Project_1B_ Project_Template

March 10, 2022

1 Part I. ETL Pipeline for Pre-Processing the Files

1.1 PLEASE RUN THE FOLLOWING CODE FOR PRE-PROCESSING THE FILES

Import Python packages

```
In [1]: # Import Python packages
        import pandas as pd
        import cassandra
        import re
        import os
        import glob
        import numpy as np
        import json
        import csv
```

Creating list of filepaths to process original event csv data files

```
In [2]: # checking your current working directory
        print(os.getcwd())

        # Get your current folder and subfolder event data
        filepath = os.getcwd() + '/event_data'

        # Create a for loop to create a list of files and collect each filepath
        for root, dirs, files in os.walk(filepath):

        # join the file path and roots with the subdirectories using glob
            file_path_list = glob.glob(os.path.join(root,'*'))
            #print(file_path_list)
```

```
/home/workspace
```

Processing the files to create the data file csv that will be used for Apache Casssandra tables

```
In [3]: # initiating an empty list of rows that will be generated from each file
        full_data_rows_list = []
```

```
        # for every filepath in the file path list
        for f in file_path_list:

        # reading csv file
            with open(f, 'r', encoding = 'utf8', newline='') as csvfile:
                # creating a csv reader object
                csvreader = csv.reader(csvfile)
                next(csvreader)

         # extracting each data row one by one and append it
                for line in csvreader:
                    #print(line)
                    full_data_rows_list.append(line)

        # uncomment the code below if you would like to get total number of rows
        #print(len(full_data_rows_list))
        # uncomment the code below if you would like to check to see what the list of event data
        #print(full_data_rows_list)

        # creating a smaller event data csv file called event_datafile_full csv that will be use
        # Apache Cassandra tables
        csv.register_dialect('myDialect', quoting=csv.QUOTE_ALL, skipinitialspace=True)

        with open('event_datafile_new.csv', 'w', encoding = 'utf8', newline='') as f:
            writer = csv.writer(f, dialect='myDialect')
            writer.writerow(['artist','firstName','gender','itemInSession','lastName','length',\
                        'level','location','sessionId','song','userId'])
            for row in full_data_rows_list:
                if (row[0] == ''):
                    continue
                writer.writerow((row[0], row[2], row[3], row[4], row[5], row[6], row[7], row[8],
```

In [4]:
```
        # check the number of rows in your csv file
        with open('event_datafile_new.csv', 'r', encoding = 'utf8') as f:
            print(sum(1 for line in f))
```

```
6821
```

2 Part II. Complete the Apache Cassandra coding portion of your project.

2.1 Now you are ready to work with the CSV file titled event_datafile_new.csv, located within the Workspace directory. The event_datafile_new.csv contains the following columns:

- artist

- firstName of user
- gender of user
- item number in session
- last name of user
- length of the song
- level (paid or free song)
- location of the user
- sessionId
- song title
- userId

The image below is a screenshot of what the denormalized data should appear like in the **event_datafile_new.csv** after the code above is run:

2.2 Begin writing your Apache Cassandra code in the cells below

Creating a Cluster

```
In [5]: # This should make a connection to a Cassandra instance your local machine
        # (127.0.0.1)

        from cassandra.cluster import Cluster
        cluster = Cluster(['127.0.0.1'])

        # To establish connection and begin executing queries, need a session
        session = cluster.connect()
```

Create Keyspace

```
In [6]: # Create a Keyspace
        try:
            session.execute("""
            CREATE KEYSPACE IF NOT EXISTS sparkifydb
            WITH REPLICATION =
            { 'class' : 'SimpleStrategy', 'replication_factor' : 1 }""")
        )

        except Exception as e:
            print(e)
```

Set Keyspace

```
In [7]: # Set KEYSPACE to the keyspace specified above
        try:
            session.set_keyspace('sparkifydb')
        except Exception as e:
            print(e)
```

2.2.1 Now we need to create tables to run the following queries. Remember, with Apache Cassandra you model the database tables on the queries you want to run.

2.3 Create queries to ask the following three questions of the data

2.3.1 1. Give me the artist, song title and song's length in the music app history that was heard during sessionId = 338, and itemInSession = 4

2.3.2 2. Give me only the following: name of artist, song (sorted by itemInSession) and user (first and last name) for userid = 10, sessionid = 182

2.3.3 3. Give me every user name (first and last) in my music app history who listened to the song 'All Hands Against His Own'

In [8]:
```
## Query 1:  Give me the artist, song title and song's length in the music app history t
## sessionId = 338, and itemInSession = 4
## Description: Here the Primary Key has two fields: sessionId is the partition key, and
## Partitioning is done by sessionId and within that partition, rows are ordered by the
query = "CREATE TABLE IF NOT EXISTS song_details "
query = query + "(sessionId INT, itemInSession INT, artist TEXT, song TEXT, length FLOAT
try:
    session.execute(query)
except Exception as e:
    print(e)
```

In [9]:
```
# We have provided part of the code to set up the CSV file. Please complete the Apache C
file = 'event_datafile_new.csv'

with open(file, encoding = 'utf8') as f:
    csvreader = csv.reader(f)
    next(csvreader) # skip header
    for line in csvreader:
## TO-DO: Assign the INSERT statements into the `query` variable
        query = "INSERT INTO song_details(sessionId, itemInSession, artist, song, length
        query = query + "VALUES (%s, %s, %s, %s, %s)"
        ## Assign which column element should be assigned for each column in the INSERT
        ## For e.g., to INSERT artist_name and user first_name, you would change the cod
        session.execute(query, (int(line[8]), int(line[3]), line[0], line[9], float(line
```

