ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

BUSINESS ANALYTICS
Master of Science

Big Data Systems – Cassandra Assignment

2015-2016

Assignment Supervisor:
Associate Prof. Louridas Panagiotis

Author:
Konstantinos Chronis BAFT1502

Date:
Athens, 14/07/2016

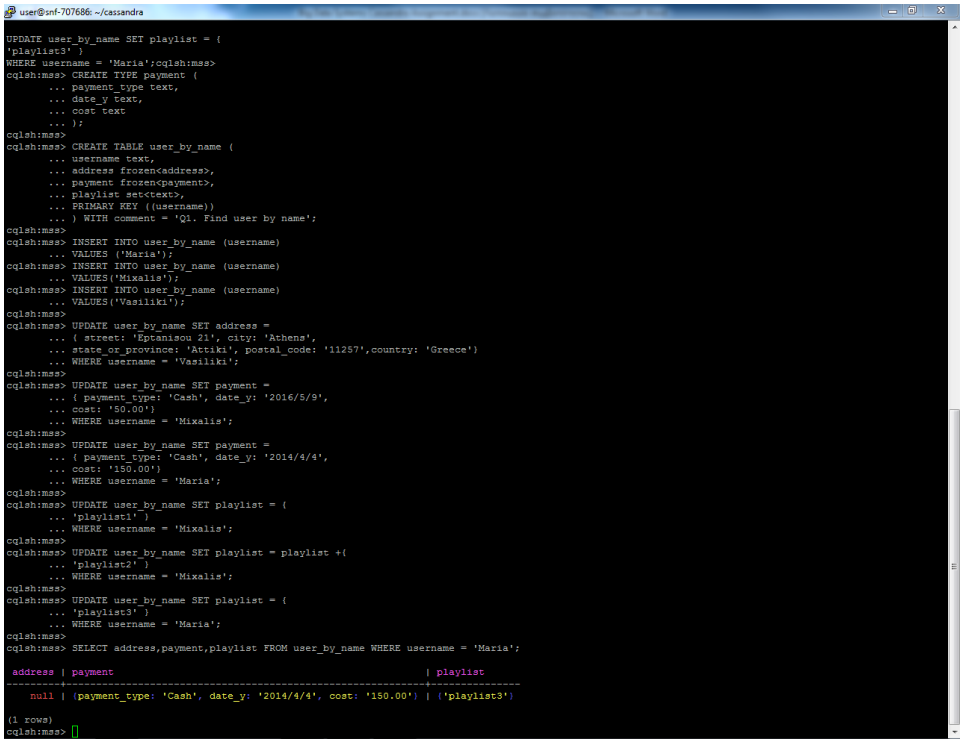
Introduction

In this assignment we were asked to design a Cassandra database for a music streaming service. The documentation of the database design can be found below, while the straight CQL code file containing the table creation and the queries can be found in the music.cql file.

Documentation(with screenshots)

Query 1:

The table **user_by_name** was created for the purposes of this query containing all the columns that was asked from us for the description of the user for this assignment. The primary key was set as the name itself of the user.



```
user@snf-707686: ~/cassandra

UPDATE user_by_name SET playlist = {
'playlist3' }
WHERE username = 'Maria';cqlsh:mss>
cqlsh:mss> CREATE TYPE payment (
       ... payment_type text,
       ... date_y text,
       ... cost text
       ... );
cqlsh:mss>
cqlsh:mss> CREATE TABLE user_by_name (
       ... username text,
       ... address frozen<address>,
       ... payment frozen<payment>,
       ... playlist set<text>,
       ... PRIMARY KEY ((username))
       ... ) WITH comment = 'Q1. Find user by name';
cqlsh:mss>
cqlsh:mss> INSERT INTO user_by_name (username)
       ... VALUES ('Maria');
cqlsh:mss> INSERT INTO user_by_name (username)
       ... VALUES('Mixalis');
cqlsh:mss> INSERT INTO user_by_name (username)
       ... VALUES('Vasiliki');
cqlsh:mss>
cqlsh:mss> UPDATE user_by_name SET address =
       ... { street: 'Ēptanisou 21', city: 'Athens',
       ... state_or_province: 'Attiki', postal_code: '11257',country: 'Greece'}
       ... WHERE username = 'Vasiliki';
cqlsh:mss>
cqlsh:mss> UPDATE user_by_name SET payment =
       ... { payment_type: 'Cash', date_y: '2016/5/9',
       ... cost: '50.00'}
       ... WHERE username = 'Mixalis';
cqlsh:mss>
cqlsh:mss> UPDATE user_by_name SET payment =
       ... { payment_type: 'Cash', date_y: '2014/4/4',
       ... cost: '150.00'}
       ... WHERE username = 'Maria';
cqlsh:mss>
cqlsh:mss> UPDATE user_by_name SET playlist = {
       ... 'playlist1' }
       ... WHERE username = 'Mixalis';
cqlsh:mss>
cqlsh:mss> UPDATE user_by_name SET playlist = playlist +{
       ... 'playlist2' }
       ... WHERE username = 'Mixalis';
cqlsh:mss>
cqlsh:mss> UPDATE user_by_name SET playlist = {
       ... 'playlist3' }
       ... WHERE username = 'Maria';
cqlsh:mss>
cqlsh:mss> SELECT address,payment,playlist FROM user_by_name WHERE username = 'Maria';

 address | payment                                                          | playlist
---------+------------------------------------------------------------------+---------------
    null | {payment_type: 'Cash', date_y: '2014/4/4', cost: '150.00'} | {'playlist3'}

(1 rows)
cqlsh:mss>
```

Query 2:

The table **song_by_name** was created for the second query containing again all the columns that was asked from us for the description of the song for this assignment. The primary key was set as the name itself of the song.

```
cqlsh:mss> //*****************************************************************************************************
cqlsh:mss> //Query 2 - Find song by name
cqlsh:mss> //*****************************************************************************************************
cqlsh:mss> CREATE TABLE song_by_name (
       ... songname text,
       ... artists set<text> ,
UPDATE song_by_name SET artists = {
       ... album_name text,
WHERE songname = 'Numb';
       ... year text,
       ... play_count text,
       ... song_binary text,
       ... PRIMARY KEY ((songname))
WHERE songname = 'What I have done';

       ... )WITH comment = 'Q2. Find song by name';

//SELECT * FROM song_by_name;

SELECT artists,album_name,year,play_count FROM song_by_name WHERE songname = 'Clocks';cqlsh:mss>
cqlsh:mss> INSERT INTO song_by_name (songname,album_name,year,play_count,song_binary)
       ... VALUES ('Numb', 'Numb', '2011', '85000000','00111101');
cqlsh:mss>
cqlsh:mss> INSERT INTO song_by_name (songname,album_name,year,play_count,song_binary)
       ... VALUES ('What I have done', 'Numb', '2011', '100000000','00101101');
cqlsh:mss>
cqlsh:mss> INSERT INTO song_by_name (songname,album_name,year,play_count,song_binary)
       ... VALUES ('Clocks', 'Clocks', '2010', '44000000','11111101');
cqlsh:mss>
cqlsh:mss> UPDATE song_by_name SET artists = {
       ... 'Linkin Park' }
       ... WHERE songname = 'Numb';
cqlsh:mss>
cqlsh:mss>
cqlsh:mss> UPDATE song_by_name SET artists = {
       ... 'Linkin Park' }
       ... WHERE songname = 'What I have done';
cqlsh:mss>
cqlsh:mss> UPDATE song_by_name SET artists = {
       ... 'Coldplay' }
       ... WHERE songname = 'Clocks';
cqlsh:mss>
cqlsh:mss> //SELECT * FROM song_by_name;
cqlsh:mss>
cqlsh:mss> SELECT artists,album_name,year,play_count FROM song_by_name WHERE songname = 'Clocks';

 artists      | album_name | year | play_count
--------------+------------+------+------------
 {'Coldplay'} |     Clocks | 2010 |   44000000

(1 rows)
cqlsh:mss>
```

