Project_1B_ Project_Template

January 3, 2025

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 [18]: # 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 [19]: # 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)
```

```
/workspace/home
```

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

```
In [20]: # 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 dat
        # print(full_data_rows_list)

        # creating a smaller event data csv file called event_datafile_full csv that will be us
        # 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]
```

```
8056
```

```
In [21]: # 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 [22]: # This should make a connection to a Cassandra instance your local machine
         # (127.0.0.1)

         from cassandra.cluster import Cluster
         cluster = Cluster()

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

Create Keyspace

```
In [23]: # TO-DO: Create a Keyspace

         try:
             session.execute("""
             CREATE KEYSPACE IF NOT EXISTS song_library
             WITH REPLICATION =
             { 'class' : 'SimpleStrategy', 'replication_factor' : 1 }"""
         )

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

Set Keyspace

In [24]:
```
# TO-DO: Set KEYSPACE to the keyspace specified above

try:
    session.set_keyspace('song_library')
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 [25]:
```
## TO-DO: Query 1:  Give me the artist, song title and song's length in the music app h
## sessionId = 338, and itemInSession = 4

# Define the query to create the music_library table
query = """
CREATE TABLE IF NOT EXISTS user_song_sessions (
    user_id INT,
    session_id INT,
    item_in_session INT,
    artist TEXT,
    song TEXT,
    first_name TEXT,
    last_name TEXT,
    location TEXT,
    level TEXT,
    gender TEXT,
    length FLOAT,  -- Updated the type to FLOAT
    PRIMARY KEY ((user_id, session_id), item_in_session)
) WITH CLUSTERING ORDER BY (item_in_session ASC);

"""
try:
    session.execute(query)
except Exception as e:
    print(e)
```

In [26]:
```
import csv

# Define the file path
file = 'event_datafile_new.csv'

# Open and read the CSV file
with open(file, encoding='utf8') as f:
    csvreader = csv.reader(f)
    next(csvreader)  # Skip the header row

    # Iterate through each line in the CSV
    for line in csvreader:
        # Define the query for inserting data
        query = """
        INSERT INTO user_song_sessions (
                user_id,
                session_id,
                item_in_session,
                artist,
                song,
                first_name,
                last_name,
                location,
                level,
                gender,
                length
        ) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s);
        """

        # Prepare the data to be inserted
        data = (
            int(line[10]),
            int(line[8]),
            int(line[3]),
            (line[0]),
            line[9],
            (line[1]),
            line[4],
            line[7],
            (line[6]),
            line[2],
            float(line[5])
        )

        # Execute the query
```

```
            try:
                session.execute(query, data)
            except Exception as e:
                print(f"Error inserting data: {e}")
```

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

```
In [27]: query = "SELECT artist,song,length FROM user_song_sessions WHERE session_id=338 AND ite
         try:
             rows = session.execute(query)
             # Iterate through the results
             for row in rows:
                 print(row)  # This will print each row

             # Alternatively, if you want to see specific columns:
             for row in rows:
                 print(row.artist, row.song, row.sessionId, row.itemInSession)
         except Exception as e:
             print(f"Error: {e}")
```

```
Row(artist='Faithless', song='Music Matters (Mark Knight Dub)', length=495.30731201171875)
```

```
In [28]: ## TO-DO: Query 2: Give me only the following: name of artist, song (sorted by itemInSe
         ## for userid = 10, sessionid = 182
```

```
In [29]: query = "SELECT artist,song,first_name,last_name FROM user_song_sessions WHERE user_id=
         try:
             rows = session.execute(query)
             # Iterate through the results
             for row in rows:
                 print(row)  # This will print each row

             # Alternatively, if you want to see specific columns:
             for row in rows:
                 print(row.artist, row.song, row.sessionId, row.itemInSession)
         except Exception as e:
             print(f"Error: {e}")
```

```
Row(artist='Down To The Bone', song="Keep On Keepin' On", first_name='Sylvie', last_name='Cruz')
Row(artist='Three Drives', song='Greece 2000', first_name='Sylvie', last_name='Cruz')
Row(artist='Sebastien Tellier', song='Kilometer', first_name='Sylvie', last_name='Cruz')
Row(artist='Lonnie Gordon', song='Catch You Baby (Steve Pitron & Max Sanna Radio Edit)', first_n
```

```
In [30]: ## TO-DO: Query 3: Give me every user name (first and last) in my music app history who
```

In [ ]:

In [31]:
```
query = """
CREATE TABLE IF NOT EXISTS songs_by_title (
    song varchar,
    user_id varchar,
    first_name varchar,
    last_name varchar,
    PRIMARY KEY (song, user_id)
);

"""

try:
    session.execute(query)
except Exception as e:
    print(e)
```

In [33]:
```
# import csv

# # Define the file path
# file = 'event_datafile_new.csv'

# # Open and read the CSV file
# with open(file, encoding='utf8') as f:
#     csvreader = csv.reader(f)
#     next(csvreader)  # Skip the header row

#     # Iterate through each line in the CSV
#     for line in csvreader:
#         # Define the query for inserting data
#         query = """
#         INSERT INTO user_song_sessions (
#                 song,
#                 user_id,
#                 first_name,
#                 last_name
#         ) VALUES (%s, %s, %s, %s);
#         """

#         # Prepare the data to be inserted
#         data = (
#             line[9],
#             line[10],
#             line[1],
#             line[4]
#         )
```

```
#         # Execute the query
#         try:
#             session.execute(query, data)
#         except Exception as e:
#             print(f"Error inserting data: {e}")
```

```
In [36]: query = "SELECT first_name,last_name FROM songs_by_title WHERE song='All Hands Against
         try:
             rows = session.execute(query)
             # Iterate through the results
             for row in rows:
                 print(row)  # This will print each row

             # Alternatively, if you want to see specific columns:
             for row in rows:
                 print(row.artist, row.song, row.sessionId, row.itemInSession)
         except Exception as e:
             print(f"Error: {e}")
```

2.3.4 Drop the tables before closing out the sessions

```
In [43]: ## TO-DO: Drop the table before closing out the sessions
```

```
In [22]: for t in ["user_song_sessions"]:
             query = f"DROP TABLE {t}"
             try:
                 rows = session.execute(query)
             except Exception as e:
                 print(e)
```

2.3.5 Close the session and cluster connectionű

```
In [23]: session.shutdown()
         cluster.shutdown()
```

```
In [ ]:
```

```
In [ ]:
```