Project_1B_ Project_Template

October 11, 2020

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 [2]: # 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 [3]: # checking your current working directory
        print(f"Current working directory: {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)
```

```
Current working directory: /home/workspace
```

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

```
In [4]: # 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(f"Total number of rows of event_data: {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],

Total number of rows of event_data: 8056


In [5]: # 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 [6]: # 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 [7]: # TO-DO: Create a Keyspace
        try:
            session.execute("""
                CREATE KEYSPACE IF NOT EXISTS sparkifydb
                WITH REPLICATION = {
                    'class': 'SimpleStrategy',
                    'replication_factor': 1
                }
            """)
        except Exception as e:
            print("Issue on creating a Keyspace")
            print(e)
```

Set Keyspace

```
In [8]: # TO-DO: Set KEYSPACE to the keyspace specified above
        try:
            session.set_keyspace('sparkifydb')
        except Exception as e:
            print("Failed to set keyspace")
            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 [27]: ## TO-DO: Query 1:  Give me the artist, song title and song's length in the music app h
         ## sessionId = 338, and itemInSession = 4

         create_table_query1 = """
                                 CREATE TABLE IF NOT EXISTS song_library
                                 (
                                     artist TEXT,
                                     itemInSession INT,
                                     length TEXT,
                                     sessionId INT,
                                     song TEXT,
                                     PRIMARY KEY (sessionId, itemInSession)
                                 );
                                 """
         try:
             session.execute(create_table_query1)
             print("Create song_library table completed")
         except Exception as e:
             print('Failed to create a table for query1')
             print(e)
```

```
Create song_library table completed
```

```
In [28]: # 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:
                query = "INSERT INTO song_library (artist, itemInSession, length, sessionId, so
                query = query + " VALUES (%s, %s, %s, %s, %s);"
                session.execute(query, (line[0], int(line[3]), line[5], int(line[8]), line[9]))

        print("Loading data into song_library table completed")
```

```
Loading data into song_library table completed
```

Do a SELECT to verify that the data have been inserted into each table

```
In [29]: ## TO-DO: Add in the SELECT statement to verify the data was entered into the table
         query1 = """
                     SELECT artist, song, length
                     FROM song_library
                     WHERE sessionId=338
                     AND itemInSession=4;
                  """
         try:
             rows = session.execute(query1)
             print("Finished querying...")
         except Exception as e:
             print(e)

         print("Starting reading...")
         for row in rows:
             print(f"Artist: {row.artist}, Song: {row.song}, Length: {row.length}")
```

```
Finished querying...
Starting reading...
Artist: Faithless, Song: Music Matters (Mark Knight Dub), Length: 495.3073
```

2.3.4 COPY AND REPEAT THE ABOVE THREE CELLS FOR EACH OF THE THREE QUESTIONS

```
In [30]: ## TO-DO: Query 2: Give me only the following: name of artist, song (sorted by itemInSe
         ## for userid = 10, sessionid = 182
         create_table_query2 = """
                                   CREATE TABLE IF NOT EXISTS song_listened_by_user_session
                                   (
                                       artist text,
                                       firstName text,
                                itemInSession int,
                                length text,
                                level text,
                                sessionId int,
                                song text,
                                userId int,
                                PRIMARY KEY ((userId, sessionId), itemInSession)
                            );
                            """
        try:
            session.execute(create_table_query2)
            print("Create new create_table_query2 table completed...")
        except Exception as e:
            print('Failed to create a table for query2')
            print(e)


Create new create_table_query2 table completed...
```

```
In [31]: # INSERT data into the song_listened_by_user_session
         # 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:
                 query = "INSERT INTO song_listened_by_user_session (artist, firstName, itemInSe
                         level, sessionId, song, userId)"
                 query = query + " VALUES (%s, %s, %s, %s, %s, %s, %s, %s);"
                 session.execute(query, (line[0], line[1], int(line[3]), line[5], line[6], int(l

         print("Load data into Cassandra completed...")