Query 3:

The table **song_by_user** was created for the third query containing the name of the user, the name of the song and the date the user played the song. The primary key was set as the username along with the name/names of the song/songs. Each time you write data into Cassandra, a timestamp is generated for each column value that is updated. Internally, Cassandra uses these timestamps for resolving any conflicting changes that are made to the same value. Generally, the last timestamp wins. Thus ensures that the results of the query are given in reverse chronological order.

```
cqlsh:mss> //*****************************************************************************************************
cqlsh:mss> //Query 3 - Find songs played by a user, arranged in reverse chronological order
PRIMARY KEY ((username),songname)
)WITH comment = 'Q3. Find songs played by a user, arranged in reverse chronological order';
cqlsh:mss> //*****************************************************************************************************
cqlsh:mss> CREATE TABLE songs_by_user (
INSERT INTO songs_by_user (username,songname,song_date)
       ... username text,
       ... songname text,
       ... song_date text,
       ... PRIMARY KEY ((username),songname)
       ... )WITH comment = 'Q3. Find songs played by a user, arranged in reverse chronological order';

cqlsh:mss>
cqlsh:mss> INSERT INTO songs_by_user (username,songname,song_date)
       ... VALUES ('Vasiliki', 'Clocks', '2012/4/3');
cqlsh:mss>
cqlsh:mss> INSERT INTO songs_by_user (username,songname,song_date)
       ... VALUES ('Maria', 'Numb', '2014/6/9');
cqlsh:mss>
cqlsh:mss> INSERT INTO songs_by_user (username,songname,song_date)
       ... VALUES ('Maria', 'What I have done', '2013/5/4');
cqlsh:mss>
cqlsh:mss> INSERT INTO songs_by_user (username,songname,song_date)
       ... VALUES ('Maria', 'Clocks', '2016/8/10');
cqlsh:mss>
cqlsh:mss> //SELECT * FROM songs_by_user;
cqlsh:mss>
cqlsh:mss> SELECT songname,song_date,writetime(song_date) FROM songs_by_user WHERE username = 'Maria';

 songname         | song_date | writetime(song_date)
------------------+-----------+----------------------
           Clocks | 2016/8/10 |     1468584407621174
             Numb |  2014/6/9 |     1468584407594156
 What I have done |  2013/5/4 |     1468584407607856

(3 rows)
cqlsh:mss>
```

Query 4:

The table **playlist_by_name** was created for the query 4 containing the name of the playlist, its description along with the name of the user and the names of the songs. The primary key was set as the name of the playlist only because it alone must be unique.

```
cqlsh:mss> //Query 4 - Find playlist by name
cqlsh:mss> //*****************************************************************************************
cqlsh:mss> CREATE TABLE playlist_by_name (

       ... playlistname text,
       ... description text,
       ... username text,
       ... songname set<text>,
       ... PRIMARY KEY ((playlistname))
VALUES ('playlist2', 'rock playlist', 'Maria');
       ... )WITH comment = 'Q4. Find playlist by name';

UPDATE playlist_by_name SET songname = {'Clocks'}
WHERE playlistname = 'playlist1';

UPDATE playlist_by_name SET songname = {'What I have done'}
WHERE playlistname = 'playlist2';

UPDATE playlist_by_name SET songname = {'Numb','Clocks'}
WHERE playlistname = 'playlist1';

//SELECT * FROM playlist_by_name;

SELECT username, songname, description FROM playlist_by_name WHERE playlistname = 'playlist1';cqlsh:mss>
cqlsh:mss> INSERT INTO playlist_by_name (playlistname,description,username)
       ... VALUES ('playlist1', 'chill playlist', 'Maria');
cqlsh:mss>
cqlsh:mss> INSERT INTO playlist_by_name (playlistname,description,username)
       ... VALUES ('playlist1', 'rock playlist', 'Mixalis');
cqlsh:mss>
cqlsh:mss> INSERT INTO playlist_by_name (playlistname,description,username)
       ... VALUES ('playlist2', 'rock playlist', 'Maria');
cqlsh:mss>
cqlsh:mss> UPDATE playlist_by_name SET songname = {'Clocks'}
       ... WHERE playlistname = 'playlist1';
cqlsh:mss>
cqlsh:mss> UPDATE playlist_by_name SET songname = {'What I have done'}
       ... WHERE playlistname = 'playlist2';
cqlsh:mss>
cqlsh:mss> UPDATE playlist_by_name SET songname = {'Numb','Clocks'}
       ... WHERE playlistname = 'playlist1';
cqlsh:mss>
cqlsh:mss> //SELECT * FROM playlist_by_name;
cqlsh:mss>
cqlsh:mss> SELECT username, songname, description FROM playlist_by_name WHERE playlistname = 'playlist1';

 username | songname           | description
----------+--------------------+-------------
  Mixalis | {'Clocks', 'Numb'} | rock playlist

(1 rows)
cqlsh:mss>
```

