**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://rasinsrv07.cstcis.cti.depaul.edu/CSC455/OneDayOfTweets.txt

Execute the following tasks with 50,000 tweets, 200,000 tweets, and 600,000 tweets.

* 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 and writes them into a file, just don’t forget to add ‘\n’ at the end so they are, in fact, on separate lines.

50,000:

﻿

import urllib

import time

def WriteTweets(url):

tic = time.perf\_counter()

webFD = urllib.request.urlopen(url)

n = 50000

for i in range(n):

ln = webFD.readline()

decoded\_file = ln.decode('utf-8')

output = open('output\_final.txt', 'a')

output.write(decoded\_file)

output.write('\n')

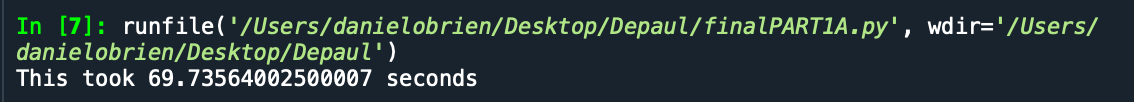
output.close()

toc = time.perf\_counter()

FunctionTime = toc - tic

print('This took {} seconds'.format(FunctionTime))

WriteTweets('http://rasinsrv07.cstcis.cti.depaul.edu/CSC455/OneDayOfTweets.txt')



200,000:

﻿import urllib

import time

def WriteTweets(url):

tic = time.perf\_counter()

webFD = urllib.request.urlopen(url)

n = 200000

for i in range(n):

ln = webFD.readline()

decoded\_file = ln.decode('utf-8')

output = open('output\_final.txt', 'a')

output.write(decoded\_file)

output.write('\n')

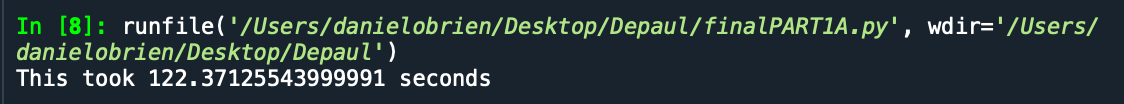
output.close()

toc = time.perf\_counter()

FunctionTime = toc - tic

print('This took {} seconds'.format(FunctionTime))

WriteTweets('http://rasinsrv07.cstcis.cti.depaul.edu/CSC455/OneDayOfTweets.txt')



600,000:

﻿import urllib

import time

def WriteTweets(url):

tic = time.perf\_counter()

webFD = urllib.request.urlopen(url)

n = 600000

for i in range(n):

ln = webFD.readline()

decoded\_file = ln.decode('utf-8')

output = open('output\_final.txt', 'a')

output.write(decoded\_file)

output.write('\n')

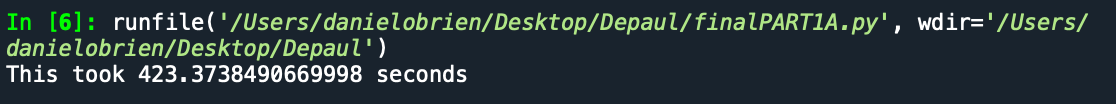
output.close()

toc = time.perf\_counter()

FunctionTime = toc - tic

print('This took {} seconds'.format(FunctionTime))

WriteTweets('http://rasinsrv07.cstcis.cti.depaul.edu/CSC455/OneDayOfTweets.txt')



**NOTE**: Do not call read() or readlines(). That command will attempt to read the entire file which is too much data.

* 1. For *text*, *in\_reply\_to\_user\_id* and *in\_reply\_to\_screenname* in Tweet table and for *screen\_name* in User table, find the length of the longest string in the file in 1-a and compare it to your data type size. You only need to do 1-b for 600,000 tweets and you don’t need the SQLite database for it (i.e., you can do that based on the file itself).

﻿import json

import time

def Queries(txtfile):

tic = time.perf\_counter()

file = open(txtfile, 'r')

lines = file.readlines()

longestText = 0

Longest\_in\_reply\_to\_user\_id = 0

Longest\_in\_reply\_to\_screenname = 0

Longest\_Screen\_name = 0

for line in lines:

tweet = json.loads(line)

text = tweet['text']

in\_reply\_to\_user\_id = tweet['in\_reply\_to\_user\_id\_str']

in\_reply\_to\_screenname = tweet['in\_reply\_to\_screen\_name']

screen\_name = tweet['user']['screen\_name']

if len(text) > longestText:

longestText = len(text)

if in\_reply\_to\_user\_id:

if len(in\_reply\_to\_user\_id) > Longest\_in\_reply\_to\_user\_id:

Longest\_in\_reply\_to\_user\_id = len(in\_reply\_to\_user\_id)

if in\_reply\_to\_screenname:

if len(in\_reply\_to\_screenname) > Longest\_in\_reply\_to\_screenname:

Longest\_in\_reply\_to\_screenname = len(in\_reply\_to\_screenname)

if len(screen\_name) > Longest\_Screen\_name:

Longest\_Screen\_name = len(screen\_name)

toc = time.perf\_counter()

FunctionTime = toc - tic

print('This took {} seconds'.format(FunctionTime))

print("The longest text is {} characters, the longest in\_reply\_to\_user\_id is {} characters, the longest in\_reply\_to\_screen\_name is {} characters, and the longest screen\_name is {} characters".format(longestText, Longest\_in\_reply\_to\_user\_id, Longest\_in\_reply\_to\_screenname, Longest\_Screen\_name))

Queries('output\_final.txt')



* 1. 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 fix 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).

