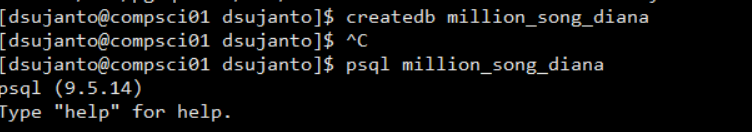
**Diana Sujanto**

**CAP 5322 - Midterm**

1. Create database : million\_song\_diana on compsci01



Five tables are created: artists, artist\_term, cities, similarity, tracks

SQL scripts:

CREATE TABLE artists(

artist\_id character varying (50) primary key,

artist\_name text,

lat real,

lng real) ;

CREATE TABLE artist\_term (

artist\_term\_id serial primary key,

artist\_id character varying (50),

term character varying(50),

FOREIGN KEY (artist\_id) REFERENCES artists(artist\_id));

CREATE TABLE cities(

zip character(5),

state character(2),

city character(50),

lat real,

lng real)

CREATE TABLE similarity (

target character varying (50),

similarity character varying (50))

CREATE TABLE tracks (

track\_id character varying (50) primary key,

title text,

song\_id character varying(50),

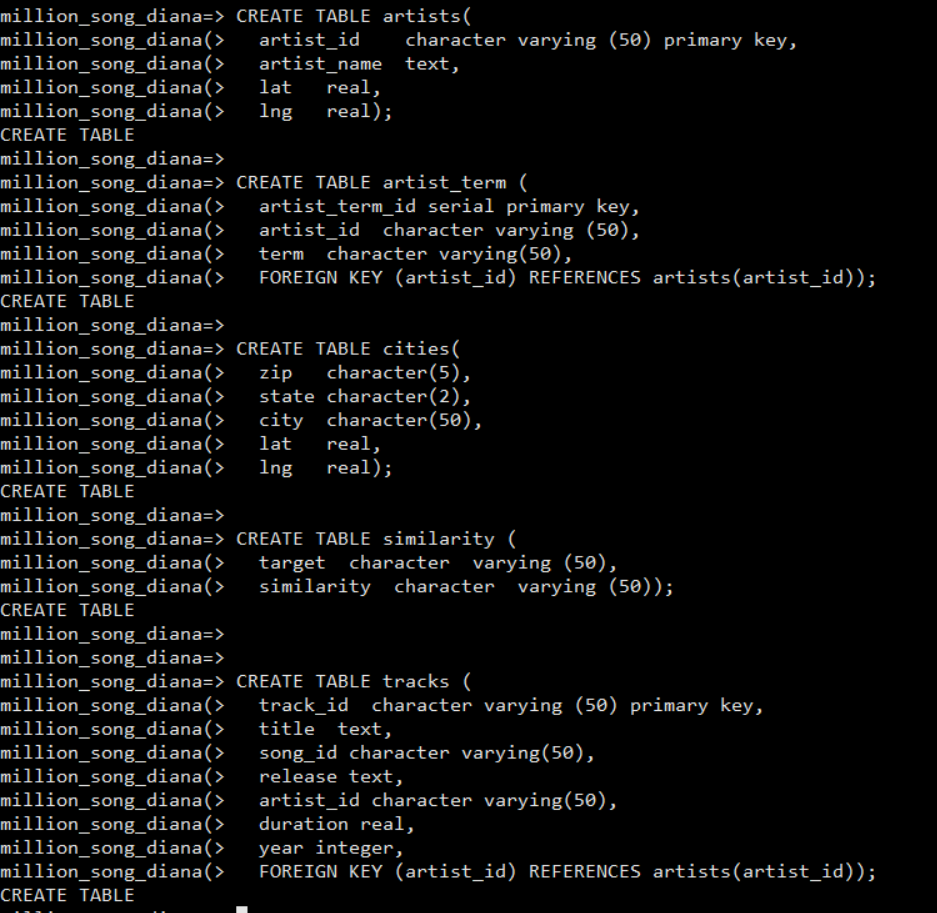
release text,

artist\_id character varying(50),

duration real,

year integer,

FOREIGN KEY (artist\_id) REFERENCES artists(artist\_id))



Import .csv files:

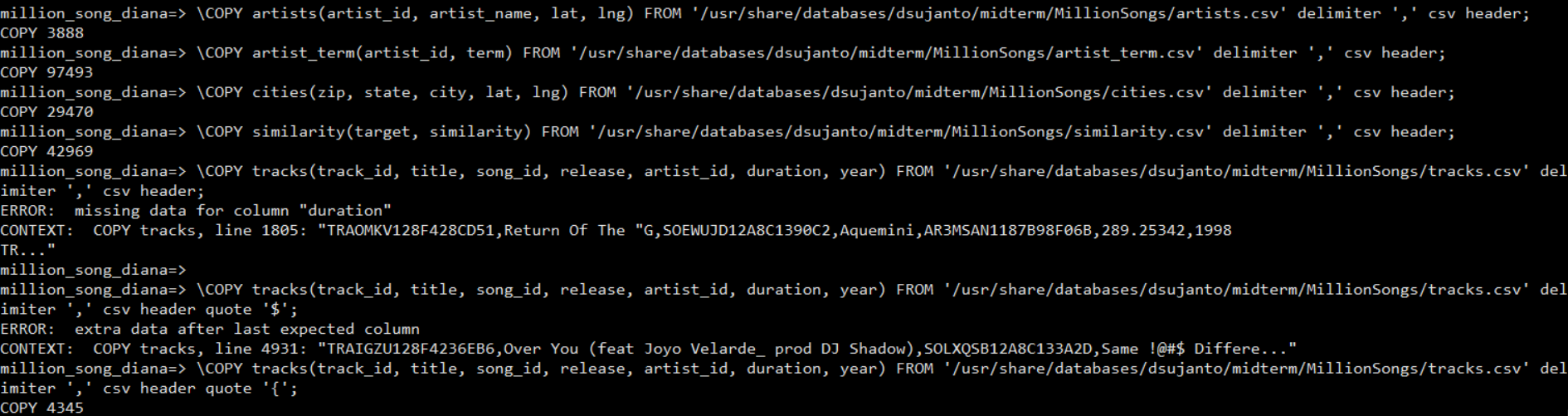
\COPY artists(artist\_id, artist\_name, lat, lng) FROM '/usr/share/databases/dsujanto/midterm/MillionSongs/artists.csv' delimiter ',' csv header;

\COPY artist\_term(artist\_id, term) FROM '/usr/share/databases/dsujanto/midterm/MillionSongs/artist\_term.csv' delimiter ',' csv header;

\COPY cities(zip, state, city, lat, lng) FROM '/usr/share/databases/dsujanto/midterm/MillionSongs/cities.csv' delimiter ',' csv header;

\COPY similarity(target, similarity) FROM '/usr/share/databases/dsujanto/midterm/MillionSongs/similarity.csv' delimiter ',' csv header;

\COPY tracks(track\_id, title, song\_id, release, artist\_id, duration, year) FROM '/usr/share/databases/dsujanto/midterm/MillionSongs/tracks.csv' delimiter ',' csv header;



Since I had trouble importing tracks.csv, I used ‘$’ in Copy statement:

\COPY tracks(track\_id, title, song\_id, release, artist\_id, duration, year) FROM '/usr/share/databases/dsujanto/midterm/MillionSongs/tracks.csv' delimiter ',' csv header quote ‘$’;

But it still did not work because ‘$ was in the .csv file. So, I used the ‘{‘.

