Project_1B_ Project_Template

June 17, 2020

Idelfonso Gutierrez June 2020

0.1 ETL Pipeline for Pre-Processing the Files with Python

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,'*'))
```

```
/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:
                    full_data_rows_list.append(line)

        print(len(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],
```

```
8056
```

```
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
```

0.2 Raw Data

Knowing our data types will allow us to do our tables `CREATE` precisely. Additionally, parsing any data to the correct type when doing out `INSERT` statements

```
In [25]: df = pd.read_csv('event_datafile_new.csv')
         df.dtypes
```

```
Out[25]: artist            object
         firstName         object
         gender            object
```

```
itemInSession      int64
lastName          object
length           float64
level             object
location          object
sessionId          int64
song              object
userId             int64
dtype: object
```

0.3 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:

0.4 Apache Cassandra Tables

Creating a Cluster

```
In [21]: from cassandra.cluster import Cluster
         try:
             cluster = Cluster(['127.0.0.1']) #For locally installed Apache Cassandra instance
             session = cluster.connect()
         except Exception as e:
             print(e)
```

Create Keyspace

```
In [22]: try:
             session.execute("""
             CREATE KEYSPACE IF NOT EXISTS music_app_history
             WITH REPLICATION =
             { 'class' : 'SimpleStrategy', 'replication_factor' : 1 }""")
         )

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

Set Keyspace

```
In [23]: try:
             session.set_keyspace('music_app_history')
         except Exception as e:
             print(e)
```

0.4.1 *Question 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

0.4.2 *Answer:*

Since we are working with a NoSQL database, Apache Cassandra, we need to think about our query first instead of how we can build relationships within entities. The latter it only works for relational databases, Postgres for example.

- *parameters:* sessionId, itemInSession
- *ouput:* artist, song title, and the length of song

By knowing our output, we can construct a table with these specific fields, discarding any other. Our results are based on *sessionId* and *itemInSession*; therefore these will become our PRIMARY KEY. Using the following `SELECT artist FROM table_one WHERE session_id=1 AND item_in_session=4` where `session_id` is the only primary key specify in our `CREATE` statement. The results will throw filtering error since we haven't specify the item_in_session as part of our primary key, which we must do.

With the requirements gathered above we can conclude the following

```
CREATE TABLE IF NOT EXISTS song_session (sessionId int,
                                         itemInSession int,
                                         artist_name text,
                                         song_title text,
                                         song_length float,
                                         PRIMARY KEY (sessionId, itemInSession))
```

```
In [8]: query = "CREATE TABLE IF NOT EXISTS song_info_session"
        query = query + "(session_id int, item_in_session float, artist text, song text, length
        try:
            rows = session.execute(query)
        except Exception as e:
            print(e)
```

Extracting from our CSV file and writting into our table

```
In [9]: 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_info_session (session_id, item_in_session, artist, son
                query = query + "VALUES (%s, %s, %s, %s, %s)"
                session.execute(query, (int(line[8]), float(line[3]), line[0], line[9], float(li
```

The analysis performed

```
In [10]: try:
             rows = session.execute("select artist, song, length from song_info_session where se
         except Exception as e:
             print(e)

         for row in rows:
             print(row.artist, row.song, row. length)
```

```
Faithless Music Matters (Mark Knight Dub) 495.30731201171875
```

0.4.3 *Question 2:*

Retrieve the name of the artist, song (sorted by itemInSession) and user (first and last name) for userid = 10, sessionid = 182

0.4.4 *Answer:*

- *parameters:* userId, sessionId
- *ouput:* artist, song title, first name and last name of the user

Spite that our requirements tells mention user, we must look further into our data. knowing that NoSQL doesn't handle relationships, this could be overlooked. Unless our `CSV` file contains a field that it's only name is `user` and it holds the full name of the user. In our case, we have first name and last name as fields in our text file, and most importantly we really a user field (except for user_id)

For this feature, we'll be using the **itemInSession as our clustering column**

```
CREATE TABLE IF NOT EXISTS song_playlist_session (userId int,
                                                  sessionId int,
                                                  itemInSession float,
                                                  artist text, song text,
                                                  firstName text,
                                                  lastName text,
                                                  PRIMARY KEY ((userId, sessionId), itemInSessio
```

```
In [11]: query = "CREATE TABLE IF NOT EXISTS song_playlist_session"
         query = query + "(userId int, sessionId int, itemInSession float, artist text, song tex
         try:
             session.execute(query)
         except Exception as e:
             print(e)
```

Extracting from our CSV file and writting into our table

```
In [12]: 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_playlist_session (userId, sessionId, itemInSession, a
                 query = query + "VALUES (%s, %s, %s, %s, %s, %s, %s)"
                 session.execute(query, (int(line[10]), int(line[8]), float(line[3]), line[0], l
```

The analysis performed

```
In [17]: try:
             rows = session.execute("select artist, song, firstname, lastname from song_playlist
         except Exception as e:
             print(e)

         for row in rows:
             print(row.artist, row.song, row.firstname, row.lastname)
```

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

0.4.5 *Question 3:*

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

0.4.6 *Answer:*

- *parameters:* song title
- *ouput:* first name and last name of the user

Looks like a simple query, rigth? We must be careful. We know that the first name, last name and the song's title are definitely not unique in the entire world; therefore we must rely in another feature to be able to do our `INSERT` statements. Otherwise Apache cassandra will overwritte the rows as we keep inserting with the any of those three fields if they already exist within our table. Unless we know we want to do this and just perform and `UPDATE`. In this case, we will not be doing any updates.

```
CREATE TABLE IF NOT EXISTS user_song_history (song text,
                                              userId int,
                                              firstName text,
                                              lastName text,
                                              PRIMARY KEY (song, userId))
```

```
In [18]: query = "CREATE TABLE IF NOT EXISTS user_song_history"
         query = query + "(song text, userId int, firstName text, lastName text, PRIMARY KEY (so
         try:
             session.execute(query)
         except Exception as e:
             print(e)
```

Extracting from our CSV file and writting into our table

```
In [19]: 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_song_history (song, userId, firstName, lastName)"
                 query = query + "VALUES (%s, %s, %s, %s)"
                 session.execute(query, (line[9], int(line[10]), line[1], line[4]))
```

The analysis performed

```
In [20]: try:
             rows = session.execute("select firstName, lastName from user_song_history where son
         except Exception as e:
             print(e)

         for row in rows:
             print(row.firstname, row.lastname)
```

```
Jacqueline Lynch
Tegan Levine
Sara Johnson
```

0.4.7 Drop the tables before closing out the sessions

```
In [24]: query = "drop keyspace if exists music_app_history"
         try:
             rows = session.execute(query)
         except Exception as e:
             print(e)
```

0.4.8 Close the session and cluster connectionű

```
In [22]: session.shutdown()
         cluster.shutdown()
```