600,000:﻿

import sqlite3

import urllib

import json

import time

def createSchema(url):

tic = time.perf\_counter()

conn = sqlite3.connect('dsc450\_final9.db')

c = conn.cursor()

table1 = """ CREATE TABLE User (

tweetId NUMBER(25),

name VARCHAR2(35),

id\_str NUMBER (30),

screen\_name VARCHAR2(35),

description VARCHAR2(150),

friend\_count NUMBER(10),

CONSTRAINT User\_Primary\_Key

PRIMARY KEY(tweetId, id\_str)

);"""

c.execute(table1)

table2 = """ CREATE TABLE Tweets(

created\_at VARCHAR2(45),

id\_str NUMBER (30),

text VARCHAR2(280),

source VARCHAR2(75),

in\_reply\_to\_user\_id VARCHAR(75),

in\_reply\_to\_screen\_name VARCHAR2(50),

in\_reply\_to\_status\_id VARCHAR2(50),

retweet\_count NUMBER(10),

CONSTRAINT Tweet\_Primary\_Key

PRIMARY KEY(created\_at, id\_str),

Foreign Key (id\_str) REFERENCES User(id\_str)

);"""

c.execute(table2)

table3 = """ CREATE TABLE Geo (

geo VARCHAR2(15),

latitude NUMBER(30.10),

longitude NUMBER(30.10),

id\_str NUMBER (30),

CONSTRAINT geoTablePK

PRIMARY KEY(id\_str, longitude, latitude),

Foreign Key (id\_str) REFERENCES User(id\_str)

);"""

c.execute(table3)

webFD = urllib.request.urlopen(url)

n = 600000

for i in range(n):

ln = webFD.readline()

tweet = json.loads(ln)

tweetid = tweet['id']

id\_str = tweet['id\_str']

name = tweet['user']['name']

screen\_name = tweet['user']['screen\_name']

description = tweet['user']['description']

friend\_count = tweet['user']['friends\_count']

List = [tweetid, id\_str, name, screen\_name, description, friend\_count]

c.executemany("INSERT OR IGNORE INTO User VALUES(?,?,?,?,?,?)", (List,))

conn.commit()

created\_at = tweet['created\_at']

text = tweet['text']

source = tweet['source']

in\_reply\_to\_user\_id = tweet['in\_reply\_to\_user\_id']

in\_reply\_to\_screen\_name = tweet['in\_reply\_to\_screen\_name']

in\_reply\_to\_status\_id = tweet['in\_reply\_to\_status\_id']

retweetCount = tweet['retweet\_count']

List2 = [created\_at, id\_str, text, source, in\_reply\_to\_user\_id, in\_reply\_to\_screen\_name, in\_reply\_to\_status\_id, retweetCount]

c.executemany("INSERT OR IGNORE INTO Tweets VALUES(?,?,?,?,?,?,?,?)", (List2,))

if tweet['geo']:

geoType = tweet['geo']['type']

longLat = tweet['geo']['coordinates']

long = longLat[0]

lat = longLat[1]

List3 = [geoType, long, lat, id\_str]

c.executemany("INSERT OR IGNORE INTO Geo VALUES(?,?,?,?);", (List3,))

conn.commit()

query1 = "SELECT COUNT(\*) FROM User;"

query2 = "SELECT COUNT(\*) FROM Tweets"

query3 = "SELECT COUNT(\*) FROM Geo"

x = c.execute(query1).fetchall()

y = c.execute(query2).fetchall()

z = c.execute(query3).fetchall()

print("There are {} rows in User table.".format(x))

print("There are {} rows in Tweets table.".format(y))

print("There are {} rows in Geo table.".format(z))

toc = time.perf\_counter()

FunctionTime = toc - tic

print('This took {} seconds'.format(FunctionTime))

createSchema('http://rasinsrv07.cstcis.cti.depaul.edu/CSC455/OneDayOfTweets.txt')

Text

Description automatically generated

200,000:

﻿import sqlite3

import urllib

import json

import time

def createSchema(url):

tic = time.perf\_counter()

conn = sqlite3.connect('dsc450\_final90.db')

c = conn.cursor()

table1 = """ CREATE TABLE User (

tweetId NUMBER(25),

name VARCHAR2(35),

id\_str NUMBER (30),

screen\_name VARCHAR2(35),

description VARCHAR2(150),

friend\_count NUMBER(10),

CONSTRAINT User\_Primary\_Key

PRIMARY KEY(tweetId, id\_str)

);"""

c.execute(table1)

table2 = """ CREATE TABLE Tweets(

created\_at VARCHAR2(45),

id\_str NUMBER (30),

text VARCHAR2(280),

source VARCHAR2(75),

in\_reply\_to\_user\_id VARCHAR(75),

in\_reply\_to\_screen\_name VARCHAR2(50),

in\_reply\_to\_status\_id VARCHAR2(50),

retweet\_count NUMBER(10),

CONSTRAINT Tweet\_Primary\_Key

PRIMARY KEY(created\_at, id\_str),

Foreign Key (id\_str) REFERENCES User(id\_str)

);"""

c.execute(table2)

table3 = """ CREATE TABLE Geo (

geo VARCHAR2(15),

latitude NUMBER(30.10),

longitude NUMBER(30.10),

id\_str NUMBER (30),

CONSTRAINT geoTablePK

PRIMARY KEY(id\_str, longitude, latitude),

Foreign Key (id\_str) REFERENCES User(id\_str)

);"""

c.execute(table3)

webFD = urllib.request.urlopen(url)

n = 200000

for i in range(n):

ln = webFD.readline()

tweet = json.loads(ln)

tweetid = tweet['id']

id\_str = tweet['id\_str']

name = tweet['user']['name']

screen\_name = tweet['user']['screen\_name']

description = tweet['user']['description']

friend\_count = tweet['user']['friends\_count']

List = [tweetid, id\_str, name, screen\_name, description, friend\_count]

c.executemany("INSERT OR IGNORE INTO User VALUES(?,?,?,?,?,?)", (List,))

conn.commit()

created\_at = tweet['created\_at']

text = tweet['text']

source = tweet['source']

in\_reply\_to\_user\_id = tweet['in\_reply\_to\_user\_id']

in\_reply\_to\_screen\_name = tweet['in\_reply\_to\_screen\_name']

in\_reply\_to\_status\_id = tweet['in\_reply\_to\_status\_id']

retweetCount = tweet['retweet\_count']

List2 = [created\_at, id\_str, text, source, in\_reply\_to\_user\_id, in\_reply\_to\_screen\_name, in\_reply\_to\_status\_id, retweetCount]

c.executemany("INSERT OR IGNORE INTO Tweets VALUES(?,?,?,?,?,?,?,?)", (List2,))

if tweet['geo']:

geoType = tweet['geo']['type']

longLat = tweet['geo']['coordinates']

long = longLat[0]

lat = longLat[1]

List3 = [geoType, long, lat, id\_str]

c.executemany("INSERT OR IGNORE INTO Geo VALUES(?,?,?,?);", (List3,))

conn.commit()

query1 = "SELECT COUNT(\*) FROM User;"

query2 = "SELECT COUNT(\*) FROM Tweets"

query3 = "SELECT COUNT(\*) FROM Geo"

x = c.execute(query1).fetchall()

y = c.execute(query2).fetchall()

z = c.execute(query3).fetchall()

print("There are {} rows in User table.".format(x))

print("There are {} rows in Tweets table.".format(y))

