Lesson 3 Demo 2 Primary Key

May 11, 2021

1 Lesson 3 Demo 2: Focus on Primary Key

1.0.1 In this demo we are going to walk through the basics of creating a table with a good Primary Key in Apache Cassandra, inserting rows of data, and doing a simple SQL 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 First let's 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 Let's create a keyspace to do our 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 our 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 Let's imagine we would like to start creating a new Music Library of albums. We are going to work with one of the queries from Exercise 1.

1.0.5 We want to ask 1 question of our data

1. Give me every album in my music library that was released in a given year `select * from music_library WHERE YEAR=1970`

1.0.6 Here is our Collection of Data

1.0.7 How should we model this data? What should be our Primary Key and Partition Key? Since our data is looking for the YEAR let's start with that. Is Partitioning our data by year a good idea? In this case our data is very small, but if we had a larger data set of albums partitions by YEAR might be a find choice. We would need to validate from our dataset. We want 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(year)`

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

1.0.8 Let's insert our data into of table

```
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)
```

1.0.9 Let's Validate our Data Model -- Did it work?? If we look for Albums from 1965 we should expect to see 2 rows.

`select * from music_library WHERE YEAR=1965`

```
In [7]: query = "select * from music_library WHERE YEAR=1965"
        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 The Who My Generation London
```

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

1.0.11 Let's Try Again. Let's focus on making the PRIMARY KEY unique. Look at our dataset do we have anything that is unique for each row? 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 year. Let's make a composite key of the `YEAR` AND `ALBUM NAME`. This is assuming that an album name is unique to the year it was released (not a bad bet). -- But remember this is just a demo, you will need to understand your dataset fully (no betting!)

```
In [8]: query = "CREATE TABLE IF NOT EXISTS music_library1 "
        query = query + "(year int, artist_name text, album_name text, city text, PRIMARY KEY (y
        try:
            session.execute(query)
        except Exception as e:
            print(e)
```

```
In [9]: query = "INSERT INTO music_library1 (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)
```

1.0.12 Let's Validate our Data Model -- Did it work?? If we look for Albums from 1965 we should expect to see 2 rows.

```
select * from music_library WHERE YEAR=1965
```

```
In [10]: query = "select * from music_library1 WHERE YEAR=1965"
         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 The Who My Generation London
1965 The Beatles Rubber Soul Oxford
```

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

1.0.14 For the sake of the demo, I will drop the table.

```
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)
```

1.0.15 And Finally close the session and cluster connection

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

```
In [ ]:
```