Model Data using Cassandra

The aim of the project is to solve the three queries given below.

Introduction

There is a music streaming app called SoundCloud, that has been using their music streaming app and collecting data on songs and user activity and their aim is to analyze this data especially understanding what songs users are listening to. Currently, they are not making use of a NoSQL db and they have the data stored as a CSV file, thus its difficult for them to query the data. So our task is to create a NoSQL database for helping them with the analysis.

Import Packages

In [1]:
```
import pandas as pd
import numpy as np
import cassandra
import csv
```

In [2]:
```
# Check the number of rows in the new CSV file
with open('event_data.csv', 'r', encoding='utf8') as f:
    print("Number of rows in the  CSV file:", sum(1 for line in f))
```

```
Number of rows in the  CSV file: 6821
```

The image below is a screenshot of what the data appears like in the event_data.csv

 No description has been provided for this image

Creating a Cluster

In [3]:
```
# Task: Make a connection to the cassandra instance on your local machine(127.0.0.1) and
# create a session to establish connection and begin executing queries

from cassandra.cluster import Cluster

try:
    # Connect to the Cassandra cluster
    cluster = Cluster(['127.0.0.1'])
    session = cluster.connect()

    # Print cluster and session information
    print("Cluster:", cluster)
    print("Session:", session)

    # Get and print information about connected nodes
    for host in cluster.metadata.all_hosts():
        print("Host ID:", host.address, "\nConnection Status:", host.is_up)

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

```
Cluster: <cassandra.cluster.Cluster object at 0x12759fb90>
Session: <cassandra.cluster.Session object at 0x141063690>
Host ID: 127.0.0.1
Connection Status: True
```

Create & Set Keyspace

In [4]:
```
# Task: Create a Keyspace and Set KEYSPACE to the keyspace specified above

try:
    # Create keyspace if not exists and set keyspace
    session.execute("""
    CREATE KEYSPACE IF NOT EXISTS soundcloud
    WITH REPLICATION =
    { 'class' : 'SimpleStrategy', 'replication_factor' : 1 }
    """)

    session.set_keyspace('soundcloud')

    print("Keyspace 'soundcloud' created successfully and set as the current keyspace.")
    #print("Current keyspace:", session.execute("SELECT * FROM system.local")[0])
    current_keyspace = session.keyspace

    # Print the current keyspace
    print("Current keyspace:", current_keyspace)
```

```
except Exception as e:
    print("An error occurred:", e)
```

```
Keyspace 'soundcloud' created successfully and set as the current keyspace.
Current keyspace: soundcloud
```

List of Queries

1. Find the artist_name, song_title and length of song the SoundCloud app history that was heard during session_number = 338, and item_in_session_number = 4

2. Find the artist_name, song_title (sorted by item_in_session_number) and name(fname and lname) of the user for user_id = 10, session_number = 182

3. Find every name(fname and lname) of the user from the SoundCloud app history that listened to the song_title 'All Hands Against His Own'

Query1 Table1: How should we model this data? Think about what should be our Primary Key/Partition Key/Clustering Key

In [5]:
```
## Task: Query 1: Find the artist_name, song_title and length of song the SoundCloud app history
## that was heard during session_number = 338, and item_in_session_number = 4
## make use of create table command
table_1 = 'song_info_by_session'
try:
    # Create the query
    query_1 = (f"""CREATE TABLE IF NOT EXISTS {table_1} (
        session_number int,
        item_in_session int,
        artist_name text,
        song_title text,
        length float,
        PRIMARY KEY (session_number, item_in_session))""")

    # Execute the query
    session.execute(query_1)

    # If the query runs successfully, print the status
    print("Table 'song_info_by_session' created successfully.")
    print("\nSchema of 'song_info_by_session' table:", '\n')
    #print("-"*146)
    result = session.execute("SELECT * FROM system_schema.columns WHERE keyspace_name = 'soundcloud' AND table_name

    song_info_by_session_df = pd.DataFrame(result)
    #print(song_info_by_session_df.to_markdown())
    #print("-"*146)

except Exception as e:
    print("An error occurred:", e)

song_info_by_session_df
```

```
Table 'song_info_by_session' created successfully.