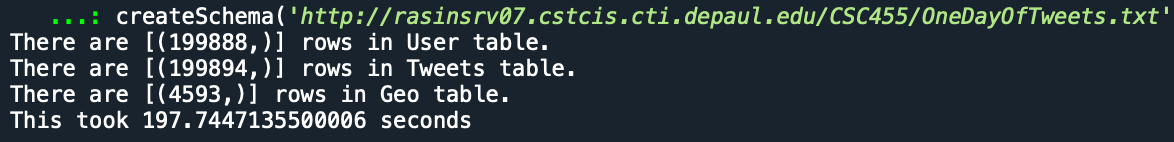
print("There are {} rows in Geo table.".format(z))

toc = time.perf\_counter()

FunctionTime = toc - tic

print('This took {} seconds'.format(FunctionTime))

createSchema('http://rasinsrv07.cstcis.cti.depaul.edu/CSC455/OneDayOfTweets.txt')



50,000:

﻿

import sqlite3

import urllib

import json

import time

def createSchema(url):

tic = time.perf\_counter()

conn = sqlite3.connect('dsc450\_final91.db')

c = conn.cursor()

table1 = """ CREATE TABLE User (

tweetId NUMBER(25),

name VARCHAR2(35),

id\_str NUMBER (30),

screen\_name VARCHAR2(35),

description VARCHAR2(150),

friend\_count NUMBER(10),

CONSTRAINT User\_Primary\_Key

PRIMARY KEY(tweetId, id\_str)

);"""

c.execute(table1)

table2 = """ CREATE TABLE Tweets(

created\_at VARCHAR2(45),

id\_str NUMBER (30),

text VARCHAR2(280),

source VARCHAR2(75),

in\_reply\_to\_user\_id VARCHAR(75),

in\_reply\_to\_screen\_name VARCHAR2(50),

in\_reply\_to\_status\_id VARCHAR2(50),

retweet\_count NUMBER(10),

CONSTRAINT Tweet\_Primary\_Key

PRIMARY KEY(created\_at, id\_str),

Foreign Key (id\_str) REFERENCES User(id\_str)

);"""

c.execute(table2)

table3 = """ CREATE TABLE Geo (

geo VARCHAR2(15),

latitude NUMBER(30.10),

longitude NUMBER(30.10),

id\_str NUMBER (30),

CONSTRAINT geoTablePK

PRIMARY KEY(id\_str, longitude, latitude),

Foreign Key (id\_str) REFERENCES User(id\_str)

);"""

c.execute(table3)

webFD = urllib.request.urlopen(url)

n = 50000

for i in range(n):

ln = webFD.readline()

tweet = json.loads(ln)

tweetid = tweet['id']

id\_str = tweet['id\_str']

name = tweet['user']['name']

screen\_name = tweet['user']['screen\_name']

description = tweet['user']['description']

friend\_count = tweet['user']['friends\_count']

List = [tweetid, id\_str, name, screen\_name, description, friend\_count]

c.executemany("INSERT OR IGNORE INTO User VALUES(?,?,?,?,?,?)", (List,))

conn.commit()

created\_at = tweet['created\_at']

text = tweet['text']

source = tweet['source']

in\_reply\_to\_user\_id = tweet['in\_reply\_to\_user\_id']

in\_reply\_to\_screen\_name = tweet['in\_reply\_to\_screen\_name']

in\_reply\_to\_status\_id = tweet['in\_reply\_to\_status\_id']

retweetCount = tweet['retweet\_count']

List2 = [created\_at, id\_str, text, source, in\_reply\_to\_user\_id, in\_reply\_to\_screen\_name, in\_reply\_to\_status\_id, retweetCount]

c.executemany("INSERT OR IGNORE INTO Tweets VALUES(?,?,?,?,?,?,?,?)", (List2,))

if tweet['geo']:

geoType = tweet['geo']['type']

longLat = tweet['geo']['coordinates']

long = longLat[0]

lat = longLat[1]

List3 = [geoType, long, lat, id\_str]

c.executemany("INSERT OR IGNORE INTO Geo VALUES(?,?,?,?);", (List3,))

conn.commit()

query1 = "SELECT COUNT(\*) FROM User;"

query2 = "SELECT COUNT(\*) FROM Tweets"

query3 = "SELECT COUNT(\*) FROM Geo"

x = c.execute(query1).fetchall()

y = c.execute(query2).fetchall()

z = c.execute(query3).fetchall()

print("There are {} rows in User table.".format(x))

print("There are {} rows in Tweets table.".format(y))

print("There are {} rows in Geo table.".format(z))

toc = time.perf\_counter()

FunctionTime = toc - tic

print('This took {} seconds'.format(FunctionTime))

createSchema('http://rasinsrv07.cstcis.cti.depaul.edu/CSC455/OneDayOfTweets.txt')

Text

Description automatically generated

* 1. 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-c, but reading tweets from your file, not from the web.

50,000:

﻿import sqlite3

import json

import time

def createSchema(textfile):

tic = time.perf\_counter()

conn = sqlite3.connect('dsc450\_final3.db')

c = conn.cursor()

table1 = """ CREATE TABLE User (

tweetId NUMBER(25),

name VARCHAR2(35),

id\_str NUMBER (30),

userid NUMBER (30),

screen\_name VARCHAR2(35),

description VARCHAR2(150),

friend\_count NUMBER(10),

CONSTRAINT User\_Primary\_Key

PRIMARY KEY(tweetId, id\_str)

);"""

c.execute(table1)

table2 = """ CREATE TABLE Tweets(

created\_at VARCHAR2(45),

id\_str NUMBER (30),

userid NUMBER(30),

text VARCHAR2(280),

source VARCHAR2(75),

in\_reply\_to\_user\_id VARCHAR(75),

in\_reply\_to\_screen\_name VARCHAR2(50),

in\_reply\_to\_status\_id VARCHAR2(50),

retweet\_count NUMBER(10),

CONSTRAINT Tweet\_Primary\_Key

PRIMARY KEY(created\_at, id\_str),

Foreign Key (userid) REFERENCES User(userid)

);"""

c.execute(table2)

table3 = """ CREATE TABLE Geo (

geo VARCHAR2(15),

latitude NUMBER(30.10),

longitude NUMBER(30.10),

id\_str NUMBER (30),

userid NUMBER (30),

CONSTRAINT geoTablePK

PRIMARY KEY(id\_str, longitude, latitude),

Foreign Key (userid) REFERENCES User(userid)

