Syed Noor Razi Ali
2070326

DSC 450: Database Processing for Large-Scale Analytics
Assignment Module 7

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 33 and 100.

In [43]:
```
import random
from unicodedata import name
import pandas
import numpy
```

In [47]:
```
def random_numbers(x):
    mylist = []
    i=0
    while i < x:
        y = random.randint(33, 100)
        mylist.append(y)
        i+=1
    return mylist

random_numbers(22)
```

Out[47]:
```
[46,
 36,
 58,
 92,
 69,
 66,
 58,
 33,
 73,
 38,
 81,
 45,
 62,
 57,
 88,
 38,
 81,
 68,
 93,
 43,
 81,
 91]
```

b. Use your function from 1-a to create a list of 70 random numbers with your code and use pandas.Series to determine how many of the numbers are below 37

In [52]:
```
def random_numbers(x):
    mylist = []
    i=0
    while i < x:
        y = random.randint(33, 100)
        mylist.append(y)
        i+=1
    return mylist

num = random_numbers(22)
num
```

Out[52]:
```
[55,
 75,
 57,
 38,
 58,
 95,
 38,
 84,
 72,
 71,
 70,
 99,
 71,
 68,
 54,
 55,
 87,
 88,
 47,
 36,
 56,
 82]
```

In [53]:
```
y = len(pandas.Series(num)[pandas.Series(num) < 37])

print("\nNumbers below 37:", y, "\n")
```

```
Numbers below 37: 1
```

c. Using the same list of 70 random numbers, 1) create a numpy array, modify it to 7x10 (you can do this by calling numpy.reshape(yourArray, (7,10)) and then replace all the numbers that are greater than or equal to 50 (50-100) by 50.

```
    while i < x:
        y = random.randint(33, 100)
        mylist.append(y)
        i+=1
    return mylist

num = random_numbers(22)
num
```

Out[13]:
```
[75,
 82,
 39,
 92,
 58,
 52,
 76,
 95,
 44,
 75,
 34,
 36,
 93,
 47,
 80,
 87,
 98,
 80,
 88,
 90,
 47,
 36]
```

In [14]:
```
y = len(pandas.Series(num)[pandas.Series(num) < 37])

print("\nNumbers below 37:", y, "\n")
```

```
Numbers below 37: 3
```

In [15]:
```
# reshape into a numpy array
new_array = numpy.reshape(num, (7, 10))
new_array[new_array >= 50] = 50
print("\n Replaced ARRAY \n ", new_array)

new_array
```

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.

In [18]:
```
import urllib.request
import json
import sqlite3
import ssl


ssl._create_default_https_context = ssl._create_unverified_context
Create_TBL_users = """
CREATE TABLE Users(
    id VARCHAR2(40),
    name VARCHAR2(40),
    screen_name VARCHAR2(40),
    description VARCHAR2(40),
    friends_count INTEGER,

    CONSTRAINT id_PK
        PRIMARY KEY (id)
)
"""
Create_TBL_tweets = """
CREATE TABLE Tweet(
    created_at VARCHAR2(50),
    id_str VARCHAR2(65),
    text VARCHAR(280),
    source VARCHAR(100),
    in_reply_to_user_id VARCHAR2(65),
    in_reply_to_screen_name VARCHAR2(100),
    in_reply_to_status_id NUMBER(65),
    retweet_count NUMBER(10),
    contributors VARCHAR2(10),
    user_id VARCHAR2(38),

    CONSTRAINT tweets_PK
        PRIMARY KEY (id_str)

    CONSTRAINT tweets_PK
        FOREIGN KEY (user_id)
            REFERENCES users(id)
);
"""
conn = sqlite3.connect('dsc450.db')  # open the connection
cursor = conn.cursor()

# drop tables if existing
cursor.execute('DROP TABLE IF EXISTS Users;')
cursor.execute('DROP TABLE IF EXISTS Tweet;')

# create tables users and tweets
cursor.execute(Create_TBL_users)
cursor.execute(Create_TBL_tweets)
```

Out[18]:
```
<sqlite3.Cursor at 0x26f00b1ae30>
```

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.