\COPY tracks(track\_id, title, song\_id, release, artist\_id, duration, year) FROM '/usr/share/databases/dsujanto/midterm/MillionSongs/tracks.csv' delimiter ',' csv header quote '{';

Also, I updated the year = 0 to year = null in tracks table:



1. A new table called “track\_votes” with 3 columns (user\_id character(20), track\_id character varying(50), danceability boolean) can be created to store user votes.

A column will be added to the tracks table: danceability with Boolean datatype

After insert trigger can be used to accommodate these changes:

Once a user entered the vote, this ***after insert trigger*** will be executed to update the danceability column in tracks table. If the song is danceable, danceability = TRUE, otherwise danceability = FALSE.

CREATE TRIGGER danceability

After insert on track\_votes

For each row

Update tracks

Set danceability = track\_votes.danceability

Where track\_id = track\_votes.track\_id

1. A. Added the primary key to the table already

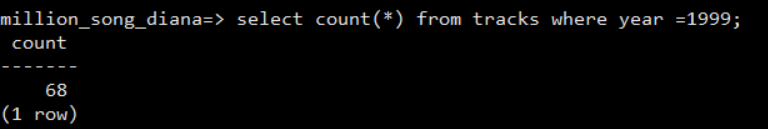
A select statement that uses the primary key index, and show that it does in fact use the index:

explain select \* from artist\_term where artist\_term\_id = 100;

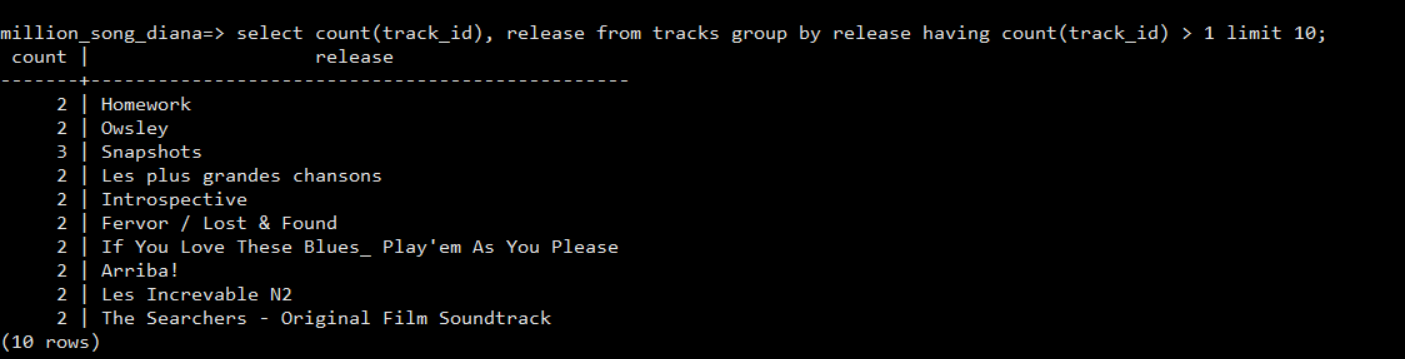


B. The ‘term’ would be a song attribute in my ideal song database because it describes the type of the song rather than the artist. It violates third normal form where term does not depend on the artist directly but depends on the song.

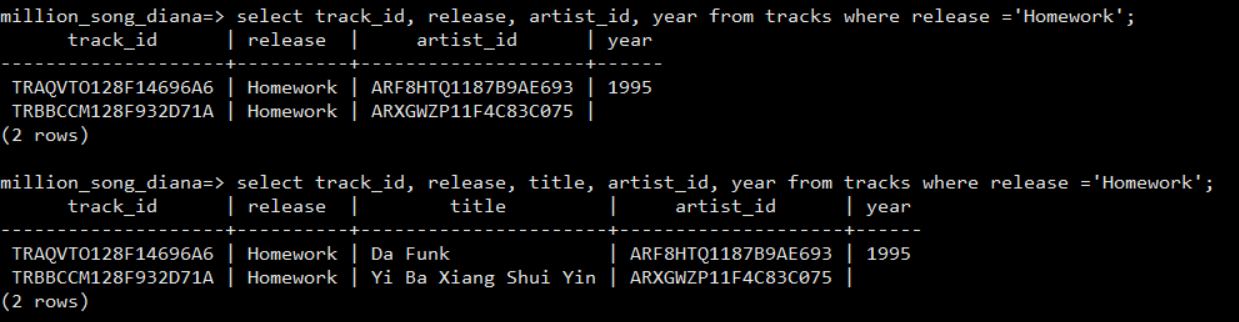
1. First, I counted the total tracks from 1999:



Then, I looked to see if there are more than one tracks in the album being released in the same year.