);"""

c.execute(table3)

file = open(textfile, 'r')

n = 50000

for i in range(n):

ln = file.readline()

tweet = json.loads(ln)

tweetid = tweet['id']

id\_str = tweet['id\_str']

name = tweet['user']['name']

screen\_name = tweet['user']['screen\_name']

userid = tweet['user']['id']

description = tweet['user']['description']

friend\_count = tweet['user']['friends\_count']

List = [tweetid, id\_str, name, userid, screen\_name, description, friend\_count]

c.executemany("INSERT OR IGNORE INTO User VALUES(?,?,?,?,?,?,?)", (List,))

conn.commit()

created\_at = tweet['created\_at']

text = tweet['text']

source = tweet['source']

in\_reply\_to\_user\_id = tweet['in\_reply\_to\_user\_id']

in\_reply\_to\_screen\_name = tweet['in\_reply\_to\_screen\_name']

in\_reply\_to\_status\_id = tweet['in\_reply\_to\_status\_id']

retweetCount = tweet['retweet\_count']

List2 = [created\_at, id\_str, userid, text, source, in\_reply\_to\_user\_id, in\_reply\_to\_screen\_name, in\_reply\_to\_status\_id, retweetCount]

c.executemany("INSERT OR IGNORE INTO Tweets VALUES(?,?,?,?,?,?,?,?,?)", (List2,))

if tweet['geo']:

geoType = tweet['geo']['type']

longLat = tweet['geo']['coordinates']

long = longLat[0]

lat = longLat[1]

List3 = [geoType, long, lat, id\_str, userid]

c.executemany("INSERT OR IGNORE INTO Geo VALUES(?,?,?,?,?);", (List3,))

conn.commit()

query1 = "SELECT COUNT(\*) FROM User;"

query2 = "SELECT COUNT(\*) FROM Tweets"

query3 = "SELECT COUNT(\*) FROM Geo"

x = c.execute(query1).fetchall()

y = c.execute(query2).fetchall()

z = c.execute(query3).fetchall()

print("There are {} rows in User table.".format(x))

print("There are {} rows in Tweets table.".format(y))

print("There are {} rows in Geo table.".format(z))

toc = time.perf\_counter()

FunctionTime = toc - tic

print('This took {} seconds'.format(FunctionTime))

createSchema('output\_final.txt')

Text

Description automatically generated

200,000:

﻿import sqlite3

import json

import time

def createSchema(textfile):

tic = time.perf\_counter()

conn = sqlite3.connect('dsc450\_final2.db')

c = conn.cursor()

table1 = """ CREATE TABLE User (

tweetId NUMBER(25),

name VARCHAR2(35),

id\_str NUMBER (30),

userid NUMBER (30),

screen\_name VARCHAR2(35),

description VARCHAR2(150),

friend\_count NUMBER(10),

CONSTRAINT User\_Primary\_Key

PRIMARY KEY(tweetId, id\_str)

);"""

c.execute(table1)

table2 = """ CREATE TABLE Tweets(

created\_at VARCHAR2(45),

id\_str NUMBER (30),

userid NUMBER(30),

text VARCHAR2(280),

source VARCHAR2(75),

in\_reply\_to\_user\_id VARCHAR(75),

in\_reply\_to\_screen\_name VARCHAR2(50),

in\_reply\_to\_status\_id VARCHAR2(50),

retweet\_count NUMBER(10),

CONSTRAINT Tweet\_Primary\_Key

PRIMARY KEY(created\_at, id\_str),

Foreign Key (userid) REFERENCES User(userid)

);"""

c.execute(table2)

table3 = """ CREATE TABLE Geo (

geo VARCHAR2(15),

latitude NUMBER(30.10),

longitude NUMBER(30.10),

id\_str NUMBER (30),

userid NUMBER (30),

CONSTRAINT geoTablePK

PRIMARY KEY(id\_str, longitude, latitude),

Foreign Key (userid) REFERENCES User(userid)

);"""

c.execute(table3)

file = open(textfile, 'r')

n = 200000

for i in range(n):

ln = file.readline()

tweet = json.loads(ln)

tweetid = tweet['id']

id\_str = tweet['id\_str']

name = tweet['user']['name']

screen\_name = tweet['user']['screen\_name']

userid = tweet['user']['id']

description = tweet['user']['description']

friend\_count = tweet['user']['friends\_count']

List = [tweetid, id\_str, name, userid, screen\_name, description, friend\_count]

c.executemany("INSERT OR IGNORE INTO User VALUES(?,?,?,?,?,?,?)", (List,))

conn.commit()

created\_at = tweet['created\_at']

text = tweet['text']

source = tweet['source']

in\_reply\_to\_user\_id = tweet['in\_reply\_to\_user\_id']

in\_reply\_to\_screen\_name = tweet['in\_reply\_to\_screen\_name']

in\_reply\_to\_status\_id = tweet['in\_reply\_to\_status\_id']

retweetCount = tweet['retweet\_count']

List2 = [created\_at, id\_str, userid, text, source, in\_reply\_to\_user\_id, in\_reply\_to\_screen\_name, in\_reply\_to\_status\_id, retweetCount]

c.executemany("INSERT OR IGNORE INTO Tweets VALUES(?,?,?,?,?,?,?,?,?)", (List2,))

if tweet['geo']:

geoType = tweet['geo']['type']

longLat = tweet['geo']['coordinates']

long = longLat[0]

lat = longLat[1]

List3 = [geoType, long, lat, id\_str, userid]

c.executemany("INSERT OR IGNORE INTO Geo VALUES(?,?,?,?,?);", (List3,))

conn.commit()

query1 = "SELECT COUNT(\*) FROM User;"

query2 = "SELECT COUNT(\*) FROM Tweets"

query3 = "SELECT COUNT(\*) FROM Geo"

x = c.execute(query1).fetchall()

y = c.execute(query2).fetchall()

z = c.execute(query3).fetchall()

print("There are {} rows in User table.".format(x))

print("There are {} rows in Tweets table.".format(y))

print("There are {} rows in Geo table.".format(z))

toc = time.perf\_counter()

FunctionTime = toc - tic

print('This took {} seconds'.format(FunctionTime))

createSchema('output\_final.txt')

Text

Description automatically generated

600,000:

﻿import sqlite3

import json

import time

def createSchema(textfile):

tic = time.perf\_counter()

conn = sqlite3.connect('dsc450\_final7.db')

c = conn.cursor()

table1 = """ CREATE TABLE User (

tweetId NUMBER(25),

name VARCHAR2(35),

id\_str NUMBER (30),

screen\_name VARCHAR2(35),

description VARCHAR2(150),

friend\_count NUMBER(10),

CONSTRAINT User\_Primary\_Key

PRIMARY KEY(tweetId, id\_str)

);"""

c.execute(table1)

table2 = """ CREATE TABLE Tweets(

created\_at VARCHAR2(45),

id\_str NUMBER (30),

text VARCHAR2(280),

source VARCHAR2(75),

in\_reply\_to\_user\_id VARCHAR(75),

in\_reply\_to\_screen\_name VARCHAR2(50),

in\_reply\_to\_status\_id VARCHAR2(50),

retweet\_count NUMBER(10),

CONSTRAINT Tweet\_Primary\_Key

PRIMARY KEY(created\_at, id\_str),

Foreign Key (id\_str) REFERENCES User(id\_str)

);"""

c.execute(table2)

table3 = """ CREATE TABLE Geo (

geo VARCHAR2(15),

latitude NUMBER(30.10),

longitude NUMBER(30.10),

id\_str NUMBER (30),

CONSTRAINT geoTablePK

PRIMARY KEY(id\_str, longitude, latitude),

Foreign Key (id\_str) REFERENCES User(id\_str)

);"""

c.execute(table3)

file = open(textfile, 'r')

n = 600000

for i in range(n):

ln = file.readline()

tweet = json.loads(ln)

tweetid = tweet['id']

id\_str = tweet['id\_str']

name = tweet['user']['name']

screen\_name = tweet['user']['screen\_name']

description = tweet['user']['description']

friend\_count = tweet['user']['friends\_count']

List = [tweetid, id\_str, name, screen\_name, description, friend\_count]

c.executemany("INSERT OR IGNORE INTO User VALUES(?,?,?,?,?,?)", (List,))

conn.commit()

created\_at = tweet['created\_at']

text = tweet['text']

source = tweet['source']

in\_reply\_to\_user\_id = tweet['in\_reply\_to\_user\_id']

in\_reply\_to\_screen\_name = tweet['in\_reply\_to\_screen\_name']

in\_reply\_to\_status\_id = tweet['in\_reply\_to\_status\_id']

retweetCount = tweet['retweet\_count']

List2 = [created\_at, id\_str, text, source, in\_reply\_to\_user\_id, in\_reply\_to\_screen\_name, in\_reply\_to\_status\_id, retweetCount]

c.executemany("INSERT OR IGNORE INTO Tweets VALUES(?,?,?,?,?,?,?,?)", (List2,))

if tweet['geo']:

geoType = tweet['geo']['type']

longLat = tweet['geo']['coordinates']

long = longLat[0]

lat = longLat[1]

List3 = [geoType, long, lat, id\_str]

c.executemany("INSERT OR IGNORE INTO Geo VALUES(?,?,?,?);", (List3,))

conn.commit()

query1 = "SELECT COUNT(\*) FROM User;"

query2 = "SELECT COUNT(\*) FROM Tweets"

query3 = "SELECT COUNT(\*) FROM Geo"

x = c.execute(query1).fetchall()

y = c.execute(query2).fetchall()

z = c.execute(query3).fetchall()

print("There are {} rows in User table.".format(x))

print("There are {} rows in Tweets table.".format(y))

print("There are {} rows in Geo table.".format(z))

toc = time.perf\_counter()

FunctionTime = toc - tic

print('This took {} seconds'.format(FunctionTime))

createSchema('output\_final.txt')

Text

Description automatically generated

* 1. Repeat the same step with a batching size of 1000 (i.e. by inserting 1000 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 (however many geo locations were accumulated for each 1000 tweets).

For the different numbers of tweets, I had this code read the file with 50,000, 200,000 and 600,000 tweets respectively. The output times are recorded below.

﻿import sqlite3

import json

import time

def createSchema(textfile):

tic = time.perf\_counter()

conn = sqlite3.connect('dsc450\_final29.db')

c = conn.cursor()

table1 = """ CREATE TABLE User (

tweetId NUMBER(25),

name VARCHAR2(35),

id\_str NUMBER (30),

screen\_name VARCHAR2(35),

description VARCHAR2(150),

friend\_count NUMBER(10),

CONSTRAINT User\_Primary\_Key

PRIMARY KEY(tweetId, id\_str)

);"""

c.execute(table1)

table2 = """ CREATE TABLE Tweets(

created\_at VARCHAR2(45),

id\_str NUMBER (30),

text VARCHAR2(280),

source VARCHAR2(75),

in\_reply\_to\_user\_id VARCHAR(75),

in\_reply\_to\_screen\_name VARCHAR2(50),

in\_reply\_to\_status\_id VARCHAR2(50),

retweet\_count NUMBER(10),

CONSTRAINT Tweet\_Primary\_Key

PRIMARY KEY(created\_at, id\_str),

Foreign Key (id\_str) REFERENCES User(id\_str)

);"""

c.execute(table2)

table3 = """ CREATE TABLE Geo (

geo VARCHAR2(15),

latitude NUMBER(30.10),

longitude NUMBER(30.10),

id\_str NUMBER (30),

CONSTRAINT geoTablePK

PRIMARY KEY(id\_str, longitude, latitude),

Foreign Key (id\_str) REFERENCES User(id\_str)

);"""

c.execute(table3)

with open(textfile, 'r') as file:

lines = []

for line in file:

lines.append(line)

if len(lines) >= 1000:

inserts1 = []

inserts2 = []

inserts3 = []

for ln in lines:

tweet = json.loads(ln)

tweetid = tweet['id']

id\_str = tweet['id\_str']

name = tweet['user']['name']

screen\_name = tweet['user']['screen\_name']

description = tweet['user']['description']

friend\_count = tweet['user']['friends\_count']

List = (tweetid, id\_str, name, screen\_name, description, friend\_count)

inserts1.append(List)

created\_at = tweet['created\_at']

text = tweet['text']

source = tweet['source']

in\_reply\_to\_user\_id = tweet['in\_reply\_to\_user\_id']

in\_reply\_to\_screen\_name = tweet['in\_reply\_to\_screen\_name']

in\_reply\_to\_status\_id = tweet['in\_reply\_to\_status\_id']

retweetCount = tweet['retweet\_count']

