Cassandra_project_output

April 18, 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(len(file_path_list), file_path_list[0])
```

```
/home/workspace
30 /home/workspace/event_data/2018-11-27-events.csv
```

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],
```

```
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))
df = pd.read_csv('event_datafile_new.csv')
df.head()
```

```
6821
```

Out[4]:

	artist	firstName	gender	\
0	Barry Tuckwell/Academy of St Martin-in-the-Fie...	Mohammad	M	

```
1                                      Jimi Hendrix  Mohammad      M
2                                      Building 429  Mohammad      M
3                                        The B-52's    Gianna      F
4                                  Die Mooskirchner    Gianna      F

   itemInSession   lastName     length level  \
0              0  Rodriguez  277.15873  paid
1              1  Rodriguez  239.82975  paid
2              2  Rodriguez  300.61669  paid
3              0      Jones  321.54077  free
4              1      Jones  169.29914  free

                                location  sessionId  \
0  Sacramento--Roseville--Arden-Arcade, CA        961
1  Sacramento--Roseville--Arden-Arcade, CA        961
2  Sacramento--Roseville--Arden-Arcade, CA        961
3    New York-Newark-Jersey City, NY-NJ-PA        107
4    New York-Newark-Jersey City, NY-NJ-PA        107

                                              song  userId
0  Horn Concerto No. 4 in E flat K495: II. Romanc...      88
1                          Woodstock Inprovisation      88
2                             Majesty (LP Version)      88
3                                       Love Shack      38
4                                Frisch und g'sund      38
```

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()

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

In [6]:
```
# Utility function
def get_insert_values_from_line(line, columns, file_columns):
    '''
    Example:
        line = ['Barry Tuckwell/Academy of St Martin-in-the-Fields/Sir Neville Marriner'
        'Mohammad', 'M', '0', 'Rodriguez', '277.15873', 'paid', 'Sacramento--Roseville--
        '961', 'Horn Concerto No. 4 in E flat K495: II. Romance (Andante cantabile)', '8

        columns = [('sessionId', int),
                   ('itemInSession', int),
                   ('artist', str),
                   ('song', str),
                   ('length', float)]

        file_columns = ['artist', 'firstName', 'gender', 'itemInSession',
        'lastName', 'length', 'level', 'location', 'sessionId', 'song', 'userId']

        output of this function = (961, 0, 'Barry Tuckwell/Academy of St Martin-in-the-M
        'Horn Concerto No. 4 in E flat K495: II. Romance (Andante cantabile)', 277.15873

    Args:
        line : This is a line from a csv file
        columns: A two element tuple of column name and python data transformation that
        file_columns: All columns that map to all data in line

    Returns:
        A tuple with all transformation applied to the denoted columns in the columns in

    '''
    output = [func(line[file_columns.index(x)]) for x, func in columns]
    return tuple(output)

# runs any query
def run_query(query):
    try:
        results = session.execute(query)
```

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

Create Keyspace

```
In [7]: # DROP the keyspace
        query = '''
        DROP KEYSPACE IF EXISTS udacity_project;
        '''
        run_query(query)

        # Create a Keyspace
        query = '''
            CREATE KEYSPACE IF NOT EXISTS udacity_project
            WITH REPLICATION = {
            'class': 'SimpleStrategy',
            'replication_factor': 1
            }

            '''

        run_query(query)

Out[7]: <cassandra.cluster.ResultSet at 0x7fbf3c522160>
```