Schema of 'song_info_by_session' table:
```

Out[5]:

	keyspace_name	table_name	column_name	clustering_order	column_name_bytes	kind	position	type
0	soundcloud	song_info_by_session	artist_name	none	b'artist_name'	regular	-1	text
1	soundcloud	song_info_by_session	item_in_session	asc	b'item_in_session'	clustering	0	int
2	soundcloud	song_info_by_session	length	none	b'length'	regular	-1	float
3	soundcloud	song_info_by_session	session_number	none	b'session_number'	partition_key	0	int
4	soundcloud	song_info_by_session	song_title	none	b'song_title'	regular	-1	text

Let's insert our data into of table

In [6]:
```
# We have provided part of the code to set up the CSV file. Please complete the Apache Cassandra code below#
# I have change the code a littel bit for understanding and to analysiing the output of the code, also to check wea

file_name = 'event_data.csv'

try:
    with open(file_name, encoding='utf8') as f:
        csv_reader = csv.reader(f)
```

```
        next(csv_reader)  # Skip the header in the CSV file
        for row in csv_reader:
            ## Task: Write the INSERT statements and assign it to the query variable
            query1 = "INSERT INTO song_info_by_session (session_number, item_in_session, artist_name, song_title, l
            query1 += "VALUES (%s, %s, %s, %s, %s)"


            # Task: Match the column in the csv file to the column in the INSERT statement.
            ## e.g., if you want to INSERT gender from csv file into the database you will use row[2]
            ## e.g., if you want to INSERT location from csv file into database you will use row[7]

            session.execute(query1, (int(row[8]), int(row[3]), row[0], row[9], float(row[5])))

    print(f"Data insertion in {table_1} table completed successfully.", '\n')

except Exception as e:
    print("An error occurred:", e)

# Print the output of the query
print(f"Table {table_1} after inserting the data")
rows = session.execute("SELECT * FROM song_info_by_session")
rows_dataframe = pd.DataFrame(rows)
rows_dataframe
```

```
Data insertion in song_info_by_session table completed successfully.

Table song_info_by_session after inserting the data
```

Out[6]:

	session_number	item_in_session	artist_name	length	song_title
0	23	0	Regina Spektor	191.085266	The Calculation (Album Version)
1	23	1	Octopus Project	250.957916	All Of The Champs That Ever Lived
2	23	2	Tegan And Sara	180.061584	So Jealous
3	23	3	Dragonette	153.390564	Okay Dolores
4	23	4	Lil Wayne / Eminem	229.589752	Drop The World
...	...	...	...	...	...
6815	986	1	Jack Johnson	240.064850	Taylor
6816	986	2	Iron And Wine	153.050980	Naked As We Can
6817	986	3	The xx	158.249344	Fantasy
6818	986	4	The Antlers	328.881195	Epilogue
6819	986	5	Fattburger	217.207703	Groovin'

6820 rows × 5 columns

Validate our Data Model using a SELECT

In [7]:
```
## Task: Make use of the SELECT statement and for loop to check if your query works and display the results

validate_query_1 = "SELECT artist_name, song_title, length FROM song_info_by_session WHERE session_number = 338 AND
try:
    validate_row_1 = session.execute(validate_query_1)
except Exception as e:
    print(e)
print(f"From table  : {table_1}")
validate_row_dataframe = pd.DataFrame(validate_row_1)

validate_row_dataframe
```

```
From table  : song_info_by_session
```

Out[7]:

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

Query2 Table2: How should we model this data? Think about what should be our Primary Key/Partition Key/Clustering Key

In [8]:
```
## Task: Query 2: Find the artist_name, song_title (sorted by item_in_session_number) and
## name(fname and lname) of the user for user_id = 10, session_number = 182
## make use of create table command

table_2 = 'user_info_by_session'
try:
    # Create the query
    query_2 = (f"""CREATE TABLE IF NOT EXISTS {table_2} (
```

```
        user_id int,
        session_number int,
        item_in_session int,
        artist_name text,
        song_title text,
        fname text,
        lname text,
        PRIMARY KEY ((user_id, session_number), item_in_session))""")

    # Execute the query
    session.execute(query_2)

    # If the query runs successfully, print the status
    print("Table 'user_info_by_session' created successfully.")
    print("\nSchema of 'user_info_by_session' table:", '\n')
    #print("-"*146)
    result_2 = session.execute("SELECT * FROM system_schema.columns WHERE keyspace_name = 'soundcloud' AND table_na

    user_info_by_session_df = pd.DataFrame(result_2)
    #print(song_info_by_session_df.to_markdown())
    #print("-"*146)

except Exception as e:
    print("An error occurred:", e)

user_info_by_session_df
```

```
Table 'user_info_by_session' created successfully.

