Data Modeling with Apache Cassandra Project Starter Code

The objective of this project is to apply data modeling with Apache Cassandra and build an ETL pipeline using Python.

How to run the program with your own code

For the execution of your own code, we head to the Project Workspace.

In the project workspace we can open the file Project_1B_ Project_Template.ipynb and execute each of the sentences, another way to execute the file is from the terminal with the following command:

```
    jupyter notebook Project_1B_ Project_Template.ipynb
```

The summary of the files and folders within repo is provided in the table below:

File/Folder	Definition
event_data/*	Folder that contains all the csv files with the data used in this project.
images/*	Folder containing the images of the project.
event_datafile_new.csv	Contains the denormalized dataset that was generated in the ETL pipeline procedures.

File/Folder	Definition
Project_1B_ Project_Template.ipynb	Reads and processes the denormalized dataset and loads the data into the tables. This notebook contains detailed instructions on the ETL process, the data to be loaded in each of the three examples as well as their corresponding queries.
README.md	Contains the project documentation.
README.pdf	Contains the project documentation in PDF format.

Steps to complete the project:

Modeling your NoSQL database or Apache Cassandra database.

1. Design tables to answer the queries outlined in the project template.
2. Write Apache Cassandra CREATE KEYSPACE and SET KEYSPACE statements.
3. Develop your CREATE statement for each of the tables to address each question.
4. Load the data with INSERT statement for each of the tables.
5. Include IF NOT EXISTS clauses in your CREATE statements to create tables only if the tables do not already exist. We recommend you also include DROP TABLE statement for each table, this way you can run drop and create tables whenever you want to reset your database and test your ETL pipeline.
6. Test by running the proper select statements with the correct WHERE clause.

Build ETL Pipeline

1. Implement the logic in section Part I of the notebook template to iterate through each event file in event_data to process and create a new CSV file in Python.
2. Make necessary edits to Part II of the notebook template to include Apache Cassandra CREATE and INSERT statements to load processed records into relevant tables in your data model.
3. Test by running SELECT statements after running the queries on your database.

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Scenario.

A startup called Sparkify wants to analyze the data they've been collecting on songs and user activity on their new music streaming app. The analysis team is particularly interested in understanding what songs users are listening to. Currently, there is no easy way to query the data to generate the results, since the data reside in a directory of CSV files on user activity on the app.

They'd like a data engineer to create an Apache Cassandra database which can create queries on song play data to answer the questions, and wish to bring you on the project. Your role is to create a database for this analysis. You'll be able to test your database by running queries given to you by the analytics team from Sparkify to create the results.

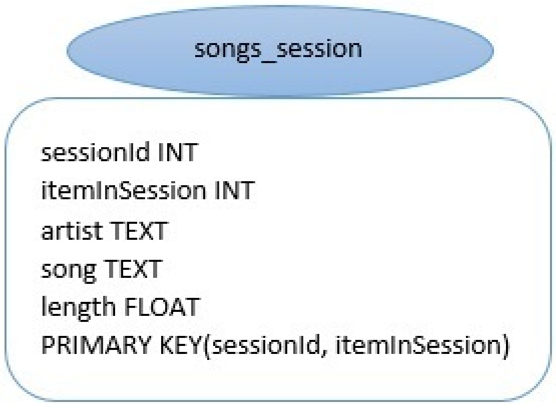
Denormalizing the Database.

The goal of denormalization in this context is to reduce the amount of time needed to read data. Unlike relational databases, non-relational databases have been optimized for fast reads and writes; therefore, denormalization is a must! Always thinking about the necessary queries first and designing the denormalization scheme accordingly. One table per query is a good strategy.

Denormalization changes the application: First, it means data redundancy, which translates to significantly increased storage costs. Second, fixing data inconsistency is now the main job of the application.

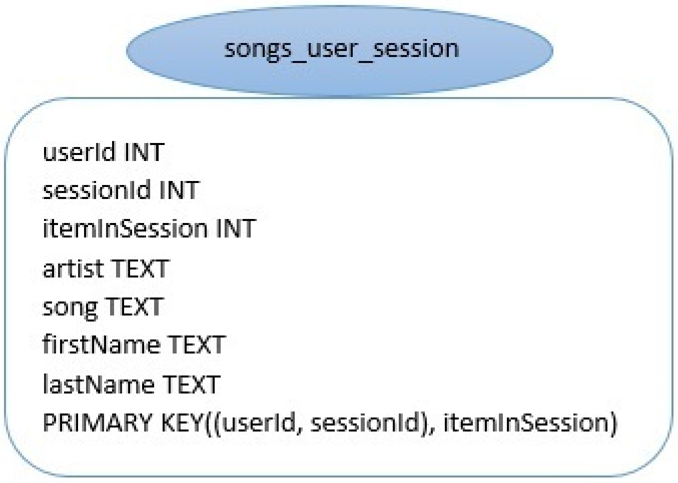
Again, Data Modeling in Apache Cassandra is query focused – and that focus needs to be put on the WHERE clause. This clause allows to do fast reads. Note that the partition key always needs to be included in the query! The clustering columns can be used to put the results in order.

