DSC 450: Database Processing for Large-Scale Analytics

Assignment Module 7

Charles Hanlon

Part 1

a. Write a function to generate a list of x random numbers, where x is the parameter indicating how many numbers to generate. Each generated number should randomly fall in the range between 27 and 100.

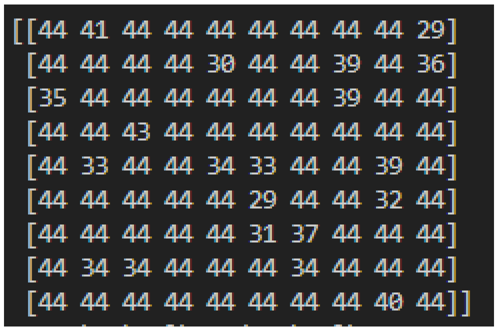
```
import random

def generateNums(x):
   res = []
   for i in range(x):
      res.append(random.randrange(27,100))

   return res
```

b. Use your function from 1-a to create a list of 90 random numbers with your code and use pandas.Series to determine how many of the numbers are below 44

```
import pandas

def analyzeNums(nums):
   series = pandas.Series(nums)
   res = len(series[series < 44])

   return res
```

c. Using the same list of 90 random numbers, 1) create a numpy array, modify it to 9x10 (you can do this by calling numpy.reshape(yourArray, (9,10)) and then replace all the numbers that are greater than or equal to 44 (44-100) by 44.

```
import numpy

def modifyNums(nums):
   arr = numpy.reshape(nums, (9,10))
   arr[arr >= 44] = 44

   return arr
```



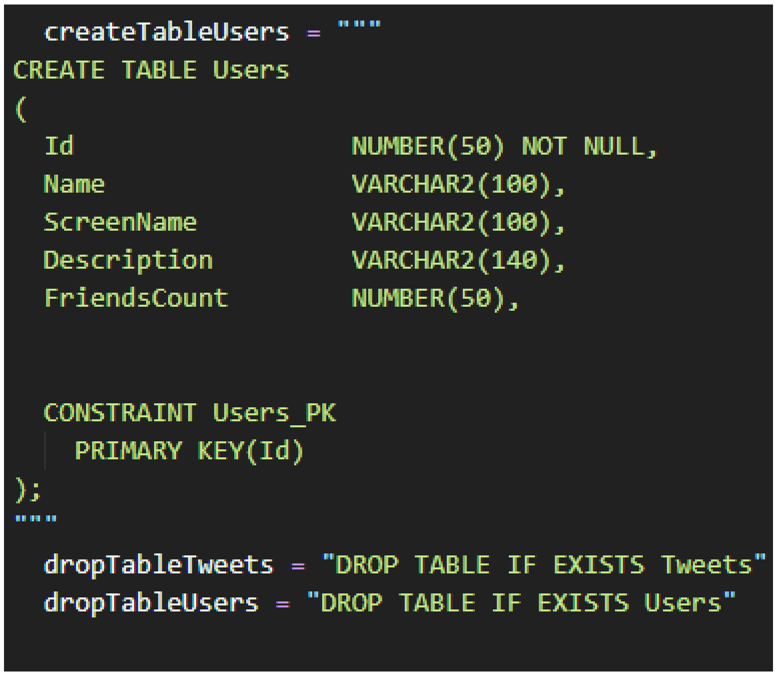
```
[[44 41 44 44 44 44 44 44 44 29]
 [44 44 44 44 30 44 44 39 44 36]
 [35 44 44 44 44 44 44 39 44 44]
 [44 44 43 44 44 44 44 44 44 44]
 [44 33 44 44 34 33 44 44 39 44]
 [44 44 44 44 44 29 44 44 32 44]
 [44 44 44 44 44 31 37 44 44 44]
 [44 34 34 44 44 44 34 44 44 44]
 [44 44 44 44 44 44 44 44 40 44]]
```

Part 2

In this part we are going to work with a larger collection of tweets (10,000) that are available here.
https://dbgroup.cdm.depaul.edu/DSC450/Module7.txt

The tweets are all on separate lines, but some of the tweets are intentionally damaged and will not parse properly. You will need to store these tweets in a separate "error" file. At the bottom of the page you can find python code that will let you skip over badly formed tweets.

a. Create a new SQL table for the user dictionary. It should contain the following attributes "id", "name", "screen_name", "description" and "friends_count". Modify your SQL table from Module 5 to include "user_id" columns which will be a foreign key referencing the user table.



```
  createTableUsers = """
CREATE TABLE Users
(
  Id                   NUMBER(50) NOT NULL,
  Name                 VARCHAR2(100),
  ScreenName           VARCHAR2(100),
  Description          VARCHAR2(140),
  FriendsCount         NUMBER(50),


  CONSTRAINT Users_PK
    PRIMARY KEY(Id)
);
"""
  dropTableTweets = "DROP TABLE IF EXISTS Tweets"
  dropTableUsers = "DROP TABLE IF EXISTS Users"
```

b. Write python code that is going to read and load the Module7.txt file directly from the web and populate both of your tables (Tweet table from Module5 and User table from this assignment). You can use the same code from the previous assignment with an additional step of inserting data into the newly created table.
For tweets that could not parse, write them into a Module7_errors.txt file.

