Eleftherios Vazaios

BAPT1516

Big Data Systems

Cassandra Assignment

Contents

[Introduction 2](#_Toc456086205)

[Question 1: Users by name 2](#_Toc456086206)

[Question 2: Songs by name 2](#_Toc456086207)

[Question 3: Song History 2](#_Toc456086208)

[Question 4: Playlists by name 2](#_Toc456086209)

[Question 5: Playlists by genre 2](#_Toc456086210)

[Question 6: Playlists by creator 3](#_Toc456086211)

[Question 7: Followers of a playlist 3](#_Toc456086212)

[Question 8: Followers of a user 3](#_Toc456086213)

[Question 9: Songs in a playlist 3](#_Toc456086214)

[Question 10: Times a playlist was played 3](#_Toc456086215)

[Question 11: Times a song was played 3](#_Toc456086216)

[Question 12: Playlists by popularity 3](#_Toc456086217)

[Question 13: Users by popularity 4](#_Toc456086218)

# Introduction

This assignment, implemented using the Cassandra database, aims to produce a structure that supports simple queries for a music streaming service. The structure needs to contain all the necessary information about users, songs and playlists as well as answer everyday questions that might occur.

This was achieved using the Cassandra Query Language (or simply CQL), having the characteristic property of creating separate tables to answer specific queries. The purpose of the current report is to describe the structure and provide insight as to the attributes and data types selected.

Note that the tables are created with the assumption that some appropriate tool will be used for accurate data insertion and table cooperation. In the current form, the tables do not guarantee integrity but this is the nature of Cassandra.

# Question 1: Users by name

The only information needed here are basic demographics for each user of our service. Therefore, besides their username (also the table’s primary key since it is unique), the structure also contains an address (simple text type as we don’t really need the separate components for further analysis) as well as the payment information of the user. The latter can be studied further. To accommodate that, a new column type was created, containing payment option (essentially card type), card number and current balance.

# Question 2: Songs by name

For the songs stored in the table in question, we need to keep data on name, artist, album, genre, year as well as the song itself. The latter, being a large binary file, can only be stored as a blob type (up to 2GB). Furthermore, since a song name is hardly unique in such a large streaming service, a combined primary key was created from the columns name and artist.

# Question 3: Song History

Here, we need to project the history for a specific user. The columns needed are the username, song name, its artist and the timestamp when the song was listened to. Since a specific user can only listen to 1 song at a specific time, the columns username and time that the song was heard will form the table’s key. Since we want to order the results by starting from the most recent, the parameter “clustering order by” was used on the time that the song was played, with the value “DESC” to display in descending order.

# Question 4: Playlists by name

For this question we simply need each playlist with a short description. Since its name is unique, it can be used as a primary key.

# Question 5: Playlists by genre

This question is quite similar to the previous one, except we need to execute queries based on the genre of each playlist. In order to add this column to the “WHERE” clause, it needs to be part of the table’s primary key. Apart from that, we also need to create an index on it (preferable) in order to avoid the warning about all filtering (and the option to turn it on which is not recommended).

# Question 6: Playlists by creator

This is handled in the exactly the same as the previous question, by replacing the genre with the playlist’s creator.

# Question 7: Followers of a playlist

In order to display the followers for each playlist we only need two columns, the username and the playlist name. They both need to be used as key, since it is a many to many relationship (and a user cannot follow a playlist twice).

# Question 8: Followers of a user

Exactly the same as the previous question, only replacing playlist name with username.

# Question 9: Songs in a playlist

To achieve a setup combining songs and playlists we simply need the name of the playlist and the song, as well as the artist (since the song name is not unique). All those columns need to be contained in the primary key, since a playlist can contain many songs, even some with the same name.

# Question 10: Times a playlist was played

For this query we simply need a table storing the unique playlist name (also primary key), as well as the times it was listened to by any user. This number will be updated whenever someone listens to any song in the playlist.

# Question 11: Times a song was played

Similar to the previous question, we have a table that stores the song’s name, the artist and the play count. Once again, the primary key is the combination of song name and artist. An index was also created on the latter, so as to allow queries that involve both the song’s name and its artist (in the case of coinciding names for instance).

# Question 12: Playlists by popularity

For this question it was required that we order the playlists available by their popularity in descending order. Popularity is defined by number of followers. The columns needed are playlist name and number of followers. We would normally use both in a combined primary key and set clustering order to descending for the popularity column.

However, this does not solve the issue since ordering happens after the rows have been partitioned based on the playlist name. Because of that the ordering parameter is useless. What we want is to store all rows in a single partition (which can accommodate up to 2 billion records), so that sorting is actually meaningful. This can be achieved by using a constant variable and setting it as the primary key along with the others. Assuming that this variable will always be 1 for example and partitioning will happen based on that, all our rows will end up in the same partition. Owing to that, the clustering order parameter is actually used and we manage to get the desired outcome.

Note that the dummy inserts showing the results are also provided for this question (and the next one), unlike the previous ones.

# Question 13: Users by popularity

Much like the previous question, we now want to see a list of users sorted by decreasing popularity. Using the same principle and another constant (again set to 1 for all inserts) we get the desired result. Apart from that constant, the columns used (also present in the primary key) are username and number of followers.