The tables that will contain the data of our consultations are:



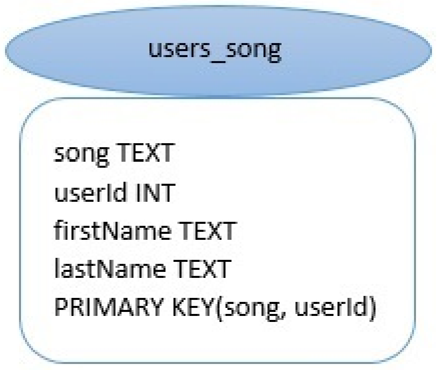
songs_session

sessionId INT
itemInSession INT
artist TEXT
song TEXT
length FLOAT
PRIMARY KEY(sessionId, itemInSession)



songs_user_session

userId INT
sessionId INT
itemInSession INT
artist TEXT
song TEXT
firstName TEXT
lastName TEXT
PRIMARY KEY((userId, sessionId), itemInSession)



users_song

song TEXT
userId INT
firstName TEXT
lastName TEXT
PRIMARY KEY(song, userId)

Apache Cassandra.

Aside from being a backbone for Facebook, Uber, and Netflix, Cassandra is a very scalable and resilient database that is easy to master and simple to configure. Apache Cassandra uses its own query language – CQL – which is similar to SQL. Note that JOINS, GROUP BY, or subqueries are not supported by CQL.

Some terms used in Cassandra differ from those we already know:

A keyspace, for example, is analogous to the term database in a relational database.

Another example is a partition, which is a collection of rows. Cassandra organizes data into partitions; there, each partition consists of multiple columns.

Partitions are stored on a node. Nodes (or servers) are generally part of a cluster where each node is responsible for a fraction of the partitions.

The Primary Key defines how each row can be uniquely identified and how the data is distributed across the nodes in our system. A partition key is responsible for identifying the partition or node in the cluster that stores a row – whereas the purpose of a clustering key (or clustering column) is to store row data within a partition in a sorted order.

When we have only one partition key and no clustering column, it is called a Single Primary Key. Should we use one (or more) partition key(s) and one (or more) clustering column(s) instead, we call it a Compound Primary Key or Composite Primary Key.

Data.

The data used in this project, it's better to understand what they represents.

Song Dataset.

We'll be working with dataset: event_data. The directory of CSV files partitioned by date. For example, here are the file paths for this dataset.

```
event_data/2018-11-01-events.csv
event_data/2018-11-02-events.csv
event_data/2018-11-03-events.csv
.
.
event_data/2018-11-30-events.csv
```

These files are in CSV format and contains several records with the song data separated by a comma, below is an example of what a single song file, 2018-11-01-events.csv, looks like.

```
Black Eyed Peas,Logged In,Sylvie,F,0,Cruz,214.93506,free,"Washington-Arlington-Ale
```

The code to pre-process the CSV files was provided already. So no need to go in-depth for it.

ETL Pipeline.

Extract, transform, load (ETL) is the general procedure of copying data from one or more sources into a destination system which represents the data differently from, or in a different context than, the sources.

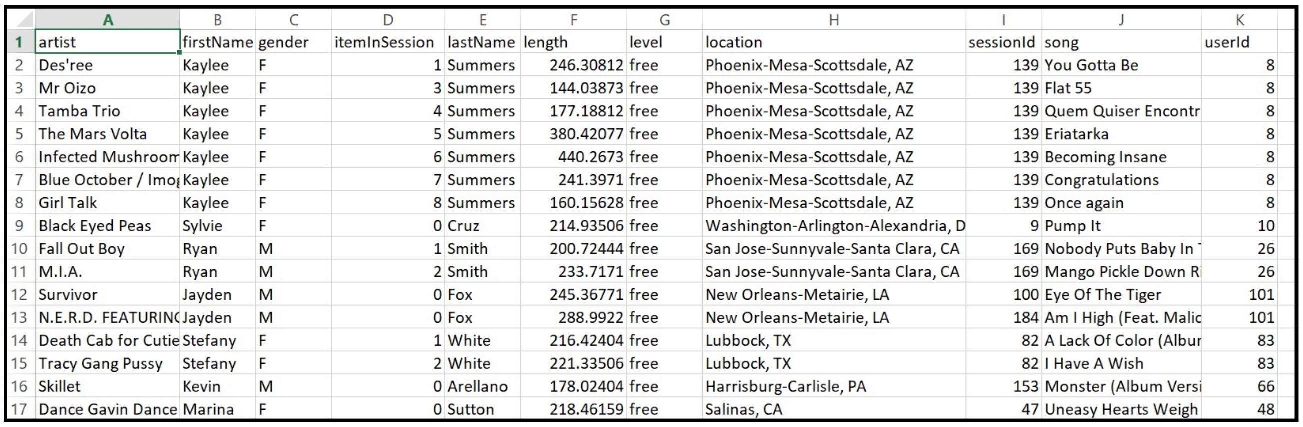
ETL Pipeline for Creating and Querying NoSQL Database.