Query 5:

For the fifth query the table **playlist_by_genre** was created containing the genre of the playlist, its name, its description along with the names of the songs. The name of the user is not needed for this one. The primary key was set as the genre combined with the name of the playlist because that mixture has to be unique.

```
cqlsh:mss> //*****************************************************************************************
cqlsh:mss> //Query 5 - Find playlist by genre
cqlsh:mss> //*****************************************************************************************
cqlsh:mss> CREATE TABLE playlist_by_genre (
       ... genre text,
UPDATE playlist_by_genre SET songname = {'Clocks'}
       ... playlistname text,

       ... description text,

       ... songname set<text>,
       ... PRIMARY KEY ((genre),playlistname)
       ... )WITH comment = 'Q5. Find playlist by genre';
cqlsh:mss>
cqlsh:mss> INSERT INTO playlist_by_genre (genre,playlistname,description)
       ... VALUES ('Rock', 'playlist1' ,'chill playlist');
cqlsh:mss>
cqlsh:mss> INSERT INTO playlist_by_genre (genre,playlistname,description)
       ... VALUES ('Pop', 'playlist1' ,'dance playlist');
cqlsh:mss>
cqlsh:mss> INSERT INTO playlist_by_genre (genre,playlistname,description)
       ... VALUES ('Rnb', 'playlist1' ,'chill playlist');
cqlsh:mss>
cqlsh:mss> INSERT INTO playlist_by_genre (genre,playlistname,description)
       ... VALUES ('Rock', 'playlist2' ,'rock playlist');
cqlsh:mss>
cqlsh:mss> UPDATE playlist_by_genre SET songname = {'Clocks'}
       ... WHERE genre = 'Rock' AND playlistname = 'playlist2';
cqlsh:mss>
cqlsh:mss> //SELECT * FROM playlist_by_genre;
cqlsh:mss>
cqlsh:mss> SELECT playlistname, description, songname FROM playlist_by_genre WHERE genre = 'Rock';

 playlistname | description    | songname
--------------+----------------+----------
    playlist1 | chill playlist |     null
    playlist2 |  rock playlist | {'Clocks'}

(2 rows)
cqlsh:mss>
```

Query 6:

The table **playlist_by_user** was created for the query 6 containing the name of the playlist, its description along with the name of the user and the names of the songs. This is similar to the table created for query 4 but in this case the primary key has to be set as the combination of the name of the playlist with the name of the creator/user. Apart from that,