SELECT query to output the data have been inserted into each table

In [10]:
```
## SELECT statement returns the artist name, song title and song length as asked in the
select_query_1 = """ SELECT artist, song, length FROM song_details
                     WHERE sessionId = %s
                     AND itemInSession = %s
                     """

try:
```

```
        rows = session.execute(select_query_1, (338, 4))

        for row in rows:
            print("Artist: "+row.artist, "\nSong Title: "+row.song, "\nSong Length: "+str(r

    except Exception as e:
        print(e)
```

```
Artist: Faithless
Song Title: Music Matters (Mark Knight Dub)
Song Length: 495.30731201171875
```

```
In [11]: ## Query 2: Give me only the following: name of artist, song (sorted by itemInSession)
         ## for userid = 10, sessionid = 182
         ## Description: Here the Primary Key has two fields: userid, sessionId are the partitio
         ## Partitioning is done by userid, sessionId and within that partition, rows are ordere
         query = "CREATE TABLE IF NOT EXISTS aritst_details "
         query = query + "(userid INT, sessionId INT, itemInSession INT, artist TEXT, song TEXT,
         try:
             session.execute(query)
         except Exception as e:
             print(e)
```

```
In [12]: # We have provided part of the code to set up the CSV file. Please complete the Apache
         file = 'event_datafile_new.csv'

         with open(file, encoding = 'utf8') as f:
             csvreader = csv.reader(f)
             next(csvreader) # skip header
             for line in csvreader:
                 #print(line)
         ## Assign the INSERT statements into the `query` variable
                 query = "INSERT INTO aritst_details(userid, sessionId, itemInSession, artist, s
                 query = query + "VALUES (%s, %s, %s, %s, %s, %s, %s)"
                 #print(int(line[10]), int(line[8]), int(line[3]), line[0], line[9], line[1], li
                 ## Assign which column element should be assigned for each column in the INSERT
                 ## For e.g., to INSERT artist_name and user first_name, you would change the co
                 session.execute(query, (int(line[10]), int(line[8]), int(line[3]), line[0], lin
```

```
In [15]: ## SELECT statement returns the artist name, song title,song length, user first name an
         select_query_2 = """ SELECT artist, song, firstName, lastName
                              FROM aritst_details
                              WHERE userid = %s
                              AND sessionId = %s
                              """
```

```
        try:
            rows = session.execute(select_query_2, (10, 182))

            for row in rows:
                print("Artist: "+row.artist, "\nSong Title: "+row.song, "\nUser First Name: "+s

        except Exception as e:
            print(e)
```

```
Artist: Down To The Bone
Song Title: Keep On Keepin' On
User First Name: Sylvie
User Last Name: Cruz
Artist: Three Drives
Song Title: Greece 2000
User First Name: Sylvie
User Last Name: Cruz
Artist: Sebastien Tellier
Song Title: Kilometer
User First Name: Sylvie
User Last Name: Cruz
Artist: Lonnie Gordon
Song Title: Catch You Baby (Steve Pitron & Max Sanna Radio Edit)
User First Name: Sylvie
User Last Name: Cruz
```

```
In [16]: ## Query 3: Give me every user name (first and last) in my music app history who listen
         ## Description: Here the Primary Key has two fields: song is the partition key, and use
         ## Partitioning is done by song and within that partition, rows are ordered by the user

         query = "CREATE TABLE IF NOT EXISTS user_details "
         query = query + "(song TEXT, userid INT, firstName TEXT, lastName TEXT, PRIMARY KEY ((s
         try:
             session.execute(query)
         except Exception as e:
             print(e)
```

```
In [17]: # We have provided part of the code to set up the CSV file. Please complete the Apache
         file = 'event_datafile_new.csv'

         with open(file, encoding = 'utf8') as f:
             csvreader = csv.reader(f)
             next(csvreader) # skip header
             for line in csvreader:
                 #print(line)
```

```
        ## Assign the INSERT statements into the `query` variable
                query = "INSERT INTO user_details(song, userid, firstname, lastname)"
                query = query + "VALUES (%s, %s, %s, %s)"
                #print(line[9],':', int(line[10]), line[1],'-', line[4])
                ## Assign which column element should be assigned for each column in the INSERT
                ## For e.g., to INSERT artist_name and user first_name, you would change the co
                session.execute(query, (line[9], int(line[10]), line[1], line[4]))
```

```
In [19]: ## SELECT statement returns the user first name and user last name as asked in the ques

         select_query_3 = """ SELECT firstname, lastname
                              FROM user_details
                              WHERE song = %s
                              """

         try:
             rows = session.execute(select_query_3, ('All Hands Against His Own', ))

             for row in rows:
                 print("User First Name: "+row.firstname, "\nUser Second Name: "+row.lastname)

         except Exception as e:
             print(e)
```

```
User First Name: Jacqueline
User Second Name: Lynch
User First Name: Tegan
User Second Name: Levine
User First Name: Sara
User Second Name: Johnson
```

2.3.4 Drop the tables before closing out the sessions

```
In [20]: ## Drop the table before closing out the sessions
         session.execute("DROP TABLE IF EXISTS song_details")
         session.execute("DROP TABLE IF EXISTS artist_details")
         session.execute("DROP TABLE IF EXISTS user_details")
```

```
Out[20]: <cassandra.cluster.ResultSet at 0x7fbc3a2c6358>
```

2.3.5 Close the session and cluster connectionű

```
In [21]: session.shutdown()
         cluster.shutdown()
```