Set Keyspace

```
In [8]: # Set KEYSPACE to the keyspace specified above
        try:
            session.set_keyspace('udacity_project')
        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 [9]: # Doing creation, insertion and selection once in a workflow
        def workflow_run(table_name, create_query, insert_query, select_query,
```

```
            column_transformations, selection_attrs):

'''
Args:
    table_name: Table name of the workflow scenario
    create_query: A CQL query to create tables
    insert_query: A CQL query to insert data in table
    select_query: A CQL query to select data from table
    column_transformation: A two element tuple list where
        1. First element: Name of column that needs to be extracted from csv file li
        2. Second element: A python function to transform the column value to approp
    selecton_attrs: A list of columns on which selection happens

Returns:
    None

Working:
    This function simulates a scenario where Create, Insert and Select queries are
    executed sequentially and the results are reported
'''



from IPython.display import display
run_query(create_query)
print("Ran create query, table {} should be created by now".format(table_name))

# We have provided part of the code to set up the CSV file. Please complete the Apac
file = 'event_datafile_new.csv'
df = pd.read_csv(file)
file_columns = df.columns.tolist()

with open(file, encoding = 'utf8') as f:
    csvreader = csv.reader(f)
    next(csvreader) # skip header
    for line in csvreader:
        session.execute(insert_query, get_insert_values_from_line(line, column_trans

print("Ran insert query, insertions in table should be complete by now")
rows = run_query(select_query)
out = []
for row in rows:
    out.append([getattr(row,s) for s in selection_attrs if hasattr(row, s)])

print("Data read complete")
display(pd.DataFrame(out, columns=selection_attrs))
```

In [10]:
```
## Query 1:  Give me the artist, song title and song's length in the music app history
## sessionId = 338, and itemInSession = 4

'''
Query design logic::

Choice of Partition keys and clustering columns:
    - We see that the query needs to uniquely identify records based on sessionId and i
    - Therefore we use session id and item session id as our partition key
    - No clustering columns are required since we are not sorting data on a node/partit
    in the context of this query

Choice of columns for select query:
    - We see that the query requires artist, song_title and song's length as the output
    - Hence the relevant output parameters are artist_name, song_title and song_length

Filtering columns:
    - The where clause is implemented on session_id and item_in_session column (in orde
    - This is as per the query specification

'''


table_name = 'session_library'
create_query = '''
CREATE TABLE IF NOT EXISTS session_library
(session_id int, item_in_session int, artist_name text, song_title text, song_length fl
PRIMARY KEY (session_id, item_in_session))
'''
insert_query = '''INSERT INTO session_library
(session_id , item_in_session , artist_name, song_title , song_length) VALUES (%s, %s,

select_query = '''
select artist_name, song_title, song_length from session_library where
session_id = 338 and item_in_session = 4
'''
column_transformations = [('sessionId', int),
                          ('itemInSession', int),
                          ('artist', str),
                          ('song', str),
                          ('length', float)]
selection_attr = ['artist_name', 'song_title', 'song_length']

# Sceanrio 1 query demonstration
workflow_run(table_name,
             create_query, insert_query, select_query,
             column_transformations, selection_attr)
```

```
Ran create query, table session_library should be created by now
Ran insert query, insertions in table should be complete by now
Data read complete
```

```
  artist_name                        song_title  song_length
0   Faithless  Music Matters (Mark Knight Dub)   495.307312
```