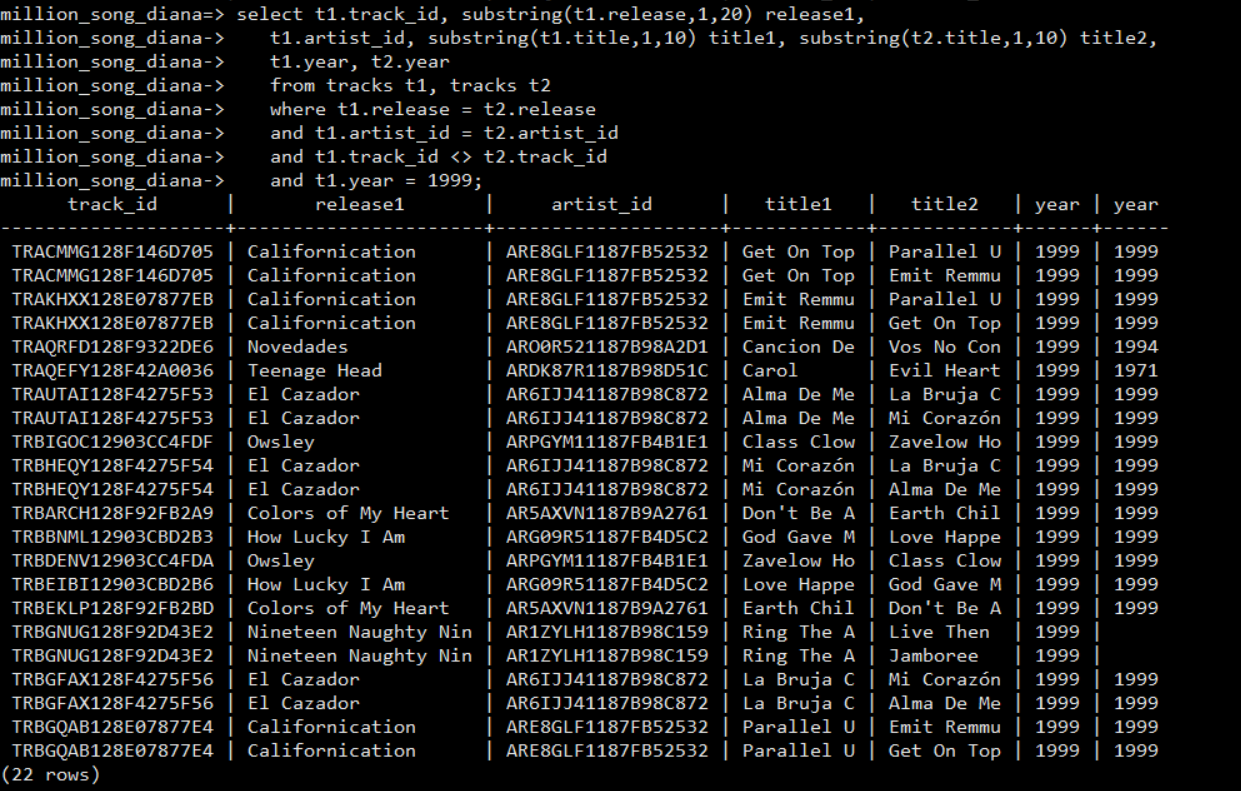


From there, I analyzed to see if one of the tracks had ‘null’ year value, as shown below:

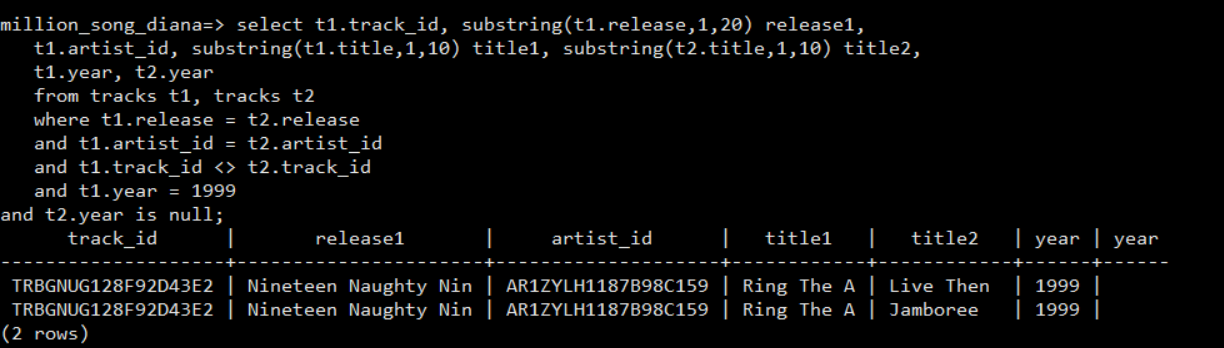


However, I realized that we needed to look for the same artist and release in the year of 1999.

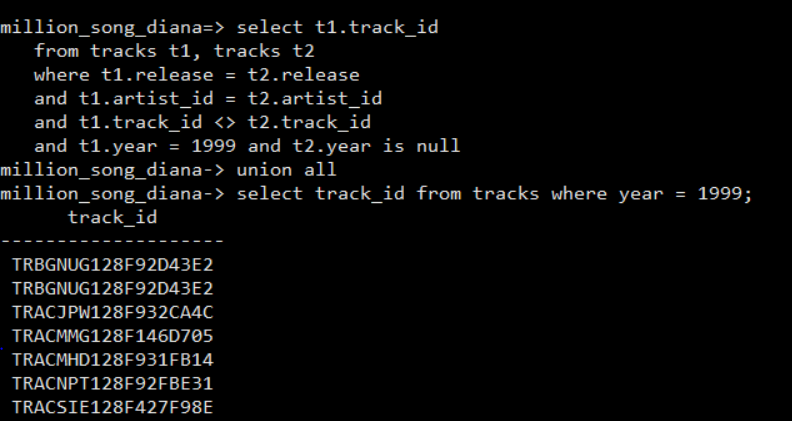
I found that there was a case where the year is null on one of tracks belong to the same artist in the same release and year of 1999:



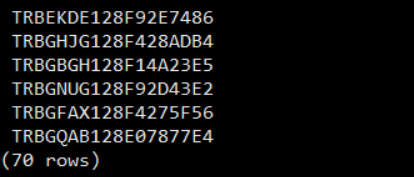
Here is the query to look for such records:



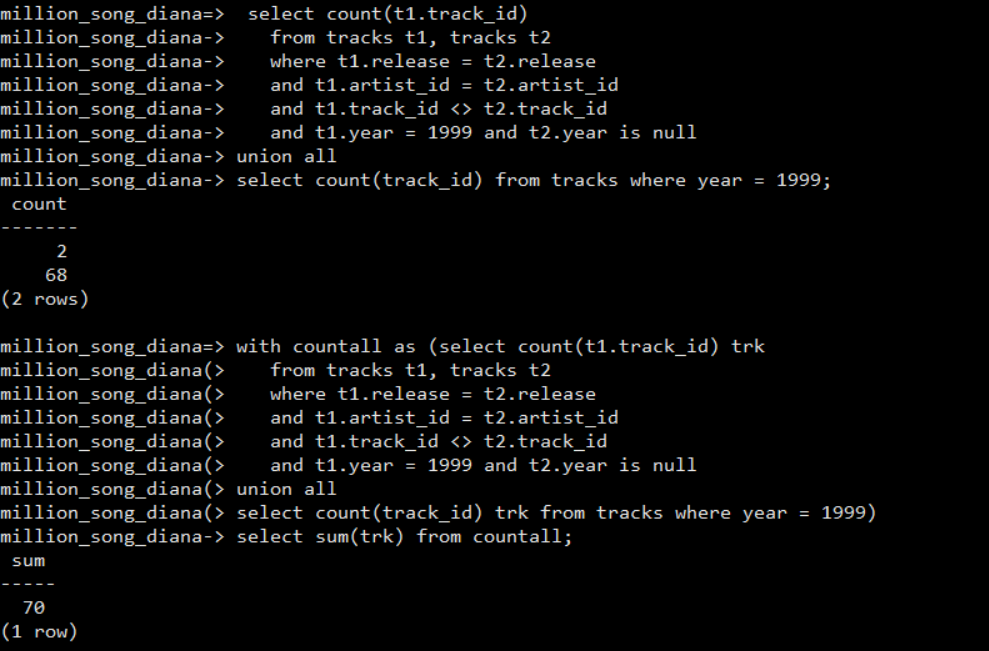
Therefore, in order for us to count the number of tracks from 1999, I used this query:



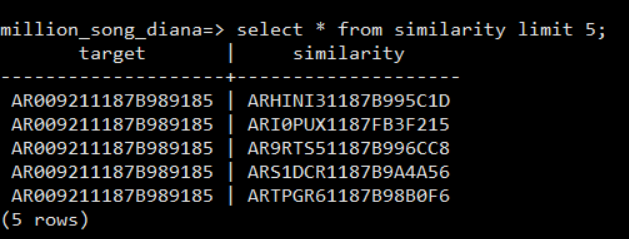
….

