**CSC 455: Database Processing for Large-Scale Analytics**

**Take-home Final**

**Due 11:00pm, Tuesday, November 24th , 2015**

Be sure to report the running times when requested in the questions.

1. We will use a full day worth of tweets as an input (there are total of 4.4M tweets in this file): http://rasinsrv07.cstcis.cti.depaul.edu/CSC455/OneDayOfTweets.txt

**I was limited by my computer’s computational speed. So this is based on 1000 tweets**

* 1. Create a 3rd table incorporating the Geo table (in addition to tweet and user tables that you already have) and extend your schema accordingly.

You will need to generate an ID for the Geo table primary key (you may use any value or 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 table because there can be multiple tweets sent from the same location). In addition to the primary key column, the geo table should have “type”, “longitude” and “latitude” columns.

* 1. Use python to download from the web and save to a local text file (not into database yet) at least 500,000 lines worth of tweets. Test your code with fewer rows first – you can reduce the number of tweets if your computer is running too slow to handle 500K tweets in a reasonable time. **How long did it take to save?**

**NOTE**: Do NOT call read() or readlines() without any parameters. That command will attempt to read the entire file and you only need 500K rows.

Part 1-B parameters:

tweets loaded to file: 1000

Difference is 20.257 seconds

Performance : 4936.587 operations per second

-----------------------------

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

If you use the posted example code be sure to turn off batching for this part. (i.e., batchRows set to 1). **How long did this step take?**

Part 1-C parameters:

tweets populated in sql table: 1000

Difference is 0.498 seconds

Performance : 200698.422 operations per second

-----------------------------

* 1. Use your locally saved tweet file (created in part-b) to repeat the database population step from part-c. That is, load 500,000 tweets into the 3-table database using your saved file with tweets (do not use the URL to read twitter data). **How does the runtime compare with part-c?**

Part 1-D parameters:

Difference is 0.085 seconds

Performance : 1174687.515 operations per second

-----------------------------

* 1. Re-run the previous step with batching size of 500 (i.e. by inserting 500 rows at a time with executemany). You can adapt the posted example code. **How does the runtime compare when batching is used?**

Part 1-E parameters:

loadTweets took 0.015 seconds.

tweet-Loaded (1026,) rows

tweetuser-Loaded (1008,) rows

tweetgeo-Loaded (20,) rows

-----------------------------

Code Submission: csc455-takehome\_Q1.py (lines 1-375)

* 1. Write and execute SQL queries to do the following. Don’t forget to report the running times in each part – you do not need to report the output:
     1. Find tweets where tweet id\_str contains “44” or “77” anywhere in the column

Part 2a runtime parameters:

i-Difference is 0.002 seconds

Performance : 53994644.696 operations per second

* + 1. Find how many unique values are there in the “in\_reply\_to\_user\_id” column

ii-Difference is 0.001 seconds

Performance : 114786644.773 operations per second

* + 1. Find the tweet(s) with the longest text message

iii-Difference is 0.002 seconds

Performance : 43405815.999 operations per second

* + 1. Find the average longitude and latitude value for each user name.

iv-Difference is 0.0 seconds

Performance : 390895060.578 operations per second

* + 1. Re-execute the query in part iv) 10 times and 100 times and measure the total runtime (just re-run the same exact query using a for-loop). Does the runtime scale linearly? (i.e., does it take 10X and 100X as much time?)

v-Difference is 0.001 seconds

Performance : 93455971.48 operations per second

v-Difference is 0.004 seconds

Performance : 22431832.282 operations per second

\*\*SQL query run 1x: 0.00025582313537597656, 10x: 0.0010700225830078125, 100x: 0.004457950592041016

The 10x run did not scale linearly

The 100x run did not scale linearly

* 1. Write python code that is going to read the locally saved tweet data file from 1-b and perform the equivalent computation for parts 2-i and 2-ii only. How does the runtime compare to the SQL queries?

Part 2b-1:

Difference is 0.052 seconds

Performance : 1941259.181 operations per second

Part 2b-2:

There are: 193 unique in\_reply\_to\_user\_id in the tweet file

Difference is 0.054 seconds

Performance : 1855911.362 operations per second

-----------------------------

* 1. Extra-credit: Perform the python equivalent for 2-iii

Part 2b-extra credit:

Difference is 0.054 seconds

Performance : 1854704.967 operations per second

-----------------------------

* 1. Extra-credit: Perform the python equivalent for 2-iv: **Not attempted**

Code Submission: csc455-takehome\_Q1.py (lines 381 -507)

* 1. Export the contents of the User table from a SQLite table into a sequence of INSERT statements within a file. This is very similar to what you did in Assignment 4. However, you have to add a unique ID column which has to be a string (you cannot use any numbers). Hint: one possibility is to replace digits with letters, e.g., chr(ord('a')+1) gives you a 'b' and chr(ord('a')+2) returns a 'c'

Difference is 1.199 seconds

Performance : 83369.953 operations per second

-----------------------------

* 1. Create a similar collection of INSERT for the User table by reading/parsing data from the local tweet file that you have saved earlier. **How do these compare in runtime? Which method was faster?**

Difference is 1.374 seconds

Performance : 72780.314 operations per second

-----------------------------

Code Submission: csc455-takehome\_Q1.py (lines 518 – 553)

1. Export all three tables (Tweet, User and Geo tables) from the database into a |-separated text file. In this part, you do not have to modify the table within the database, just output file data (do not generate INSERT statements, just raw data)
   1. For the Geo table, create a single default entry for the ‘Unknown’ location and round longitude and latitude to a maximum of 4 digits after the decimal.
   2. For the Tweet table, replace NULLs by a reference to ‘Unknown’ entry (i.e., the foreign key column that references Geo table should refer to the “Unknown” entry you created in part-a. Report how many known/unknown locations there were in total (e.g., 10,000 known, 490,000 unknown, 2% locations are available)
   3. For the User table file add a column (true/false) that specifies whether “screen\_name” or “description” attribute contains within it the “name” attribute of the same user. That is, your output file should contain all of the columns from the User table, plus the new column. You do not have to modify the original User table.

Code Submission: csc455-takehome\_Q1.py (lines 561 – 635\* end of file)