List2 = (created\_at, id\_str, text, source, in\_reply\_to\_user\_id, in\_reply\_to\_screen\_name, in\_reply\_to\_status\_id, retweetCount)

inserts2.append(List2)

if tweet['geo']:

geoType = tweet['geo']['type']

longLat = tweet['geo']['coordinates']

long = longLat[0]

lat = longLat[1]

List3 = [geoType, long, lat, id\_str]

inserts3.append(List3)

c.executemany("INSERT OR IGNORE INTO User VALUES(?,?,?,?,?,?);", inserts1)

c.executemany("INSERT OR IGNORE INTO Tweets VALUES(?,?,?,?,?,?,?,?);", inserts2)

c.executemany("INSERT OR IGNORE INTO Geo VALUES(?,?,?,?);", inserts3)

conn.commit()

lines = []

query1 = "SELECT COUNT(\*) FROM User;"

query2 = "SELECT COUNT(\*) FROM Tweets"

query3 = "SELECT COUNT(\*) FROM Geo"

x = c.execute(query1).fetchall()

y = c.execute(query2).fetchall()

z = c.execute(query3).fetchall()

print("There are {} rows in User table.".format(x))

print("There are {} rows in Tweets table.".format(y))

print("There are {} rows in Geo table.".format(z))

toc = time.perf\_counter()

FunctionTime = toc - tic

print('This took {} seconds'.format(FunctionTime))

createSchema('output\_final.txt')

50,000:

Text

Description automatically generated

200,000:

Text

Description automatically generated

600,000:

Text

Description automatically generated

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

On my plot, I used color to differentiate from each problem. I used red for part 1a, blue for part 1c, red for part 1d, and black for part 1e. As can be seen on the plot, the code for part 1e ran much faster than the other parts for each tweet amount. Parts 1a ran the second fastest. Parts 1c and 1d ran in a similar amount of time with part 1d running just a bit quicker.

**Chart, scatter chart

Description automatically generated**

**﻿**

**import matplotlib.pyplot as plt**

**plt.plot([69.74, 122.37, 423.37], [50000, 200000, 600000], 'ro')**

**plt.plot([50.89, 197.74, 576.52], [50000, 200000, 600000], 'bo')**

**plt.plot([47.35, 187.50, 573.67], [50000, 200000, 600000], 'go')**

**plt.plot([2.76, 11.21, 37.81], [50000, 200000, 600000], 'ko')**

**plt.ylabel('Number of Tweets')**

**plt.xlabel('Time in Seconds')**

**plt.savefig('Part1Final.png')**

**plt.show()**

**Part 2**

1. 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, AVG(longitude), AVG(latitude) FROM Tweet, Geo WHERE Tweet.GeoID = Geo.GeoID;*

﻿ ﻿

import sqlite3

import time

def createSchema(db):

tic = time.perf\_counter()

conn = sqlite3.connect(db)

c = conn.cursor()

query1 = """ SELECT userid, AVG(longitude), AVG(latitude)

FROM Tweets, Geo

WHERE Tweets.id\_str = Geo.id\_str

GROUP BY userid; """

x = c.execute(query1).fetchall()

print (x)

toc = time.perf\_counter()

FunctionTime = toc - tic

print('This took {} seconds'.format(FunctionTime))

createSchema('dsc450\_final1.db')

A picture containing text, plaque

Description automatically generated

1. Re-execute the query in part 2-a 10 times and 100 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 100X as much time?)

As can be seen by the outputs above and below, running this query once took roughly 1.58 seconds to run this code once, about 3.34 seconds to run this code 10 times, which indicates a little less than twice the time for 10 times as many tweets. This seems to indicate that the runtime is faster than a linear scale. It takes about 21. 41 seconds to run 100 times, which is about 14 times the time it takes to run 1 query. This is much faster than a linear scale. It would seem that the more times we run this query, the less time per query it would take.

10 times:

﻿import sqlite3

import time

def createSchema(db):

tic = time.perf\_counter()

conn = sqlite3.connect(db)

c = conn.cursor()

n = 10

for i in range(n):

query1 = """ SELECT userid, AVG(longitude), AVG(latitude)

FROM Tweets, Geo

WHERE Tweets.id\_str = Geo.id\_str

GROUP BY userid; """

x = c.execute(query1).fetchall()

print (x)

toc = time.perf\_counter()

FunctionTime = toc - tic

print('This took {} seconds'.format(FunctionTime))

createSchema('dsc450\_final1.db')

Text

Description automatically generated

100 times:

﻿ ﻿import sqlite3

import time

def createSchema(db):

tic = time.perf\_counter()

conn = sqlite3.connect(db)

c = conn.cursor()

n = 100

for i in range(n):

query1 = """ SELECT userid, AVG(longitude), AVG(latitude)

FROM Tweets, Geo

WHERE Tweets.id\_str = Geo.id\_str

GROUP BY userid; """

x = c.execute(query1).fetchall()

print (x)

toc = time.perf\_counter()

FunctionTime = toc - tic

print('This took {} seconds'.format(FunctionTime))

createSchema('dsc450\_final1.db')

A picture containing text, plaque

Description automatically generated

1. Write the equivalent of the 2-a query in python (without using SQL) by reading it from the file with 600,000 tweets.

1 time:

﻿import json

import time

def createSchema(textfile):

tic = time.perf\_counter()

with open(textfile, 'r') as file:

newDictLat = {}

newDictLong = {}

for line in file:

ln = file.readline()

tweet = json.loads(ln)

if tweet['geo']:

userid = str(tweet['user']['id'])

latlong = tweet['geo']['coordinates']

lat = latlong[0]

long = latlong[1]

if userid not in newDictLat:

count = 1

newDictLat[userid] = [count, lat]

newDictLong[userid] = [count, long]

else:

newDictLat[userid][0] += newDictLat[userid][0] + 1

newDictLong[userid][0] += newDictLong[userid][0] + 1

newDictLat[userid][1] += newDictLat[userid][1] + lat

newDictLong[userid][1] += newDictLong[userid][1] + long

avgLat = {}

avgLong = {}

for key, value in newDictLat.items():

avgLat[key] = value[1]/value[0]

for key, value in newDictLong.items():

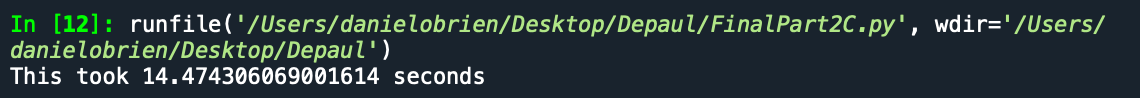
avgLong[key] = value[1]/value[0]

toc = time.perf\_counter()

FunctionTime = toc - tic

print('This took {} seconds'.format(FunctionTime))

createSchema('output\_final.txt')



1. Re-execute the query in part 2-c 10 times and 100 times and measure the total runtime. Does the runtime scale linearly?

When I ran this code 1 time it took about 14.5 seconds as can be seen from the output above. When I ran this code 10 and 100 times, it took about 145.75 seconds and about 1499.24 seconds respectively. Since the runtime for 10 and 100 times is almost exactly 10 and 100 times greater than the time it took to run once, I would say that the runtime does scale linearly.

10 times :

﻿import json

import time

def createSchema(textfile):

tic = time.perf\_counter()

n = 10

for x in range(n):

with open(textfile, 'r') as file:

newDictLat = {}

newDictLong = {}

for line in file:

ln = file.readline()

tweet = json.loads(ln)

if tweet['geo']:

userid = str(tweet['user']['id'])

latlong = tweet['geo']['coordinates']

lat = latlong[0]

long = latlong[1]

if userid not in newDictLat:

count = 1

newDictLat[userid] = [count, lat]

newDictLong[userid] = [count, long]

else:

newDictLat[userid][0] += newDictLat[userid][0] + 1

newDictLong[userid][0] += newDictLong[userid][0] + 1

newDictLat[userid][1] += newDictLat[userid][1] + lat

newDictLong[userid][1] += newDictLong[userid][1] + long

avgLat = {}

avgLong = {}

for key, value in newDictLat.items():

avgLat[key] = value[1]/value[0]

for key, value in newDictLong.items():

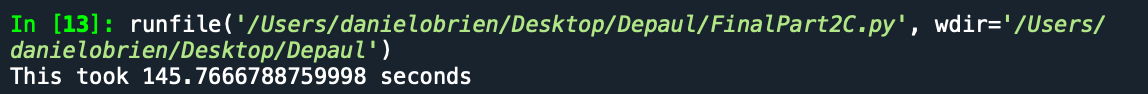
avgLong[key] = value[1]/value[0]

toc = time.perf\_counter()

FunctionTime = toc - tic

print('This took {} seconds'.format(FunctionTime))

createSchema('output\_final.txt')



100 Times:

﻿import json

import time

def createSchema(textfile):

tic = time.perf\_counter()

n = 100

for x in range(n):

with open(textfile, 'r') as file:

newDictLat = {}

newDictLong = {}

for line in file:

ln = file.readline()

tweet = json.loads(ln)

if tweet['geo']:

userid = str(tweet['user']['id'])

latlong = tweet['geo']['coordinates']

lat = latlong[0]

long = latlong[1]

if userid not in newDictLat:

count = 1

newDictLat[userid] = [count, lat]

newDictLong[userid] = [count, long]

else:

newDictLat[userid][0] += newDictLat[userid][0] + 1

newDictLong[userid][0] += newDictLong[userid][0] + 1

newDictLat[userid][1] += newDictLat[userid][1] + lat

newDictLong[userid][1] += newDictLong[userid][1] + long

avgLat = {}

avgLong = {}

for key, value in newDictLat.items():

avgLat[key] = value[1]/value[0]

for key, value in newDictLong.items():

avgLong[key] = value[1]/value[0]

toc = time.perf\_counter()

FunctionTime = toc - tic

print('This took {} seconds'.format(FunctionTime))

createSchema('output\_final.txt')



1. 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.

﻿import time

import re

def createSchema(textfile):

tic = time.perf\_counter()

LatDict = {}

LongDict = {}

with open(textfile, 'r') as file:

for line in file:

ln = file.readline()

u = re.compile('"user":{"id":(\d+)')

userid = u.findall(ln)

Fuserid = float(userid[0])

c = re.compile('"type":"Point","coordinates":\D(-?\d+\.?\d+?)')

coordinate = c.findall(ln)

if coordinate:

if Fuserid not in LatDict:

count = 1

LatDict[Fuserid] = [count, float(coordinate[0])]

LongDict[Fuserid] = [count, float(coordinate[1])]

else:

LatDict[Fuserid][0] += LatDict[Fuserid][0] + 1

LongDict[Fuserid][0] += LongDict[Fuserid][0] + 1

LatDict[Fuserid][1] += LatDict[Fuserid][1] + float(coordinate[0])

LongDict[Fuserid][1] += LongDict[Fuserid][1] + float(coordinate[1])

avgLat = {}

avgLong = {}

for key, value in LatDict.items():

avgLat[key] = value[1]/value[0]

for key, value in LongDict.items():

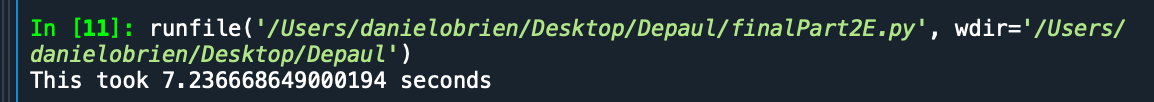
avgLong[key] = value[1]/value[0]

toc = time.perf\_counter()

FunctionTime = toc - tic

print('This took {} seconds'.format(FunctionTime))

createSchema('output\_final.txt')



1. Re-execute the query in part 2-e 10 times and 100 times and measure the total runtime. Does the runtime scale linearly?

When I ran this code once and it took about 7.24 seconds as can be seen from the output above. When I ran this code 10 times, it took about 69.69 seconds and when I ran it 100 times in 672.64 seconds. This is not exactly a linear scale, it seems to run slightly faster than a linear scale, but this may be due to other things running on my computer, it is not so much quicker per 10 and 100 times that I am ruling out a linear scale all together.

10 times:

﻿import time

import re

def createSchema(textfile):

tic = time.perf\_counter()

LatDict = {}

LongDict = {}

n = 10

for i in range(n):

with open(textfile, 'r') as file:

for line in file:

ln = file.readline()

u = re.compile('"user":{"id":(\d+)')

userid = u.findall(ln)

Fuserid = float(userid[0])

c = re.compile('"type":"Point","coordinates":\D(-?\d+\.?\d+?)')

coordinate = c.findall(ln)

if coordinate:

if Fuserid not in LatDict:

count = 1

LatDict[Fuserid] = [count, float(coordinate[0])]

LongDict[Fuserid] = [count, float(coordinate[1])]

else:

LatDict[Fuserid][0] += LatDict[Fuserid][0] + 1

LongDict[Fuserid][0] += LongDict[Fuserid][0] + 1

LatDict[Fuserid][1] += LatDict[Fuserid][1] + float(coordinate[0])

LongDict[Fuserid][1] += LongDict[Fuserid][1] + float(coordinate[1])

avgLat = {}

avgLong = {}

for key, value in LatDict.items():

avgLat[key] = value[1]/value[0]

for key, value in LongDict.items():

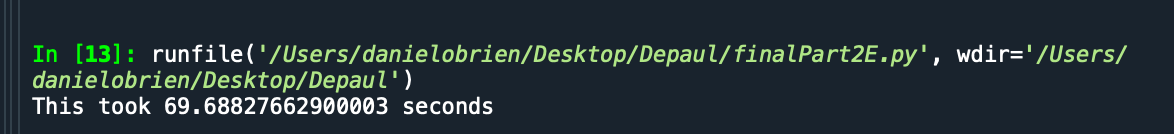
avgLong[key] = value[1]/value[0]

toc = time.perf\_counter()

FunctionTime = toc - tic

print('This took {} seconds'.format(FunctionTime))

createSchema('output\_final.txt')



100 times:

﻿import time

import re

def createSchema(textfile):

tic = time.perf\_counter()

LatDict = {}

LongDict = {}

n = 100

for i in range(n):

with open(textfile, 'r') as file:

for line in file:

ln = file.readline()

u = re.compile('"user":{"id":(\d+)')

userid = u.findall(ln)

Fuserid = float(userid[0])

c = re.compile('"type":"Point","coordinates":\D(-?\d+\.?\d+?)')

coordinate = c.findall(ln)

if coordinate:

if Fuserid not in LatDict:

count = 1

LatDict[Fuserid] = [count, float(coordinate[0])]

LongDict[Fuserid] = [count, float(coordinate[1])]

else:

LatDict[Fuserid][0] += LatDict[Fuserid][0] + 1

LongDict[Fuserid][0] += LongDict[Fuserid][0] + 1

LatDict[Fuserid][1] += LatDict[Fuserid][1] + float(coordinate[0])

LongDict[Fuserid][1] += LongDict[Fuserid][1] + float(coordinate[1])

avgLat = {}

avgLong = {}

for key, value in LatDict.items():

avgLat[key] = value[1]/value[0]

for key, value in LongDict.items():

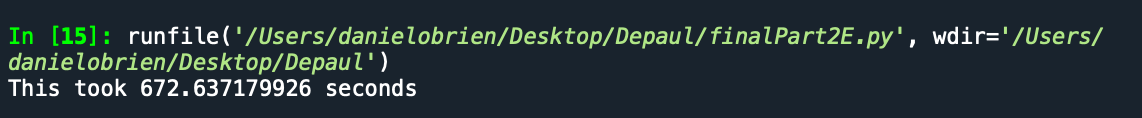
avgLong[key] = value[1]/value[0]

toc = time.perf\_counter()

FunctionTime = toc - tic

print('This took {} seconds'.format(FunctionTime))

createSchema('output\_final.txt')



**Part 3**

1. Using the database with 600,000 tweets, create a new table that corresponds to the pre-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).

﻿import sqlite3

import time

def createSchema(textfile):

tic = time.perf\_counter()

conn = sqlite3.connect('dsc450\_final1.db')

c = conn.cursor()

matview1 = """ CREATE TABLE matview2 AS

SELECT \*

FROM User

LEFT JOIN Tweets ON User.userid = Tweets.userid

LEFT JOIN Geo ON Tweets.id\_str = Geo.id\_str

; """

c.execute(matview1)

query1 = """ SELECT \*

FROM matview

LIMIT 10;"""

x = c.execute(query1).fetchall()

print(x)

toc = time.perf\_counter()

totalTime = toc - tic

print("this took {} seconds".format(totalTime))

createSchema('output\_final.txt')

1. Export the contents of your table from 3-a into a new JSON file. 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 present). How does the file size compare to the original input file?

The JSON file is much smaller than the original output file. The original output text file is about 1.9 gbs, while the json output file is about 630 mbs. So the JSON file is about a third of the size.

﻿import json

import sqlite3

import collections

conn = sqlite3.connect('dsc450\_final1.db')

c = conn.cursor()

cursor = c.execute("SELECT \* FROM matview2")

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

print(names, len(names))

rows = c.fetchall()

rowarray\_list = []

for row in rows:

t = (row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7], row[8], row[9], row[10], row[11], row[12], row[13], row[14], row[15], row[16], row[17], row[18], row[19])

rowarray\_list.append(t)

j = json.dumps(rowarray\_list)

with open ('jsonoutput.js', 'w') as file:

file.write(j)

objects\_list = []

for row in rows:

d = collections.OrderedDict()

d['tweetId'] = row[0]

d['name'] = row[1]

d['id\_str'] = row[2]

d['userid'] = row[3]

d['screen\_name'] = row[4]

d['description'] = row[5]

d['friend\_count'] = row[6]

d['created\_at'] = row[7]

d['id\_str1'] = row[8]

d['userid1'] = row[9]

d['text'] = row[10]

d['source'] = row[11]

d['in\_reply\_to\_user\_id'] = row[12]

d['in\_reply\_to\_screen\_name'] = row[13]

d['in\_reply\_to\_status\_id'] = row[14]

d['retweet\_count'] = row[15]

d['geo'] = row[16]

d['latitude'] = row[17]

d['longitude'] = row[18]

d['id\_str2'] = row[19]

objects\_list.append(d)

j = json.dumps(objects\_list)

with open("student\_objects.js", "w") as file:

file.write(j)

conn.close()

1. Export the contents of your table from 3-a into a .csv file. How does the file size compare to the original input file and to the file in 3-b?

The CSV file is much smaller than the original text file, and is also smaller than the JSON file. I was surprised to find that the JSON file was not the smallest. The CSV file is about 330 mbs, which is about half the size of the JSON file and about 1/6the size of the original text file.

﻿import sqlite3

import csv

conn = sqlite3.connect('dsc450\_final1.db')

c = conn.cursor()

c.execute("SELECT \* FROM matview2")

results = c.fetchall()

headers = [i[0] for i in c.description]

csvFile = csv.writer(open('csvOutout.csv', 'w', newline = ''),

delimiter =',', lineterminator='\r\n',

quoting=csv.QUOTE\_ALL, escapechar='\\')

csvFile.writerow(headers)

csvFile.writerows(results)

conn.close()