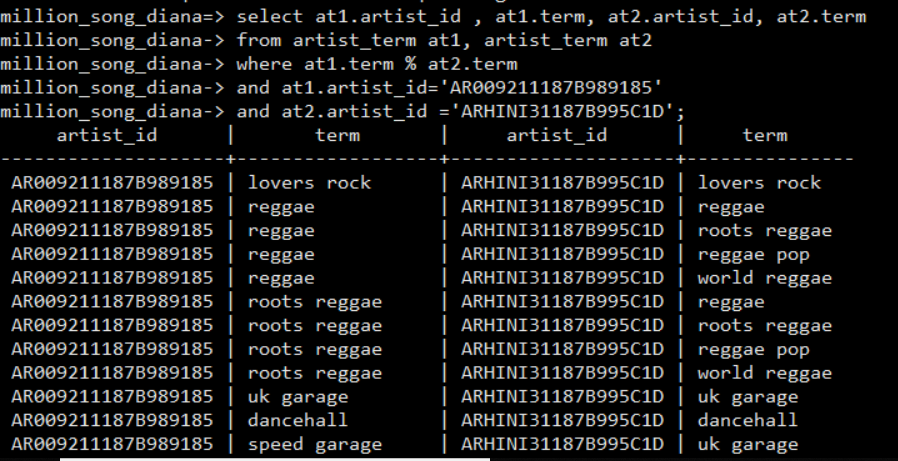


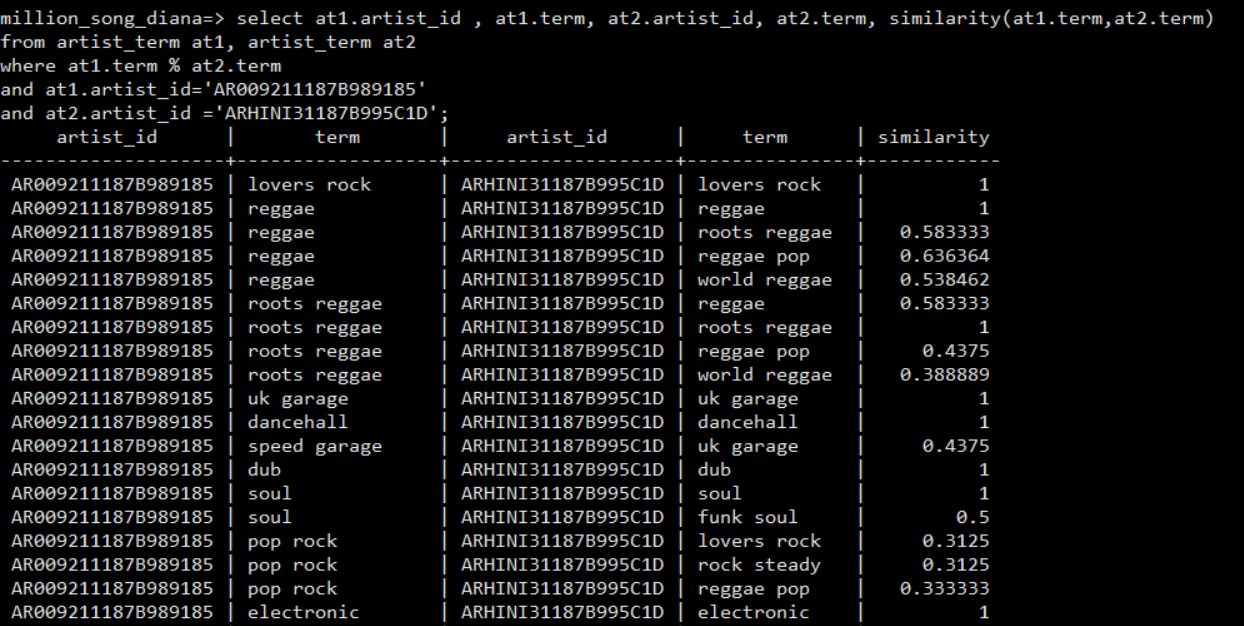
Then, to sum them up:



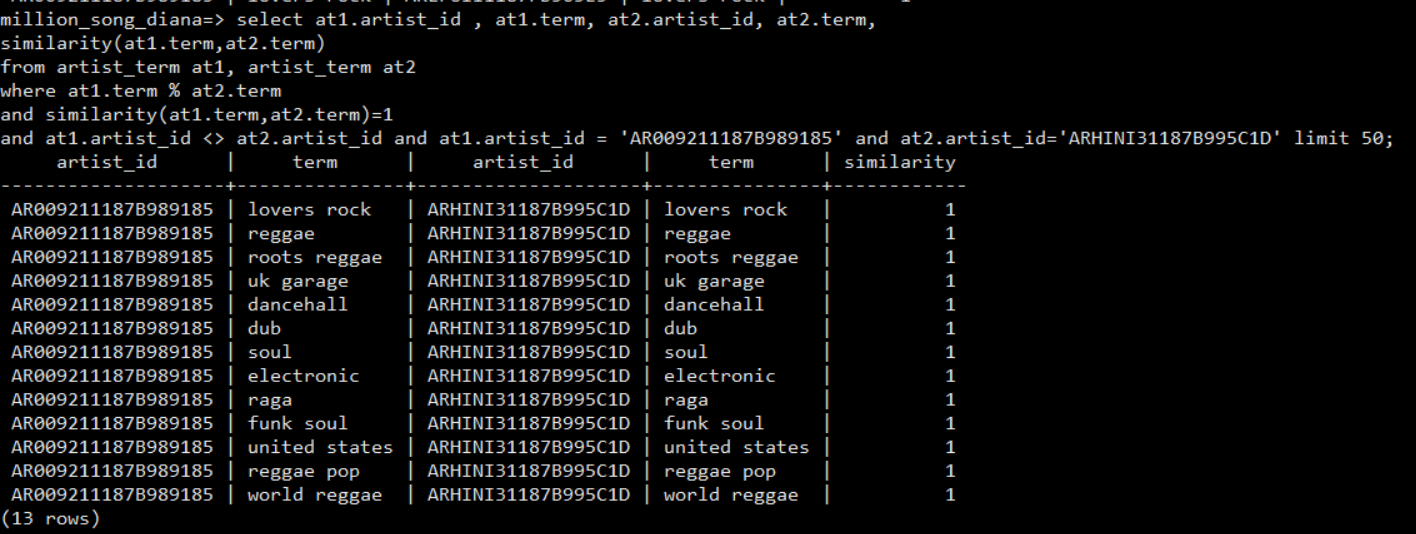
The number of tracks from 1999 = 70 tracks

5.A. First, I looked at the sample of the data in similarity table:

Then, I analyzed the artist\_term table to see if the data from similarity table came from artist\_id that have similarity in terms: 

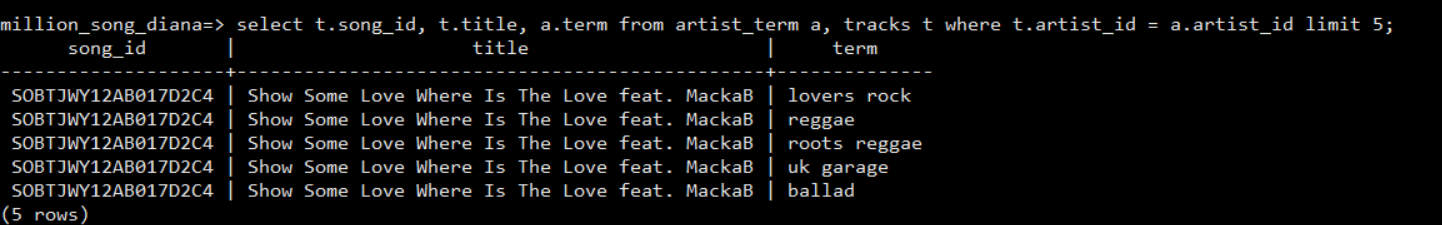


So, I think that the data in similarity table came from the artist\_term table where the artists have similarity in terms (similarity = 1)



B. By replacing ‘term’ with a cube, we can measure/ score the song using the cube\_ur\_coord. Also, it would be easier to find favorite songs and categorizing songs by the term (0 -> not in this term, 10-> completely belong in this term).