Schema of 'user_info_by_session' table:
```

Out[8]:

	keyspace_name	table_name	column_name	clustering_order	column_name_bytes	kind	position	type
0	soundcloud	user_info_by_session	artist_name	none	b'artist_name'	regular	-1	text
1	soundcloud	user_info_by_session	fname	none	b'fname'	regular	-1	text
2	soundcloud	user_info_by_session	item_in_session	asc	b'item_in_session'	clustering	0	int
3	soundcloud	user_info_by_session	lname	none	b'lname'	regular	-1	text
4	soundcloud	user_info_by_session	session_number	none	b'session_number'	partition_key	1	int
5	soundcloud	user_info_by_session	song_title	none	b'song_title'	regular	-1	text
6	soundcloud	user_info_by_session	user_id	none	b'user_id'	partition_key	0	int

Let's insert our data into of table

In [9]:

```
# We have provided part of the code to set up the CSV file. Please complete the Apache Cassandra code below#

file_name = 'event_data.csv'

try:
    with open(file_name, encoding='utf8') as f:
        csv_reader = csv.reader(f)
        next(csv_reader)  # Skip the header in the CSV file
        for row_2 in csv_reader:
            ## Task: Write the INSERT statements and assign it to the query variable
            query2 = "INSERT INTO user_info_by_session (user_id, session_number, item_in_session, artist_name, song
            query2 = query2 + " VALUES (%s, %s, %s, %s, %s, %s, %s)"


            # Task: Match the column in the csv file to the column in the INSERT statement.
            ## e.g., if you want to INSERT gender from csv file into the database you will use row[2]
            ## e.g., if you want to INSERT location from csv file into database you will use row[7]
            session.execute(query2, (int(row_2[10]), int(row_2[8]), int(row_2[3]), row_2[0], row_2[9], row_2[1], ro



    print(f"Data insertion in {table_2} table completed successfully.", '\n')

except Exception as e:
    print("An error occurred:", e)

# Print the output of the query
print(f"Table {table_2} after inserting the data")
rows_2 = session.execute("SELECT * FROM user_info_by_session")
rows_2_dataframe = pd.DataFrame(rows_2)
rows_2_dataframe
```

```
Data insertion in user_info_by_session table completed successfully.

Table user_info_by_session after inserting the data
```

Out[9]:

	user_id	session_number	item_in_session	artist_name	fname	lname	song_title
0	58	768	0	System of a Down	Emily	Benson	Sad Statue
1	58	768	1	Ghostland Observatory	Emily	Benson	Stranger Lover
2	58	768	2	Evergreen Terrace	Emily	Benson	Zero
3	85	776	2	Deftones	Kinsley	Young	Head Up (LP Version)
4	85	776	3	The Notorious B.I.G.	Kinsley	Young	Playa Hater (Amended Version)
...	...	...	...	...	...	...	...
6815	49	576	13	Biz Markie	Chloe	Cuevas	This Is Something for the Radio
6816	49	576	14	Animal Collective	Chloe	Cuevas	Brother Sport
6817	49	576	15	Linkin Park	Chloe	Cuevas	Given Up (Album Version)
6818	88	441	1	BeyoncÃ Â©	Mohammad	Rodriguez	Get Me Bodied
6819	88	441	2	Nate Dogg	Mohammad	Rodriguez	Never Leave Me Alone

6820 rows × 7 columns

Validate our Data Model using a SELECT

In [10]:
```
## Task: Make use of the SELECT statement and for loop to check if your query works and display the results

validate_query_2 = "SELECT artist_name, song_title, fname, lname FROM user_info_by_session WHERE user_id = 10 AND s
try:
    validate_row_2 = session.execute(validate_query_2)
except Exception as e:
    print(e)
print(f"From table  : {table_2}")

validate_row_2_dataframe = pd.DataFrame(validate_row_2)
validate_row_2_dataframe
```

```
From table  : user_info_by_session
```

Out[10]:

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

Query3 Table3: How should we model this data? Think about what should be our Primary Key/Partition Key/Clustering Key

In [11]:
```
## Task: Query 3: Find every name(first and lastname) of the user from the SoundCloud app history that listened
## to the song_title 'All Hands Against His Own'
## make use of create table command


table_3 = 'user_info_by_song'
try:
    # Create the query
    query_3 = (f"""CREATE TABLE IF NOT EXISTS {table_3} (
        song_title text,
        user_id int,
        fname text,
        lname text,
        PRIMARY KEY (song_title, user_id))""")

    # Execute the query
    session.execute(query_3)

    # If the query runs successfully, print the status
    print("Table 'user_info_by_song' created successfully.")
    print("\nSchema of 'user_info_by_song' table:", '\n')
    #print("-"*146)
    result_3 = session.execute("SELECT * FROM system_schema.columns WHERE keyspace_name = 'soundcloud' AND table_na

    user_info_by_song_df = pd.DataFrame(result_3)
    #print(song_info_by_session_df.to_markdown())
    #print("-"*146)

except Exception as e:
    print("An error occurred:", e)
```

```
user_info_by_song_df
```

```
Table 'user_info_by_song' created successfully.