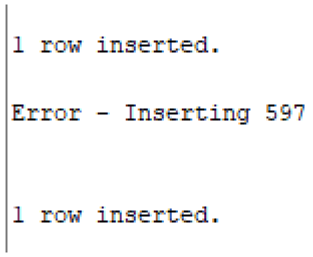
```
import sqlite3
import json
import urllib.request

def tweetStore():
  createTableTweets = """
 CREATE TABLE Tweets
 (
  CreatedOn          VARCHAR2(50),
  TweetId            VARCHAR2(50) NOT NULL,
  UserId             NUMBER(50) NOT NULL,
  Body               VARCHAR2(200),
  Source             VARCHAR2(200),
  ReplyToUser        VARCHAR2(50),
  ReplyToName        VARCHAR(100),
  ReplyToStatus      VARCHAR(100),
  Retweets           NUMBER(10),
  Contributors       VARCHAR(100),

  CONSTRAINT Tweets_PK
    PRIMARY KEY(TweetId),

  CONSTRAINT Tweets_FK
    FOREIGN KEY(UserId)
    REFERENCES User(Id)
 );
 """
  createTableUsers = """
 CREATE TABLE Users
 (
  Id                 NUMBER(50) NOT NULL,
  Name               VARCHAR2(100),
  ScreenName         VARCHAR2(100),
  Description        VARCHAR2(140),
  FriendsCount       NUMBER(50),


  CONSTRAINT Users_PK
    PRIMARY KEY(Id)
 );
 """
  dropTableTweets = "DROP TABLE IF EXISTS Tweets"
  dropTableUsers = "DROP TABLE IF EXISTS Users"
```

```
    conn = sqlite3.connect('7tweets.db') # open db conection
    cursor = conn.cursor()

    cursor.execute(dropTableTweets)
    cursor.execute(dropTableUsers)
    cursor.execute(createTableUsers)
    cursor.execute(createTableTweets)

    tweetData = 'https://dbgroup.cdm.depaul.edu/DSC450/Module7.txt'
    webFD = urllib.request.urlopen(tweetData)
    tweets = webFD.readlines()
    fd = open('errors.txt', 'w+', encoding="utf8")
    #tweets = data.split("        EndOfTweet         ")

    for tweet in tweets:
        try:
            tmp = json.loads(tweet.decode('utf8'))
            tmpDate = tmp["created_at"].split(' ')
            cleanDate = tmpDate[1] + ' ' + tmpDate[2]  + ' ' + tmpDate[5]
            currTweet = [cleanDate, tmp["id_str"], tmp['user']['id'], tmp["text"],
tmp["source"], tmp["in_reply_to_user_id"], tmp["in_reply_to_screen_name"],
tmp["in_reply_to_status_id"], tmp["retweet_count"], tmp["contributors"]]
            currUser = [tmp['user']['id'], tmp['user']['name'], tmp['user']['screen_name'],
tmp['user']['description'], tmp['user']['friends_count']]
            cursor.execute("INSERT OR IGNORE INTO Tweets VALUES(?,?,?,?,?,?,?,?,?,?)",
currTweet)
            cursor.execute("INSERT OR IGNORE INTO Users VALUES(?,?,?,?,?)", currUser)
        except ValueError:
            fd.write("Error: " + str(tweet) + "\n")

    fd.close()

tweetStore()
```

Part 3

a. Write a PL/SQL trigger that will cap the course number column in the university.sql database at 597. That is, any time an update or an insert would provide course number 598 or higher, automatically reset the value back to 597. Be sure to verify that your trigger is working with some sample data.

```
CREATE OR REPLACE TRIGGER CourseNumberCap
BEFORE INSERT OR UPDATE OF CourseNr ON course
FOR EACH ROW
WHEN (new.CourseNr > 597)
BEGIN
  :new.CourseNr := '597';
  DBMS_OUTPUT.PUT_Line('Error - Inserting 597');
END;
/
```



```
1 row inserted.

Error - Inserting 597


1 row inserted.
```

b. Write a regular expression to match credit card numbers followed by a space and a month/year pattern (e.g., 2/23 or 02/23) assuming a 16-digit credit card that may or may not include dashes (-) after each group of 4 digits. Create the code to validate that your regular expression works in either python or Oracle.

```
import re
def ccValidate():
  regex = "(?:\d{4}[-\s]?){3}\d{4}\s(?:0?[1-9]|1[0-2])/(?=\d{2}$)\d{2}"
  cc1 = '1234 1234 1234 1234 01/23'
  cc2 = '12345 1234 1234 1234 01/23'
  cc3 = '1345-1234-1234-1234 11/20'
  cc4 = '1345 1234 1234-1234 11/20'
  cc5 = '1345-1234-1234-1234 12/21'
  for test in tests:
    if re.match(regex, test):
      print(test + " is valid!")
    else:
      print(test + " is NOT valid")

ccValidate()
```