Imagine that we would like to know what the closest term for this song is. From the current design, we would not know whether this song is more closer to be a reggae or uk garage. A cube can give us more detail info on this song.



With a cube, we can use cube\_ur\_coordinate and the result will be like this:

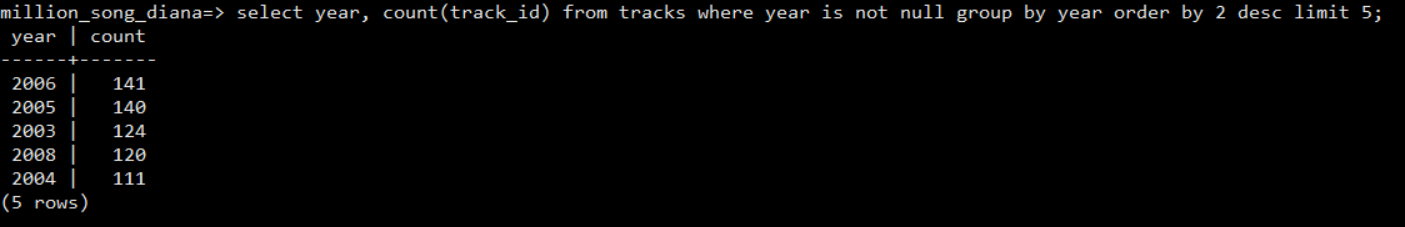
If the song is more closely categorized as ‘uk garage’, it may have score of 8 for ‘uk garage’ and only 5 for ‘reggae’. Assuming that the term was stored in this order: lovers rock, reggae, roots reggae, uk garage, ballad in the term table, when we query on this, it would result something like :

SOBTJWY12AB017D2C4 Show Some Love …. {1, 5, 4, 10, 3}

This song has a score of 1 for lovers rock, 5 for reggae, 4 for roots reggae, 10 for uk garage, 3 for ballad.

Therefore, it will be helpful to have ‘term’ with a cube.

6.A. First, I grouped the number of tracks by year and picked the top five.



Then, I created a table lookup for latitude and longitude:

CREATE TABLE location\_lookup (

state character(2),

minlat real,

maxlat real,

minlng real,

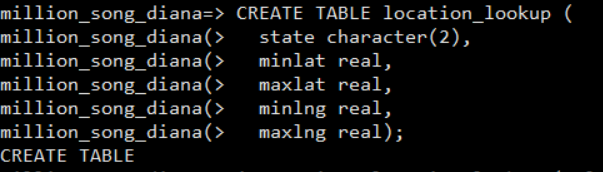
maxlng real);

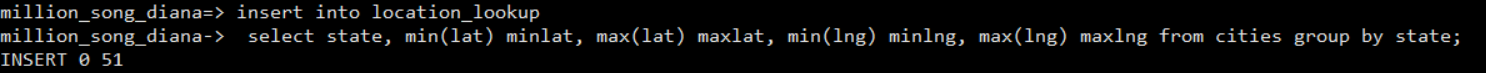
INSERT INTO location\_lookup

SELECT state, MIN(lat) minlat, MAX(lat) maxlat, MIN(lng) minlng, MAX(lng) maxlng

FROM cities

GROUP BY state;





In order for us to get the top five year and state by number of tracks, we need to join the tables: tracks, location\_lookup, and artists:

SELECT t.year||','||s.state Year\_State, COUNT(t.track\_id) Tracks

FROM tracks t, artists a, location\_lookup s

WHERE t.artist\_id = a.artist\_id

AND (a.lat > s.minlat and a.lat < s.maxlat)

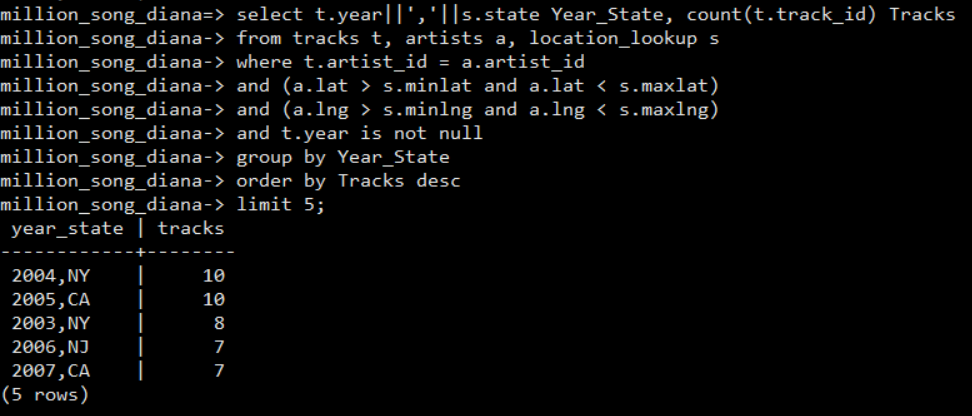
AND (a.lng > s.minlng and a.lng < s.maxlng)

AND t.year is not null

GROUP BY Year\_State

ORDER BY Tracks DESC

LIMIT 5;



B. Relational algebra : Since there is a join, the formula will be : R ⨝Ai𝜃Bj S ⨝Ai𝜃Bj T

R = tracks, S = artists, T = location lookup

Left Ai= tracks.artist\_id

Left Bj= artists.artist\_id

Right Ai = artist.lat, location\_lookup.lat

Right Bj = artist.lng, location\_lookup.lng

C. To find the state/province of each artist:

SELECT DISTINCT a.artist\_name, a.lat, a.lng, s.state

FROM artists a, location\_lookup s

WHERE (a.lat > s.minlat and a.lat < s.maxlat)

AND (a.lng > s.minlng and a.lng < s.maxlng);



For those artists’records that have missing lat/lng, we can add those to query using UNION and display ‘Unknown Location’ for the state.

SELECT DISTINCT a.artist\_name, s.state

FROM artists a, location\_lookup s

WHERE (a.lat > s.minlat and a.lat < s.maxlat)

AND (a.lng > s.minlng and a.lng < s.maxlng)

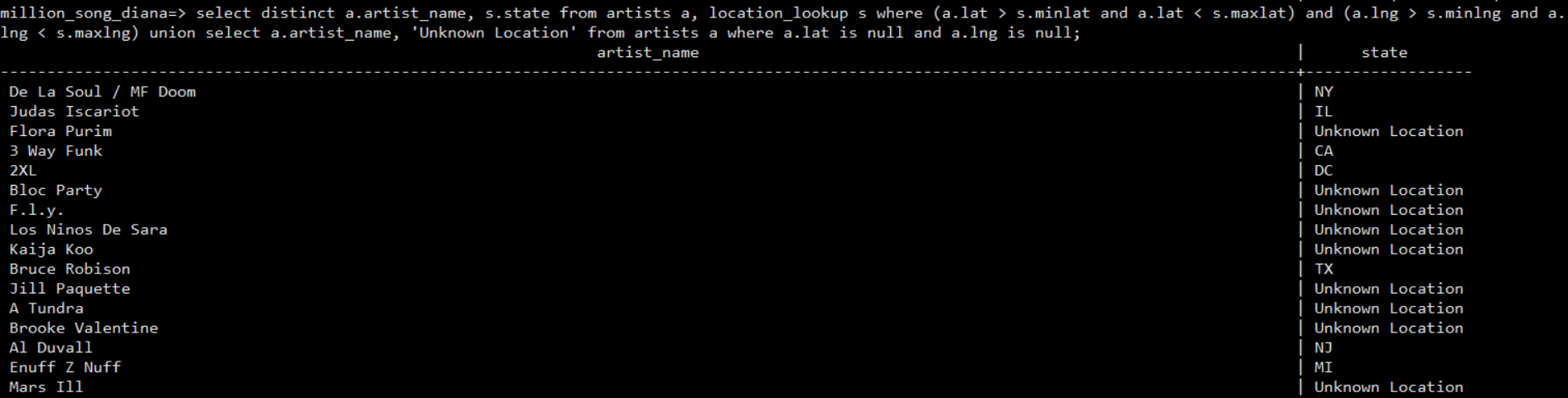
UNION

SELECT a.artist\_name, 'Unknown Location'

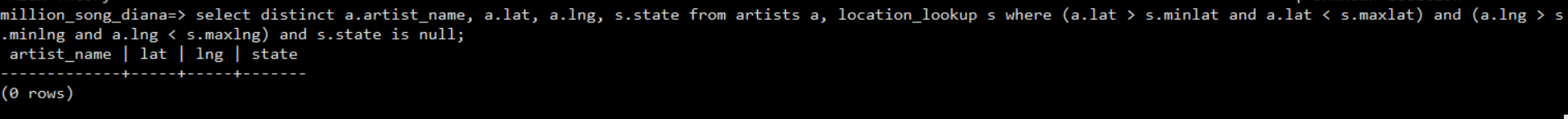
FROM artists a

WHERE a.lat is null

AND a.lng is null;



As a result, we are able to find the state for the ones with lat and lng information. There is no lat/lng where we can’t get the state from.

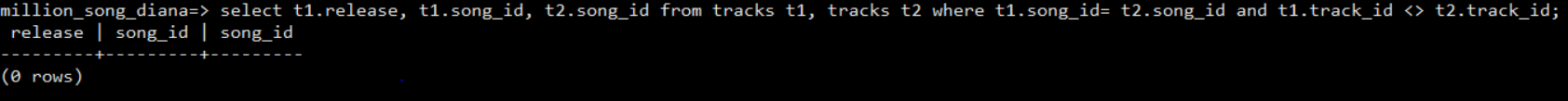


D. With a cube, we can measure the distance more accurately using lat, lng using cube\_distance function and possibly incorporate this for GPS location.

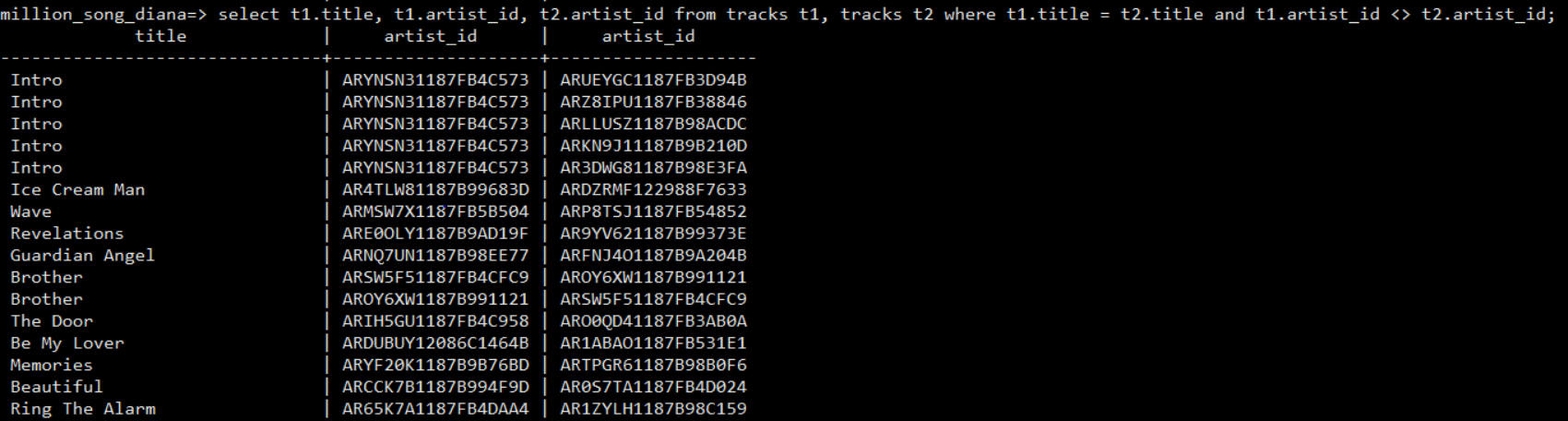
E. The location\_lookup table can be used as our lat/lng boundary for each state. Then, in both artists and cities tables, we can store the cube\_distance(lat, lng) to see how far the city is from the box/boundary (for each state).

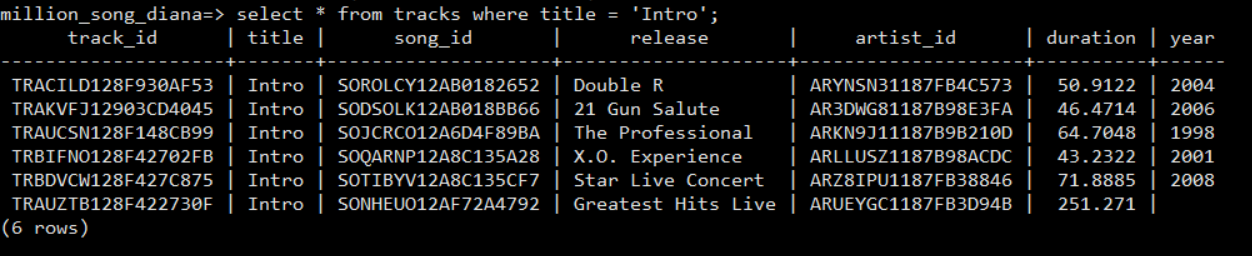
7. A. Song\_id does not represent a particular tune with particular lyrics. It is uniquely identified a title. 