Schema of 'user_info_by_song' table:
```

Out[11]:

	keyspace_name	table_name	column_name	clustering_order	column_name_bytes	kind	position	type
0	soundcloud	user_info_by_song	fname	none	b'fname'	regular	-1	text
1	soundcloud	user_info_by_song	lname	none	b'lname'	regular	-1	text
2	soundcloud	user_info_by_song	song_title	none	b'song_title'	partition_key	0	text
3	soundcloud	user_info_by_song	user_id	asc	b'user_id'	clustering	0	int

Let's insert our data into of table

In [12]:
```
# We have provided part of the code to set up the CSV file. Please complete the Apache Cassandra code below#

file_name = 'event_data.csv'

try:
    with open(file_name, encoding='utf8') as f:
        csv_reader = csv.reader(f)
        next(csv_reader)  # Skip the header in the CSV file
        for row_3 in csv_reader:
            ## Task: Write the INSERT statements and assign it to the query variable
            query3 = "INSERT INTO user_info_by_song (song_title, user_id, fname, lname)"
            query3 = query3 + " VALUES (%s, %s, %s, %s)"


            # Task: Match the column in the csv file to the column in the INSERT statement.
            ## e.g., if you want to INSERT gender from csv file into the database you will use row[2]
            ## e.g., if you want to INSERT location from csv file into database you will use row[7]
            session.execute(query3, (row_3[9], int(row_3[10]), row_3[1], row_3[4]))



    print(f"Data insertion in {table_3} table completed successfully.", '\n')

except Exception as e:
    print("An error occurred:", e)

# Print the output of the query
print(f"Table {table_3} after inserting the data")
rows_3 = session.execute("SELECT * FROM user_info_by_song")
rows_3_dataframe = pd.DataFrame(rows_3)
rows_3_dataframe
```

```
Data insertion in user_info_by_song table completed successfully.

Table user_info_by_song after inserting the data
```

Out[12]:

	song_title	user_id	fname	lname
0	Wonder What's Next	49	Chloe	Cuevas
1	In The Dragon's Den	49	Chloe	Cuevas
2	Too Tough (1994 Digital Remaster)	44	Aleena	Kirby
3	Rio De Janeiro Blue (Album Version)	49	Chloe	Cuevas
4	My Place	15	Lily	Koch
...	...	...	...	...
6613	Heartbreaker	95	Sara	Johnson
6614	Heartbreaker	97	Kate	Harrell
6615	Leave The City And Come Home	49	Chloe	Cuevas
6616	Tale Of Two Cities	30	Avery	Watkins
6617	Ball & Chain	29	Jacqueline	Lynch

6618 rows × 4 columns

Validate our Data Model using a SELECT

In [13]:
```
## Task: Make use of the SELECT statement and for loop to check if your query works and display the results

validate_query_3 = "SELECT fname, lname FROM user_info_by_song WHERE song_title = 'All Hands Against His Own'"

try:
```

```
    validate_row_3 = session.execute(validate_query_3)

except Exception as e:
    print(e)
print(f"From table  : {table_3}")

validate_row_3_dataframe = pd.DataFrame(validate_row_3)
validate_row_3_dataframe
```

```
From table  : user_info_by_song
```

Out[13]:

	fname	lname
0	Jacqueline	Lynch
1	Tegan	Levine
2	Sara	Johnson

Drop the tables before closing out the sessions

In [14]:
```
drop_statements = [
    "DROP TABLE IF EXISTS song_info_by_session",
    "DROP TABLE IF EXISTS user_info_by_session",
    "DROP TABLE IF EXISTS user_info_by_song"
]
try:
    for index, statement in enumerate(drop_statements, start=1):
        session.execute(statement)
        print(f"Table {index} dropped successfully.")

except Exception as e:
    print("An error occurred:", e)
```

```
Table 1 dropped successfully.
Table 2 dropped successfully.
Table 3 dropped successfully.
```

Close the session and cluster connection¶

In [15]:
```
try:
    session.shutdown()
    cluster.shutdown()

    print("Session and cluster shutdown successfully.")

except Exception as e:
    print("An error occurred:", e)
```

```
Session and cluster shutdown successfully.
```