in Cassandra we can only do queries on columns that are part of the primary key. If we want to do a query in another column, we have to create a secondary index. In our case, an index is needed in the username column for the query to be done correctly.

```
cqlsh:mss> //*****************************************************************************************************
cqlsh:mss> //Query 6 - Find playlist by creator
cqlsh:mss> //*****************************************************************************************************
cqlsh:mss> CREATE TABLE playlist_by_user (
       ... playlistname text,
WHERE playlistname = 'playlist1' AND username = 'Maria';
       ... description text,
UPDATE playlist_by_user SET songname = {'Numb'}
       ... username text,

       ... songname set<text>,
WHERE playlistname = 'playlist2' AND username = 'Mixalis';
       ... PRIMARY KEY ((playlistname),username)
       ... )WITH comment = 'Q6. Find playlist by creator';
cqlsh:mss>
cqlsh:mss> INSERT INTO playlist_by_user (playlistname,description,username)
       ... VALUES ('playlist1', 'rock playlist', 'Mixalis');
cqlsh:mss>
cqlsh:mss> INSERT INTO playlist_by_user (playlistname,description,username)
       ... VALUES ('playlist2', 'pop playlist', 'Mixalis');
cqlsh:mss>
cqlsh:mss> INSERT INTO playlist_by_user (playlistname,description,username)
       ... VALUES ('playlist1', 'rnb playlist', 'Maria');
cqlsh:mss>
cqlsh:mss> UPDATE playlist_by_user SET songname = {'Clocks'}
       ... WHERE playlistname = 'playlist1' AND username = 'Maria';
cqlsh:mss>
cqlsh:mss> UPDATE playlist_by_user SET songname = {'Numb'}
       ... WHERE playlistname = 'playlist1' AND username = 'Mixalis';
cqlsh:mss>
cqlsh:mss> UPDATE playlist_by_user SET songname = {'What I have done','Numb'}
       ... WHERE playlistname = 'playlist2' AND username = 'Mixalis';
cqlsh:mss>
cqlsh:mss> CREATE INDEX ON playlist_by_user (username);
cqlsh:mss>
cqlsh:mss> SELECT playlistname, description, songname FROM playlist_by_user WHERE username = 'Mixalis';

 playlistname | description   | songname
--------------+---------------+-------------------------------
    playlist2 |  pop playlist | {'Numb', 'What I have done'}
    playlist1 | rock playlist |                     {'Numb'}

(2 rows)
cqlsh:mss>
```

Query 7:

The table **<u>followers_by_playlist</u>** was created for the query 7 containing the names of the followers, the name of the playlist and its description. As for the unique key it was set as the combination of the follower and the name of the playlist. Again here an index was needed on the playlistname column for the query to be implemented correctly.

```
cqlsh:mss> //*****************************************************************************************************
cqlsh:mss> //Query 7 - Find the followers of a playlist
INSERT INTO followers_by_playlist (follower,playlistname,description)
cqlsh:mss> //*****************************************************************************************************
cqlsh:mss> CREATE TABLE followers_by_playlist (
       ... follower text,
       ... playlistname text,
       ... description text,
       ... PRIMARY KEY ((follower),playlistname)
       ... )WITH comment = 'Q7. Find the followers of a playlist';
cqlsh:mss>
cqlsh:mss> INSERT INTO followers_by_playlist (follower,playlistname,description)
       ... VALUES ('Vasiliki','playlist1','rock playlist');
cqlsh:mss>
cqlsh:mss> INSERT INTO followers_by_playlist (follower,playlistname,description)
       ... VALUES ('Vasiliki','playlist2','chill playlist');
cqlsh:mss>
cqlsh:mss> INSERT INTO followers_by_playlist (follower,playlistname,description)
       ... VALUES ('Maria','playlist1','rock playlist');
cqlsh:mss>
cqlsh:mss> CREATE INDEX ON followers_by_playlist (playlistname);
cqlsh:mss>
cqlsh:mss> SELECT follower FROM followers_by_playlist WHERE playlistname = 'playlist1';

 follower
----------
    Maria
 Vasiliki

(2 rows)
cqlsh:mss>
```

Query 8:

For the eighth query the table **<u>followers_by_user</u>** was created containing just the name of the follower and the name of the user. The primary key was set as the combination of these two columns of this table to ensure uniqueness. Again here, we had to place an index on just the user name for the purposes of this query.

```
cqlsh:mss> //*****************************************************************************************
cqlsh:mss> //Query 8 - Find the followers of a user
cqlsh:mss> //*****************************************************************************************
cqlsh:mss> CREATE TABLE followers_by_user (
       ... follower text,
       ... username text,
       ... PRIMARY KEY ((follower),username)
       ... )WITH comment = 'Q8. Find the followers of a user';
cqlsh:mss>
cqlsh:mss> INSERT INTO followers_by_user (follower,username)
       ... VALUES ('Maria','Vasiliki');
cqlsh:mss>
cqlsh:mss> INSERT INTO followers_by_user (follower,username)
       ... VALUES ('Maria','Mixalis');
cqlsh:mss>
cqlsh:mss> INSERT INTO followers_by_user (follower,username)
       ... VALUES ('Mixalis','Vasiliki');
cqlsh:mss>
cqlsh:mss> CREATE INDEX ON followers_by_user (username);
cqlsh:mss>
cqlsh:mss> SELECT follower FROM followers_by_user WHERE username = 'Mixalis';

 follower
----------
    Maria

(1 rows)
cqlsh:mss>
```

Query 9:

For query 9 it was noticed that no extra table should be created because the information required by this specific query is all contained in table playlist_by_name that was created for query 4.

```
cqlsh:mss> //*****************************************************************************************
cqlsh:mss> //Query 9 - Find the songs contained in a playlist
cqlsh:mss> //*****************************************************************************************
cqlsh:mss> SELECT  songname FROM playlist_by_name WHERE playlistname = 'playlist1';

 songname
--------------------
 {'Clocks', 'Numb'}

(1 rows)
cqlsh:mss>
```

Query 10:

The table **count_by_playlist** was created for the query 10 containing the name of the playlist and the count of its times played. As for the unique key it was set as only the name of the playlist.

```
cqlsh:mss> //*****************************************************************************************
cqlsh:mss> //Query 10 - Find how many times a playlist has been played
cqlsh:mss> //*****************************************************************************************
cqlsh:mss> CREATE TABLE count_by_playlist (
       ... playlistname text,
       ... play_count int,
       ... PRIMARY KEY ((playlistname))
       ... )WITH comment = 'Q10. Find how many times a playlist has been played';
cqlsh:mss>
cqlsh:mss> INSERT INTO count_by_playlist (playlistname,play_count)
       ... VALUES ('playlist1',100000000);
cqlsh:mss>
cqlsh:mss> INSERT INTO count_by_playlist (playlistname,play_count)
       ... VALUES ('playlist2',45000000);
cqlsh:mss>
cqlsh:mss> INSERT INTO count_by_playlist (playlistname,play_count)
       ... VALUES ('playlist3',48000000);
cqlsh:mss>
cqlsh:mss> //SELECT * FROM count_by_playlist;
cqlsh:mss>
cqlsh:mss> SELECT play_count AS times_played FROM count_by_playlist WHERE playlistname='playlist1';

 times_played
--------------
    100000000

(1 rows)
cqlsh:mss>
```

Query 11:

The table **count_by_song** was created for the query 11 containing the name of the song and the count of its times played. As for the unique key it was set as only the name of the song itself.

```
cqlsh:mss> //*****************************************************************************************
cqlsh:mss> //Query 11 - Find how many times a song has been played

cqlsh:mss> //*****************************************************************************************
cqlsh:mss> CREATE TABLE count_by_song (
       ... songname text,
       ... play_count int,
       ... PRIMARY KEY ((songname))
       ... )WITH comment = 'Q11. Find how many times a song has been played';
cqlsh:mss>
cqlsh:mss> INSERT INTO count_by_song (songname,play_count)
       ... VALUES ('Clocks',105000000);
cqlsh:mss>
cqlsh:mss> INSERT INTO count_by_song (songname,play_count)
       ... VALUES ('Numb',74000000);
cqlsh:mss>
cqlsh:mss> //SELECT * FROM count_by_song;
cqlsh:mss>
cqlsh:mss> SELECT play_count AS times_played FROM count_by_song WHERE songname='Numb';

 times_played
--------------
     74000000

(1 rows)
cqlsh:mss>
```

Query 12:

A materialized view is a table that is built from another table's data with a new primary key specified. In Cassandra, queries are optimized by primary key definition and often there is a table per query. If a new query is desired, a new table is created. Materialized views update and delete values when the original table is updated and deleted. Materialized views can help us in responding to queries to our data.

