Lesson 3 Exercise 2 Primary Key-ANSWER KEY

January 28, 2023

1 Lesson 3 Exercise 2 Solution: Focus on Primary Key

1.0.1 Walk through the basics of creating a table with a good Primary Key in Apache Cassandra, inserting rows of data, and doing a simple CQL query to validate the information.

We will use a python wrapper/ python driver called cassandra to run the Apache Cassandra queries. This library should be preinstalled but in the future to install this library you can run this command in a notebook to install locally: ! pip install cassandra-driver #### More documentation can be found here: https://datastax.github.io/python-driver/

Import Apache Cassandra python package

```
In [1]: import cassandra
```

1.0.2 Create a connection to the database

```
In [2]: from cassandra.cluster import Cluster
        try:
            cluster = Cluster(['127.0.0.1']) #If you have a locally installed Apache Cassandra i
            session = cluster.connect()
        except Exception as e:
            print(e)
```

1.0.3 Create a keyspace to work in

```
In [3]: try:
            session.execute("""
            CREATE KEYSPACE IF NOT EXISTS udacity
            WITH REPLICATION =
            { 'class' : 'SimpleStrategy', 'replication_factor' : 1 }"""
        )

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

Connect to the Keyspace. Compare this to how we had to create a new session in PostgreSQL.

```
In [4]: try:
            session.set_keyspace('udacity')
        except Exception as e:
            print(e)
```

1.0.4 Imagine you need to create a new Music Library of albums

1.0.5 Here is the information asked of the data:

1.0.6 1. Give every album in the music library that was created by a given artist

select * from music_library WHERE artist_name="The Beatles"

1.0.7 Here is the Collection of Data

1.0.8 How should we model these data?

What should be our Primary Key and Partition Key? Since the data are looking for the ARTIST, let's start with that. Is Partitioning our data by artist a good idea? In this case our data is very small. If we had a larger dataset of albums, partitions by artist might be a fine choice. But we would need to validate the dataset to make sure there is an equal spread of the data. `Table Name: music_library column 1: Year column 2: Artist Name column 3: Album Name Column 4: City PRIMARY KEY(artist_name)`

```
In [5]: query = "CREATE TABLE IF NOT EXISTS music_library"
        query = query + "(year int, artist_name text, album_name text, city text, PRIMARY KEY (a
        try:
            session.execute(query)
        except Exception as e:
            print(e)
```

1.0.9 Insert the data into the tables

```
In [6]: query = "INSERT INTO music_library (year, artist_name, album_name, city)"
        query = query + " VALUES (%s, %s, %s, %s)"

        try:
            session.execute(query, (1970, "The Beatles", "Let it Be", "Liverpool"))
        except Exception as e:
            print(e)

        try:
            session.execute(query, (1965, "The Beatles", "Rubber Soul", "Oxford"))
        except Exception as e:
            print(e)

        try:
            session.execute(query, (1965, "The Who", "My Generation", "London"))
        except Exception as e:
            print(e)

        try:
            session.execute(query, (1966, "The Monkees", "The Monkees", "Los Angeles"))
        except Exception as e:
            print(e)

        try:
            session.execute(query, (1970, "The Carpenters", "Close To You", "San Diego"))
        except Exception as e:
            print(e)
```

1.0.10 Let's Validate our Data Model -- Did it work?? If we look for Albums from The Beatles we should expect to see 2 rows.

```
select * from music_library WHERE artist_name="The Beatles"
```

```
In [7]: query = "select * from music_library WHERE artist_name='The Beatles'"
        try:
            rows = session.execute(query)
        except Exception as e:
            print(e)

        for row in rows:
            print (row.year, row.artist_name, row.album_name, row.city)
```

```
(1965, u'The Beatles', u'Rubber Soul', u'Oxford')
```

1.0.11 That didn't work out as planned! Why is that? Because we did not create a unique primary key.

1.0.12 Let's try again. This time focus on making the PRIMARY KEY unique.

1.0.13 Looking at the dataset, what makes each row unique?

1.0.14 We have a couple of options (City and Album Name) but that will not get us the query we need which is looking for album's in a particular artist. Let's make a composite key of the `ARTIST NAME` and `ALBUM NAME`. This is assuming that an album name is unique to the artist it was created by (not a bad bet). --But remember this is just an exercise, you will need to understand your dataset fully (no betting!)

```
In [1]: query = "CREATE TABLE IF NOT EXISTS music_library1 "
        query = query + "(artist_name text, album_name text, year int, city text, PRIMARY KEY (a
        try:
            session.execute(query)
        except Exception as e:
            print(e)
```

```
name 'session' is not defined
```

```
In [9]: query = "INSERT INTO music_library1 (artist_name, album_name, year, city)"
        query = query + " VALUES (%s, %s, %s, %s)"

        try:
            session.execute(query, ("The Beatles", "Let it Be", 1970, "Liverpool"))
        except Exception as e:
            print(e)

        try:
            session.execute(query, ("The Beatles", "Rubber Soul", 1965, "Oxford"))
        except Exception as e:
            print(e)

        try:
            session.execute(query, ("The Who", "My Generation", 1965, "London"))
        except Exception as e:
            print(e)

        try:
            session.execute(query, ("The Monkees", "The Monkees", 1966, "Los Angeles"))
        except Exception as e:
            print(e)

        try:
            session.execute(query, ("The Carpenters", "Close To You", 1970, "San Diego"))
        except Exception as e:
            print(e)
```

1.0.15 Validate the Data Model -- Did it work? If we look for Albums from The Beatles we should expect to see 2 rows.

```
select * from music_library WHERE artist_name="The Beatles"
```

```
In [10]: query = "select * from music_library1 WHERE artist_name='The Beatles'"
         try:
             rows = session.execute(query)
         except Exception as e:
             print(e)

         for row in rows:
             print (row.year, row.artist_name, row.album_name, row.city)
```

```
(1970, u'The Beatles', u'Let it Be', u'Liverpool')
(1965, u'The Beatles', u'Rubber Soul', u'Oxford')
```

1.0.16 Success it worked! We created a unique Primary key that evenly distributed our data.

1.0.17 Drop the tables

```
In [11]: query = "drop table music_library"
         try:
             rows = session.execute(query)
         except Exception as e:
             print(e)

         query = "drop table music_library1"
         try:
             rows = session.execute(query)
         except Exception as e:
             print(e)
```

1.0.18 Close the session and cluster connection

```
In [12]: session.shutdown()
         cluster.shutdown()
```

```
In [ ]:
```