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

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

```
In [11]: ## Query 2: Give me only the following: name of artist, song (sorted by itemInSession)
         ## for userid = 10, sessionid = 182

         '''
         Query design logic::

         Choice of Partition keys and clustering columns:
             - We see that the query needs to uniquely identify records based on user id and ses
             - We also recognize that the song needs to be sorted by item session id
             - Therefore we use user id and item session id as our partition key
             - We use item session id as our clustering column since it is required to sort the

         Choice of columns for select query:
             - We see that the query requires artist name, song_title and user's first and last
             - Hence the relevant output parameters are artist_name, song_title, first_name and

         Filtering columns:
             - The where clause is implemented on user_id and session_id column (in order)
             - This is as per the query specification
         '''

         table_name = 'session_user_library'
         column_transformations = [('userId', str), ('sessionId', int), ('artist', str),
                         ('song', str), ('itemInSession', int), ('firstName', str), ('lastName'
         create_query = '''
         CREATE TABLE IF NOT EXISTS session_user_library
         (user_id text, session_id int, artist_name text, song_title text,
         item_in_session int, first_name text, last_name text,
         PRIMARY KEY ((user_id, session_id), item_in_session))
         '''
         insert_query = '''INSERT INTO session_user_library
                           (user_id, session_id, artist_name,
                            song_title, item_in_session, first_name, last_name)
                     VALUES (%s, %s, %s, %s, %s, %s, %s)'''
```

```
        select_query = '''
        select artist_name, song_title, first_name, last_name, item_in_session from session_use
        session_id = 182 and  user_id = '10'
        '''
        selection_attr = ['artist_name', 'song_title', 'first_name', 'last_name', 'item_in_sess

        # Sceanrio 2 query demonstration
        workflow_run(table_name,
                     create_query, insert_query, select_query,
                     column_transformations, selection_attr)
```

```
Ran create query, table session_user_library should be created by now
Ran insert query, insertions in table should be complete by now
Data read complete
```

```
         artist_name                                           song_title  \
0   Down To The Bone                                 Keep On Keepin' On
1       Three Drives                                       Greece 2000
2  Sebastien Tellier                                         Kilometer
3      Lonnie Gordon  Catch You Baby (Steve Pitron & Max Sanna Radio...

  first_name last_name  item_in_session
0     Sylvie      Cruz                0
1     Sylvie      Cruz                1
2     Sylvie      Cruz                2
3     Sylvie      Cruz                3
```

In [12]:
```
        ## Query 3: Give me every user name (first and last) in my music app history
        ## who listened to the song 'All Hands Against His Own'

        '''
        Query design logic::

        Choice of Partition keys and clustering columns:
            - We see that the query needs to uniquely identify records based on song_title
            - But the query also needs all historical users who have used the app
            - Therefore, we need to partition the data based on song_title as well as the user_
            identifies all the users who have listened to a particular song
            - No clustering columns are required since we are not sorting data on a node/partit
            in the context of this query

        Choice of columns for select query:
            - We see that the query requires user's first and last name as the output
            - Hence the relevant output parameters are first_name and last_name

        Filtering columns:
```

```
            - The where clause is implemented on song_title as it is partitioned first on song_
            - Record of all users is thus available for a song_title
        '''



        table_name = 'user_song_library'
        column_transformations = [('song', str), ('userId', str),
                                  ('firstName', str), ('lastName', str)]
        create_query = '''
        CREATE TABLE IF NOT EXISTS user_song_library
        (song_title text, user_id text, first_name text, last_name text,
        PRIMARY KEY (song_title, user_id))
        '''
        insert_query = '''INSERT INTO user_song_library
                          (song_title, user_id, first_name, last_name)
                    VALUES (%s, %s, %s, %s)'''
        select_query = '''
        select first_name, last_name from user_song_library where
        song_title = 'All Hands Against His Own'
        '''
        selection_attr = ['first_name', 'last_name']

        # Sceanrio 3 query demonstration
        workflow_run(table_name,
                     create_query, insert_query, select_query,
                     column_transformations, selection_attr)
```

```
Ran create query, table user_song_library should be created by now
Ran insert query, insertions in table should be complete by now
Data read complete
```

```
   first_name last_name
0  Jacqueline     Lynch
1       Tegan    Levine
2        Sara   Johnson
```

2.3.5 Drop the tables before closing out the sessions

```
In [13]: for table in ['session_library','session_user_library', 'user_song_library']:
             delete_query = "DROP TABLE IF EXISTS {}".format(table)
             print("Deleting {}".format(table))
             run_query(delete_query)
```

```
Deleting session_library
Deleting session_user_library
```

```
Deleting user_song_library
```

2.3.6 Close the session and cluster connectionű

In [14]:
```
session.shutdown()
cluster.shutdown()
```

In [ ]:

In [ ]: