Database Processing for Large-Scale Analytics

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 50,000 tweets and 250,000 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 and writes them into a file, just don't forget to add '\n' at the end so they are, in fact, on separate lines.

   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.

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.

   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. There should **not** be any "blank" Geo entries such as (ID, None, None, None). The easiest way to create an ID is by combining lon_lat into a primary key.

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.

d. Repeat the same step with a batching size of 4000 (i.e. by inserting 4000 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 4000.

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?

Part 2

a. Write and execute a SQL query to find the 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), MIN(latitude) FROM Tweet, Geo WHERE Tweet.GeoFK = Geo.GeoID GROUP BY UserID;*

b. Re-execute the SQL query in part 2-a 10 times and 50 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 10X and 50X as much time?)

c. Write the equivalent of the 2-a query in python (without using SQL) by reading it from the file with 250,000 tweets.

d. Re-execute the query in part 2-c 10 times and 50 times and measure the total runtime. Does the runtime scale linearly?

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.

f. Re-execute the query in part 2-e 10 times and 50 times and measure the total runtime. Does the runtime scale linearly?

Part 3

a. Using the database with 250,000 tweets, create a new table that corresponds to the join of all 3 tables in your database, <u>including records without a geo location</u>. 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).

b. Export the contents of your 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 does the file size compare to the original input file?

c. Export the contents of 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 file in 3-b?