You can gracefully catch JSON errors using the following code:

```
for tweet in allTweets:
  try:
    tdict = json.loads(tweet.decode('utf8'))
  except ValueError:
    # Handle the problematic tweet, which in your case would require writing it to another file
    print (tweet)
```

In [19]:

```
tweet_Data = urllib.request.urlopen("https://dbgroup.cdm.depaul.edu/DSC450/Module7.txt")

tweets = []
errors = []


for raw in tweet_Data:
    try:
        tweets.append(json.loads(raw.decode("utf8")))
    except:
        errors.append(raw)

# writing errors to a text file
f = open("errors.txt", "w")
for er in errors:
    f.write(str(er))
'''for r in tweets:
    print(r)'''
for twe in tweets:
    tweets = twe

    cursor.execute("INSERT OR IGNORE INTO Users VALUES (?, ?, ?, ?, ?)",
                   (tweets['user']['id'],
                    tweets['user']['name'],
                    tweets['user']['screen_name'],
                    tweets['user']['description'],
                    tweets['user']['friends_count']))
    cursor.execute("INSERT OR IGNORE INTO Tweet VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?,?)",
                   (tweets['created_at'],
                    tweets['id_str'],
                    tweets['text'],
                    tweets['source'],
                    tweets['in_reply_to_user_id'],
                    tweets['in_reply_to_screen_name'],
                    tweets['in_reply_to_status_id'],
                    tweets['retweet_count'],
                    tweets['contributors'],
                    tweets['user']['id']))

cursor.execute("SELECT * FROM Users")

rows = cursor.fetchone()

print("\n1st row of Users table:\n")
for row in rows:
    print(row)

cursor.execute("SELECT * FROM Tweet")

rows = cursor.fetchone()

print("\n1st row of Tweet table:\n")
for row in rows:
    print(row)
```

```
1st row of Users table:

367361405
gugusuarez
guadalupesuare4
None
543

1st row of Tweet table:

Tue May 20 00:00:19 +0000 2014
468541694288207874
la asusto a selena me dice es joda te vy a extrañar jajajajaja ni m fui pero ta vy a tener tiempo libre y todo wi
<a href="https://mobile.twitter.com" rel="nofollow">Mobile Web (M2)</a>
None
None
None
0
None
367361405
```

Part 3

a. Write a PL/SQL trigger that will cap the course number column in the university.sql database at 598. That is, any time an update or an insert would provide course number 599 or higher, automatically reset the value back to 598. Be sure to verify that your trigger is working with some sample data.

```
CREATE OR REPLACE TRIGGER Valid_courseNr
BEFORE INSERT OR UPDATE ON Course
FOR EACH ROW
WHEN(new.CourseNr > 598)
BEGIN
 :new.CourseNr := 598;
 dbms_output.put_line('Set CourseNr to 598');
END;
/
set serveroutput on;

insert into course values ( 1333, 'NeuralNetwork', 'CSC', 855);
select * from course;
```

Script Output | Query Result

SQL | All Rows Fetched: 8 in 0.022 seconds

	CID	COURSENAME	DEPARTMENT	COURSENR
1	1020	Theory of Computation	CSC	489
2	1092	Cryptography	CSC	440
3	3201	Data Analysis	IT	223
4	9219	Desktop Databases	IT	240
5	3111	Theory of Computation	CSC	389
6	8772	Survey of Computer Graphics	GPH	425
7	2987	Topics in Digital Cinema	DC	270
8	1333	NeuralNetwork	CSC	598

b. Write a regular expression to match credit card numbers, assuming a 16-digit credit card that may or may not include spaces after each 4 digits. Create the code to validate that your regular expression works in either python or Oracle.

```
Drop Table CreditCard ;
create table CreditCard
( card_no varchar2(20),

 CONSTRAINT card_no_check
 check
 ( REGEXP_LIKE (card_no, '(\d{4} ?\d{4} ?\d{4} ?\d{4})' )));
INSERT into creditcard values('4125 6598 3003 4565') ;
INSERT into creditcard values('4125-6598-3003-4565') ;
INSERT into creditcard values('4125659830034565') ;
INSERT into creditcard values('9845321565478569') ;
select * from Creditcard;
```

Script Output | Query Result | Query Result 1 | Query Result 2

Task completed in 0.272 seconds

```
Table CREDITCARD dropped.


Table CREDITCARD created.


1 row inserted.


Error starting at line : 169 in command -
INSERT into creditcard values('4125-6598-3003-4565')
Error report -
ORA-02290: check constraint (NSYED17.CARD_NO_CHECK) violated


1 row inserted.


1 row inserted.
```

```
INSERT into creditcard values('9845321565478569') ;
select * from Creditcard;
```

Script Output | Query Result | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 3 in 0.02 seconds

	CARD_NO
1	4125 6598 3003 4565
2	4125659830034565
3	9845321565478569