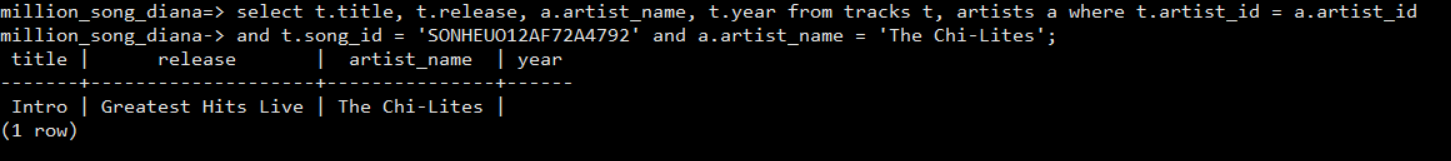
To prove:

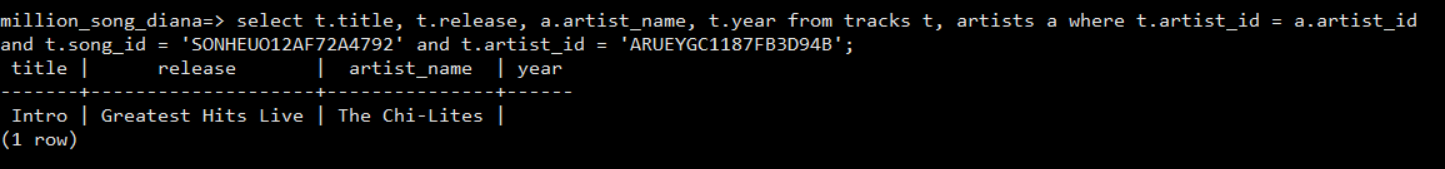


There is no duplicate song\_id in the tracks table. While it is unique, it is not enough to identify a track on a release. We still need track\_id and song\_id to identify a title in the tracks table. In YouTube, song\_id would not be recognized, instead we used title or release to look for a song.

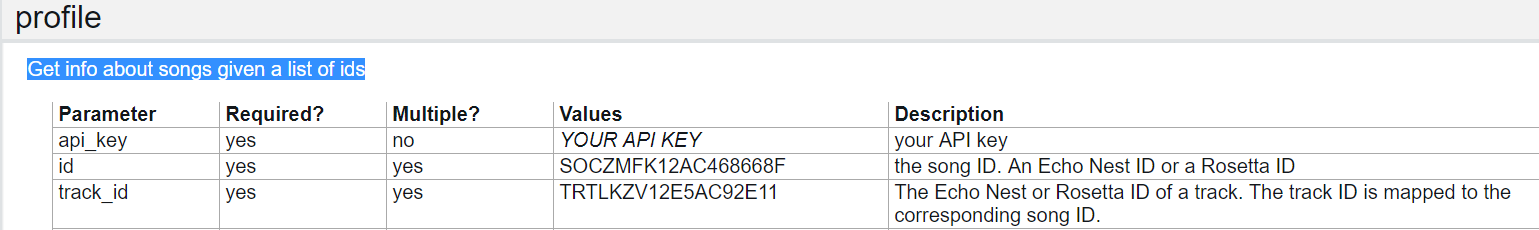
B. Song\_id does not represent a particular title performed by a particular artist. By itself, song\_id is uniquely identified a song(title). If we want to look for a particular title performed by a particular artist, we have to use both artist\_id and song\_id. 

For example, since “Intro” was performed by different artists, if we want to listen to the “Intro” performed by The Chi-Lites , we have to use both song\_id and artist\_id:





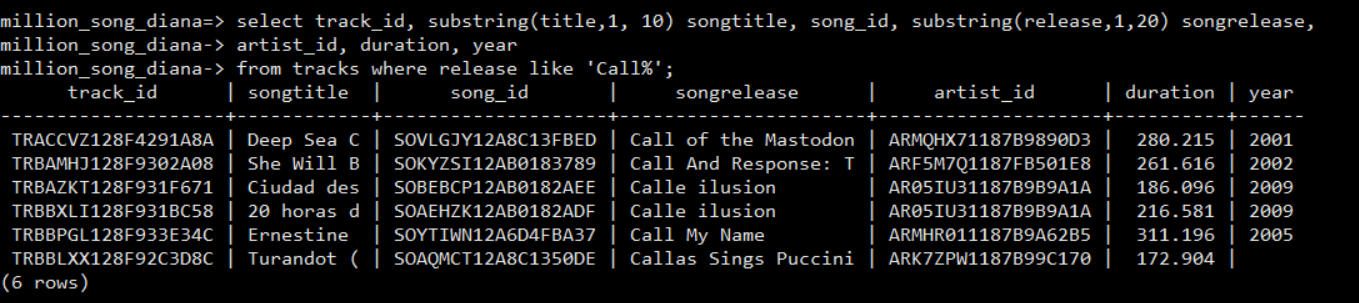
C. Song\_id is a primary key for the song entity.



D. The tracks table can be further normalized. Another table can be created to store song\_id and title. Therefore, title should be an attribute of song table.

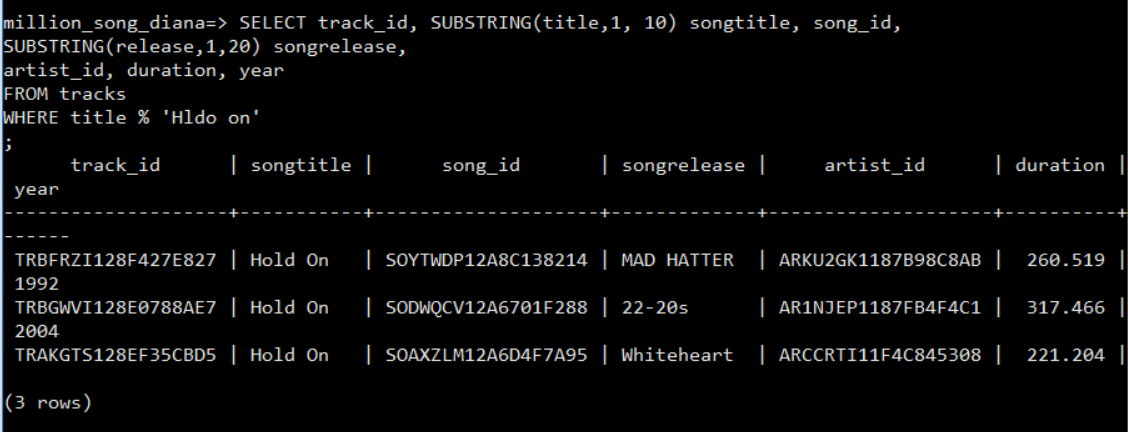
8. a. ‘like’ : Find a release that starts with ‘Call’ from tracks table.