We need to create a streamlined CSV file from all these. The final file will be used to extract and insert data into Apache Cassandra tables.

The event_datafile_new.csv has 6821 rows and 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:



	A	B	C	D	E	F	G	H	I	J	K
1	artist	firstName	gender	itemInSession	lastName	length	level	location	sessionId	song	userId
2	Des'ree	Kaylee	F	1	Summers	246.30812	free	Phoenix-Mesa-Scottsdale, AZ	139	You Gotta Be	8
3	Mr Oizo	Kaylee	F	3	Summers	144.03873	free	Phoenix-Mesa-Scottsdale, AZ	139	Flat 55	8
4	Tamba Trio	Kaylee	F	4	Summers	177.18812	free	Phoenix-Mesa-Scottsdale, AZ	139	Quem Quiser Encontr	8
5	The Mars Volta	Kaylee	F	5	Summers	380.42077	free	Phoenix-Mesa-Scottsdale, AZ	139	Eriatarka	8
6	Infected Mushroom	Kaylee	F	6	Summers	440.2673	free	Phoenix-Mesa-Scottsdale, AZ	139	Becoming Insane	8
7	Blue October / Imog	Kaylee	F	7	Summers	241.3971	free	Phoenix-Mesa-Scottsdale, AZ	139	Congratulations	8
8	Girl Talk	Kaylee	F	8	Summers	160.15628	free	Phoenix-Mesa-Scottsdale, AZ	139	Once again	8
9	Black Eyed Peas	Sylvie	F	0	Cruz	214.93506	free	Washington-Arlington-Alexandria, D	9	Pump It	10
10	Fall Out Boy	Ryan	M	1	Smith	200.72444	free	San Jose-Sunnyvale-Santa Clara, CA	169	Nobody Puts Baby In	26
11	M.I.A.	Ryan	M	2	Smith	233.7171	free	San Jose-Sunnyvale-Santa Clara, CA	169	Mango Pickle Down R	26
12	Survivor	Jayden	M	0	Fox	245.36771	free	New Orleans-Metairie, LA	100	Eye Of The Tiger	101
13	N.E.R.D. FEATURING	Jayden	M	0	Fox	288.9922	free	New Orleans-Metairie, LA	184	Am I High (Feat. Malic	101
14	Death Cab for Cutie	Stefany	F	1	White	216.42404	free	Lubbock, TX	82	A Lack Of Color (Albur	83
15	Tracy Gang Pussy	Stefany	F	2	White	221.33506	free	Lubbock, TX	82	I Have A Wish	83
16	Skillet	Kevin	M	0	Arellano	178.02404	free	Harrisburg-Carlisle, PA	153	Monster (Album Versi	66
17	Dance Gavin Dance	Marina	F	0	Sutton	218.46159	free	Salinas, CA	47	Uneasy Hearts Weigh	48

Apache Cassandra Coding Portion.

We will model our data based on the queries provided to us by the analytics team at Sparkify. But first, let's setup Apache Cassandra for this. This is a three step process:

Create a Cluster.

We create a cluster and connect it to our local host. This makes a connection to a Cassandra instance on our local machine.

```
# This should make a connection to a Cassandra instance your local machine (127.0.0.
from cassandra.cluster import Cluster

try:
    # Connect to local Apache Cassandra instance.
    cluster = Cluster(['127.0.0.1'])
    # To establish connection and begin executing queries, need a session.
    session = cluster.connect()

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

Create a Keyspace.

```
# Create a keyspace called sparkify.
try:
    session.execute("""
        CREATE KEYSPACE IF NOT EXISTS sparkify
        WITH REPLICATION = {'class': 'SimpleStrategy', 'replication_factor': 1}"""
    )

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

Set Keyspace.

```
# Set KEYSPACE to the keyspace specified above.
try:
    session.set_keyspace("sparkify")

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

Data Modeling.

In Apache Cassandra, we model our data based on the queries we will perform. Aggregation like GROUP BY, JOIN are highly discouraged in Cassandra. This is because we shouldn't scan the entire data because it is distributed on multiple nodes. It will slow down our system because sending all of that data from multiple nodes to a single machine will crash it.

Now we will create the tables to run the following queries:

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. Give me only the following: name of artist, song (sorted by itemInSession) and user (first and last name) for userid = 10, sessionid = 182.
3. Give me every user name (first and last) in my music app history who listened to the song "All Hands Against His Own".

To gain more technical detail about the coding portion please view the ETL notebook.

Conclusion.

This project provides Sparkify startup customers with tools to analyze their data and help answer their key business questions, such as "Which artist and song was heard in a specified session," "Which artist, song and user was heard in a specified session," or "Which users heard a certain song".