For our assignment, a materialized view was created to store the data of the table count_by_playlist of the tenth query. As for the view's primary key, the same primary key that was used for the table is set here.

```
cqlsh:mss> //*****************************************************************************************
cqlsh:mss> //Query 12 - List playlists in decreasing popularity; the popularity of a playlist is
cqlsh:mss> //defined by the number of times it has been played
cqlsh:mss> //*****************************************************************************************
cqlsh:mss> CREATE MATERIALIZED VIEW playlist_rank AS
       ... SELECT play_count FROM count_by_playlist
       ... WHERE playlistname IS NOT NULL
       ... AND play_count IS NOT NULL
       ... PRIMARY KEY ((playlistname),play_count)
       ... WITH CLUSTERING ORDER BY (play_count DESC);
cqlsh:mss>
cqlsh:mss> SELECT * FROM playlist_rank;

 playlistname | play_count
--------------+------------
    playlist2 |   45000000
    playlist3 |   48000000
    playlist1 |  100000000

(3 rows)
cqlsh:mss>
```

In the example table above, queries will return data that is sorted according to the clustering key(s) only when a partition key is also specified. Without a partition key specified in the WHERE clause, the actual order of the result set then becomes dependent on the hashed values of playlistname. This is apparent when follower is queried without specifying a WHERE clause and using the token() function on playlistname.

Query 13:

The **COUNTER** table **<u>user_rank</u>** was created for the query 13 containing the name of the follower and the name of the user and a counter value of the follows. As for the unique key it was set as the combination of the follower and the name of the user.

```
cqlsh:mss> //*****************************************************************************************
cqlsh:mss> //Query 13 - List users in decreasing popularity: the popularity of a user is defined
cqlsh:mss> //by the number of followers
cqlsh:mss> //*****************************************************************************************
cqlsh:mss>
cqlsh:mss> CREATE TABLE user_rank (
WHERE username ='Maria' AND follower ='Kostas';
        ... follower text,
UPDATE user_rank
        ... username text,
WHERE username ='Vasiliki' AND follower ='Mixalis';
        ... counter_value counter,
        ... PRIMARY KEY ((follower),username)

        ... )WITH comment = 'Q13. List users in decreasing popularity';
cqlsh:mss>
cqlsh:mss> UPDATE user_rank
        ... SET counter_value = counter_value + 1
        ... WHERE username ='Vasiliki' AND follower ='Maria';
cqlsh:mss>
cqlsh:mss> UPDATE user_rank
        ... SET counter_value = counter_value + 1
        ... WHERE username ='Vasiliki' AND follower ='Kostas';
cqlsh:mss>
cqlsh:mss> UPDATE user_rank
        ... SET counter_value = counter_value + 1
        ... WHERE username ='Mixalis' AND follower ='Maria';
cqlsh:mss>
cqlsh:mss> UPDATE user_rank
        ... SET counter_value = counter_value + 1
        ... WHERE username ='Maria' AND follower ='Kostas';
cqlsh:mss>
cqlsh:mss> UPDATE user_rank
        ... SET counter_value = counter_value + 1
        ... WHERE username ='Vasiliki' AND follower ='Mixalis';
cqlsh:mss>
cqlsh:mss>
cqlsh:mss> SELECT * FROM user_rank;

 follower | username | counter_value
----------+----------+---------------
  Mixalis | Vasiliki |             1
   Kostas |    Maria |             1
   Kostas | Vasiliki |             1
    Maria |  Mixalis |             1
    Maria | Vasiliki |             1

(5 rows)
cqlsh:mss>
cqlsh:mss> SELECT COUNT(*) FROM user_rank WHERE username = 'Vasiliki' ALLOW FILTERING;

 count
-------
     3

(1 rows)

Warnings :
Aggregation query used without partition key
```

Important Note:

It was noticed that when run the command inside the cqlsh “SOURCE ‘music.cql’;” queries 6, 7, 8 encountered consistency problems. On the contrary when inside the keyspace mss the queries were implemented with no problem.

End of Assignment. Thank you for your time!