LIKE can be useful when we are looking for a release that starts/ends with a keyword. For example, if we want to find a release from tracks table that starts with ‘Call’, we can do the following query:



b. % Help a user to find a song title ‘Hold On’ when the user misspell the word :’Hldo on’

This % can be useful if we misspell the keyword. For example, if we want to find ‘Hold On’ song title but we misspell:’Hldo on’ from tracks table:



SELECT track\_id, SUBSTRING(title,1, 10) songtitle, song\_id,

SUBSTRING(release,1,20) songrelease,

artist\_id, duration, year

FROM tracks

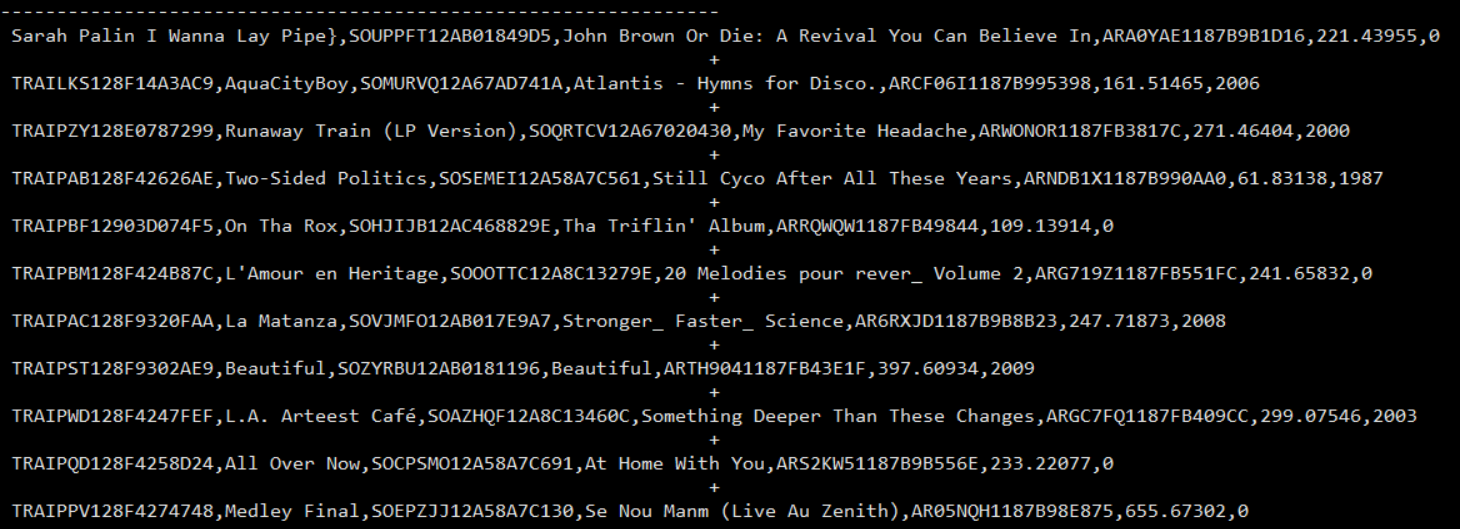
WHERE title % 'Hldo on'

c. @@ Find a song that have ‘christmas & lights’ on the title.

If we don’t remember the exact song title but we know that it contains words like ‘christmas &light’ then we can use @@. For example:

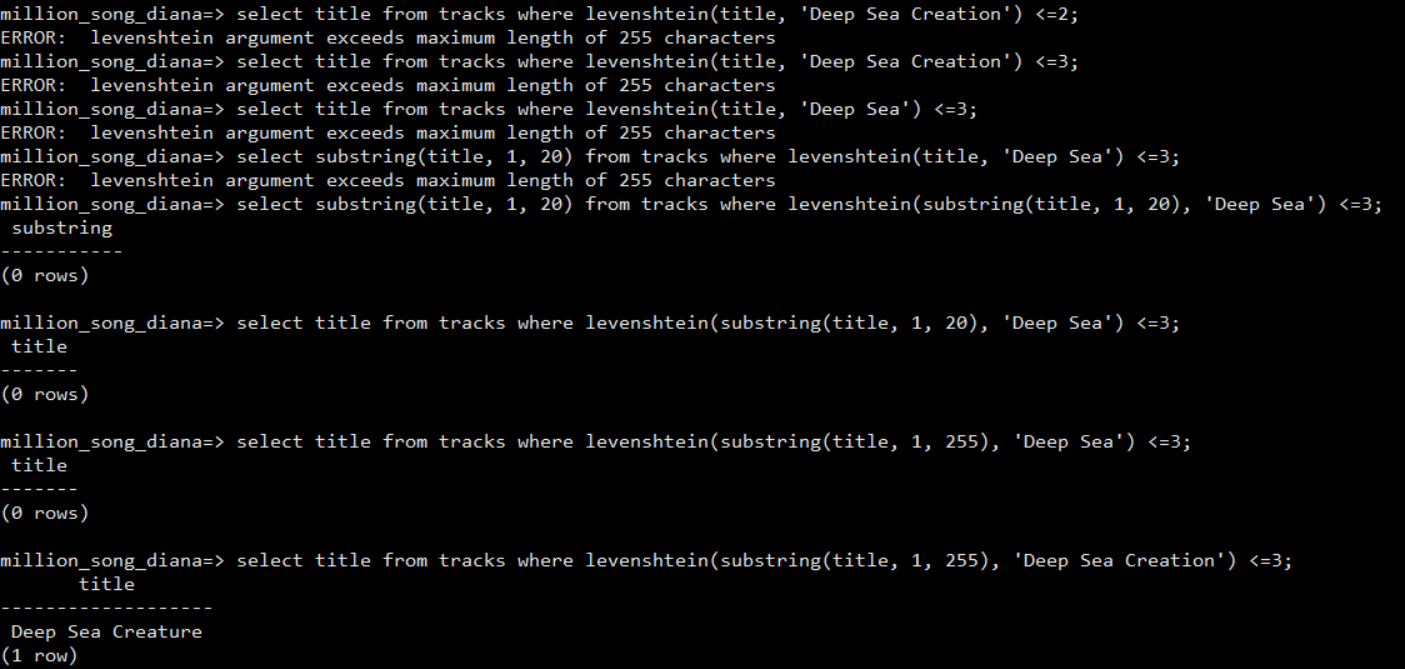
Select title from tracks where title @@ ‘christmas&lights’;

In this case, my query returned bad data since there were no matches found.



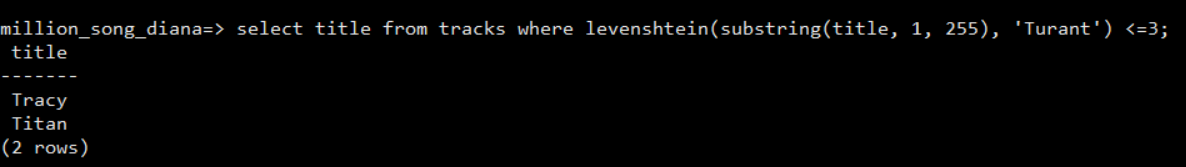
d. levenstein : Find the closest list of songs based on user’s keyword. In this case, the user entered ‘Deep Sea Creation’

This can be useful when displaying more accurate results from user’s query. We measure the distance of the keyword that was being entered by the user to our song records. Let’s say that a user entered a keyword ‘Deep Sea Creation’ when looking for a song:



Our query using ‘levenshtein’ will return ‘Deep Sea Creature’ song.

Another example:



e. metaphone() : Find an artist name that sounds like ‘Annie Philips’

This can be used if we don’t know exactly how to spell an artist\_name(for example: we are looking for ‘Annie Philippe’ but we think it spells ‘Annie Philips’ based on the sound. It will find the records sounding like ‘Annie Philips’.