Load data into Cassandra completed...
```

```
In [32]: ## Add in the SELECT statement to verify the data was entered into the song_listened_by
         ## Query 2: Give me only the following: name of artist, song (sorted by itemInSession)
         ## for userid = 10, sessionid = 182
         query2 = """
                     SELECT artist, song, firstName, itemInSession
                     FROM song_listened_by_user_session
                     WHERE userId=10 AND sessionId=182
                     ORDER BY itemInSession DESC;
                 """
         try:
             rows = session.execute(query2)
```

```
            print("completed querying...")
        except Exception as e:
            print(e)

        print("start printing...")
        for row in rows:
            print(f"Artist: {row.artist}, Song: {row.song}, User: {row.firstname}, No of Items:
```

```
completed querying...
start printing...
Artist: Lonnie Gordon, Song: Catch You Baby (Steve Pitron & Max Sanna Radio Edit), User: Sylvie,
Artist: Sebastien Tellier, Song: Kilometer, User: Sylvie, No of Items: 2
Artist: Three Drives, Song: Greece 2000, User: Sylvie, No of Items: 1
Artist: Down To The Bone, Song: Keep On Keepin' On, User: Sylvie, No of Items: 0
```

```
In [33]: ## TO-DO: Query 3: Give me every user name (first and last) in my music app history who

         create_table_query3 = """
                             CREATE TABLE IF NOT EXISTS user_on_specific_song
                             (
                                 firstName TEXT,
                                 lastName TEXT,
                                 song TEXT,
                                 userId INT,
                                 PRIMARY KEY ((song), firstName)
                             );
                             """
         try:
             session.execute(create_table_query3)
             print("Create new user_on_specific_song table completed...")
         except Exception as e:
             print('Failed to create a table for query3')
             print(e)
```

```
Create new user_on_specific_song table completed...
```

```
In [34]: # INSERT data into the user_on_specific_song
         # 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:
                 query = "INSERT INTO user_on_specific_song (firstName, lastName, song, userId)"
                 query = query + " VALUES (%s, %s, %s, %s);"
```

```
                  session.execute(query, (line[1], line[4], line[9], int(line[10])))

        print("Load data into Cassandra completed...")
```

```
Load data into Cassandra completed...
```

```
In [20]: ## Add in the SELECT statement to verify the data was entered into the song_listened_by
         ## TO-DO: Query 3: Give me every user name (first and last) in my music app history who
         query3 = """
                     SELECT firstName, lastName, song, userId
                     FROM user_on_specific_song
                     WHERE song='All Hands Against His Own';
                  """
         try:
             rows = session.execute(query3)
             print("completed querying...")
         except Exception as e:
             print(e)

         print("start printing...")
         for row in rows:
             print(f"First Name: {row.firstname}, Last Name: {row.lastname}, User ID:{row.userid
```

```
completed querying...
start printing...
First Name: Jacqueline, Last Name: Lynch, User ID:29, Song:  All Hands Against His Own
First Name: Sara, Last Name: Johnson, User ID:95, Song:  All Hands Against His Own
First Name: Tegan, Last Name: Levine, User ID:80, Song:  All Hands Against His Own
```

2.3.5 Drop the tables before closing out the sessions

```
In [4]: ## TO-DO: Drop the table before closing out the sessions

In [26]: try:
             session.execute("DROP TABLE song_library;")
             print("Dropping song_library table completed")
         except Exception as e:
             print('Failed to drop `song_library` table')
             print(e)
```

```
Dropping song_library table completed
```

```
In [12]: try:
             session.execute("DROP TABLE song_listened_by_user_session;")
             print("Dropping song_listened_by_user_session table completed")
         except Exception as e:
             print('Failed to drop `song_listened_by_user_session`table')
             print(e)
```

```
Dropping song_listened_by_user_session table completed
```

```
In [17]: try:
             session.execute("DROP TABLE user_on_specific_song;")
             print("Dropping user_on_specific_song table completed")
         except Exception as e:
             print('Failed to drop `song_listened_by_user_session`table')
             print(e)
```

```
Dropping user_on_specific_song table completed
```

2.3.6 Close the session and cluster connectionű

```
In [35]: session.shutdown()
         cluster.shutdown()
```

```
In [ ]:
```

```
In [ ]:
```