to access the data?
    a.
```
Elapsed time to create table:5.412626266479492seconds
[(93642,)]
```
        i. Time = 5.41 s
        ii. Rows = 93642
    b. Using a materialized view is far more efficient than running select. You would also avoid writing multiple complex join queries because all the data is in one place.

2. 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? Why would they be the same/different? What is the benefit of the JSON file?
    a. The JSON file was a lot smaller comparatively. Upon looking at the JSON file, it seems the data was uploaded laterally. So, the original file has more breaks in the data which makes the file larger.

3. Export the contents of 1) the Tweet table and 2) your table from 3-a into a .csv (comma separated value) file. How does the file size compare to the original input file and to the files in 3-b? Explain why you would use one of the types instead of the other?
    a. The CSV file is smaller than both the JSON file and the original file. You may want to use a CSV if there are organized columns with a lot less data because it would be easier to read.

4. Create a visualization that compares the different file sizes? How would you explain these differences to a non-technical stakeholder? Provide at least one benefit and con for each of the options before making a final recommendation.

a. 

b. Comparatively, both JSON files and CSV files are more compact than a text file. CSV file is visually better for data with organized columns, where as JSON files seem to be more suitable for adding lots of data from web applications because it's easier to read.

JSON

Pros

- Easy to read when parsing web data.

Cons

- A little larger than a CSV

CSV

Pros

- Compact and efficient

Cons

- Not easy to read when parsing through web data