CSC 455: Database Processing for Large-Scale Analytics PROJECT

Due 11:59pm, Tuesday, November 26th.

Be sure to report the running time of each part of your project

- 1. Use the same file that we have for Assignment6: http://rasinsrv07.cstcis.cti.depaul.edu/CSC455/Twitter_2013_11_12.txt
 - a. Use python to download from the web and save to a local text file at least 100,000 lines worth of tweets (you can use my posted code, but be sure to turn off the unnecessary functionality). Do not load the data into a database yet.

ANSWER:

Please see attached file for python code! The time cost is 422.95900011

seconds

b. Repeat what you did in part-a, (save 100,000 tweets to a file reading from the web), but in addition to saving tweets to the file also populate the 3-table schema built in your previous assignments. Be sure to execute commit and verify that the data has been successfully loaded (report row counts for each of the 3 tables). You can use posted sample code, but be sure to turn off batching for this part. (i.e. batchRows set to 1)

ANSWER:

Please see attached file for python code! readTweets took 423.491999865 seconds.

c. Use your locally saved tweet file to repeat the database population step from part-b. That is, load 100,000 tweets into a different 3-table database using your file from either part-a or part-b (instead of the URL). Was it faster to populate the database in this manner?

ANSWER:

Please see attached file for python code! Reading from local file is way better than reading from web, as you can see the results from python code below,

readTweets from local file took 31.8310000896 seconds

d. Write python code that is going to determine the length of the longest line (entire tweet line) among those 100,000 tweets.

ANSWER:

Please see attached file for code. Finding the longest tweet took 6.24599981308 seconds.

e. Create a new single-column table that will hold the entire tweet string with no parsing – use the information from part d) to determine the size of the column. Populate that new table with 100,000 tweets.

ANSWER:

creating and populating the tweet_string table took 17.7309999466 seconds.

f. Pick any of the database loading steps above (b, c, or e) and re-run it with batching size of 100 and 1000 (i.e. by inserting 100/1000 rows at a time with executemany). You can use the posted code.

ANSWER:

2.

This script is faster, 8.15700006485 seconds, than 1.e, which takes 17.7309999466 seconds to execute

- a. Write and execute SQL queries to do the following (don't forget to report the running time):
 - i. Find tweets where tweet ID ends in "300" or "700"

ANSWER:

finding ID by SQL query ending with 700 or 300 takes 0.0360000133514 seconds.

there are 202 id ending with 700 or 300!

ii. Find how many unique values are there in the "in reply to user id" column

ANSWER:

SQL query takes 0.301000118256 seconds, there 20497 distinct In Reply to User ID query found

iii. Find pairs of different tweet ids with the same value of "retweet_count" where one of the two tweets in the pair has geo coordinates (but <u>not</u> both of them do)

ANSWER:

I can't report run time when inserting 100000 values, because everytime I run it, I ended up with computer crash.

if just inserting 1000 values, it reports "SQL query takes 0.825000047684 seconds."

iv. Find the average longitude and latitude value for each user name.

ANSWER:

I can't report time if I insert 100000 values into table. If inserting just 1000 values,

it reports "SQL query takes 0.477999925613 seconds."

v. Re-execute the query in part iv) 10 times and 50 times in a row (in a for-loop, exact same query). Does the runtime scale linearly?

ANSWER:

If inserting just 1000 values, it reports "SQL query takes 4.74799990654 seconds.", which is linear scale!

b. Write python code that is going to read the tweet data file and perform the equivalent computation for parts i) and ii) above. How does the runtime compare?

ANSWER:

result: finding ID ending with 700 or 300 takes 4.36599993706 seconds, which is mucher longer than SQL query, there are 202 id ending with 700 or 300!

there are 20497 unique reply_to_user_id entries. finding unique reply_to_id takes 71.34565427504 seconds, which is mucher longer than SQL query

a. Apply your code from Assignment 4, part-4 to the Tweet table and User table.

ANSWER:

3.

result: output the table from SQLITE# takes 34.9079999924 seconds. Here is a screen shot on the output.

b. Apply your code from Assignment 5, part-4 to the Tweet file (use the list of IDs that end in 700 as your skipList – you can use any method you wish, but you must generate the list of IDs first).

ANSWER:

output the table from SQLITE# takes 0.3407352420 seconds.

c. If you skipped either Assignment 4 or Assignment 5, you can use Assignment6 part-4 code as a substitute.

ANSWER:

I will skip that!

- 4. Export all three tables (Tweet, User and Geo data) from the database into comma-separated text files
 - a. For Geo table, create a default entry which will replace NULLs in Tweet table.

ANSWER:

Please see part c of question 4

b. For User table add a column (true/false) that specifies whether "screen_name" or "description" attribute contains within it the "name" attribute.

ANSWER:

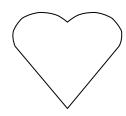
output the table from SQLITE# takes 1.93799996376 seconds.

c. For tweet table, replace NULLs by foreign key reference to the default location (created in part 4-a above) and add a new column with tweet (i.e. text field) length.

ANSWER:

output the table from SQLITE# takes 2.11500000954 seconds.

PART 5 - THANK YOU ALEX!



Maybe I should skip this....^_^