Queries Used In MySQL Workbench, Cypher Desktop, Mongo+R, Mongo Playground

SQL QUERIES

```
#1a QUERY FOR WHICH ARE MOST POPULAR ARTIST BY LOCATION
select s.artist_name, u.user_location, (count(distinct l.userid))/ux.total as
percent_users_that_listens_to_artist,
count(distinct s.track_id) as number_of_songs_by_artist_users_listened_to,
sum(l.number_of_listens) as total_times_listened
from user_profile u, listens l, songs s,
(select user_location, count(*) as total from user_profile group by user_location) as ux
where u.userid=l.userid and l.track_id=s.track_id and u.user_location=ux.user_location
and u.user_location LIKE '%MA%'
group by s.artist_name, u.user_location
ORDER BY percent_users_that_listens_to_artist DESC,
number_of_songs_by_artist_users_listened_to DESC,
total_times_listened DESC;
```

```
#1b QUERY FOR WHICH ARE MOST POPULAR GENRE POPULARITY BY LOCATION
select a.genre, u.user_location, (count(distinct l.userid))/ux.total as
percent_users_that_listens_to_genre,
count(distinct s.artist_name) as number_of_artists_in_genre_users_listen_to,
sum(l.number_of_listens) as total_times_listened
from user_profile u, listens l, songs s, artist_profile a,
(select user_location, count(*) as total from user_profile group by user_location) as ux
where u.userid=l.userid and l.track_id=s.track_id and s.artist_name=a.artist_name and
u.user_location=ux.user_location
and u.user_location LIKE '%MA%'
group by genre, u.user_location
ORDER BY percent_users_that_listens_to_genre DESC,
number_of_artists_in_genre_users_listen_to DESC,
total_times_listened DESC;
```

```
#1c QUERY FOR WHICH ARE MOST POPULAR ARTIST IN GENRE
```

```
select a.genre, a.artist_name, (((count(distinct l.userid))/8)*100) as
percent_users_that_listens_to_artist,
sum(l.number_of_listens) as total_times_listened
from user_profile u, listens l, songs s, artist_profile a
where u.userid=l.userid and l.track_id=s.track_id and s.artist_name=a.artist_name
and genre LIKE '%art-pop%'
group by genre, artist_name
order by percent_users_that_listens_to_artist DESC, total_times_listened DESC;


#2
mysql > delimiter //
CREATE TRIGGER date_check BEFORE INSERT ON concerts
FOR EACH ROW
BEGIN
IF DATEDIFF(NEW.concert_date,curdate()) > 0 THEN
SET NEW.show_status = "upcoming";
END IF;
END;//
mysql> delimiter ;


CREATE DEFINER=`root`@`localhost` TRIGGER `add_concerts_id` BEFORE INSERT ON `concerts` FOR
EACH ROW SET NEW.concertid = CONCAT(NEW.artist_name,'-',NEW.concert_location, '-',
NEW.concert_date)


CREATE DEFINER=`root`@`localhost` PROCEDURE `delete_old_dates`()
BEGIN
DELETE FROM concerts WHERE DATEDIFF(concert_date,curdate()) < 0;
END


SELECT CURDATE();
SELECT DATEDIFF('2008-05-17',curdate());

INSERT INTO concerts(concertid,artist_name,concert_location,concert_date) VALUES('sp4', 'Glass
Animals', 'MD', '2019-12-03');

CALL set_to_expired
  ("expired");

CALL delete_old_dates;

select * from spotify.concerts;


#3 #pick
```

```
CREATE VIEW advertiseto AS
select c.concertid, c.artist_name,  c.concert_location, l.userid, sum(l.number_of_listens) as
total_times_listened, c.status as stat
from concerts c, artist_profile a, songs s, listens l, user_profile u
where c.artist_name=a.artist_name and
a.artist_name=s.artist_name and s.track_id=l.track_id and l.userid=u.userid and
c.concert_location=u.user_location
group by concertid, userid
having total_times_listened>50 and stat="upcoming";
```

CYPHER QUERIES

USER LISTENS TO SONG

```
LOAD CSV WITH HEADERS FROM 'file:///user_profile.csv' AS row
WITH row.username as username, row.userid AS userid, row.user_location AS user_location
MERGE (u:user {username: username})
  SET u.userid=userid, u.user_location = user_location
RETURN u
```

```
LOAD CSV WITH HEADERS FROM 'file:///songs.csv' AS row
WITH row.track_id as track_id, row.artist_name AS artist_name, row.track_name AS track_name
MERGE (s:songs {track_id: track_id})
  SET s.artist_name=artist_name, s.track_name = track_name
RETURN s
```

```
USING PERIODIC COMMIT 10000
LOAD CSV WITH HEADERS FROM 'file:///listens.csv' AS row
WITH row.userid AS userid, row.track_id AS track_id, toInteger(row.number_of_listens) as total_listens
MATCH (u:user {userid: userid})
MATCH (s:songs {track_id: track_id})
MERGE (u)-[rel:ListensTo {total_listens:total_listens}]->(s)
RETURN u,rel,s Limit 100
```

```
MATCH (u:user)-[rel:ListensTo]->(s:songs)
RETURN u, rel, s LIMIT 50
```

GENRES BELONG TO UMBRELLA

```
LOAD CSV WITH HEADERS FROM 'file:///genres.csv' AS row
WITH row.genre_id as genre_id, row.genre_name AS genre_name
MERGE (g:genre {genre_id: genre_id})
  SET g.genre_name=genre_name
RETURN g
```

```
LOAD CSV WITH HEADERS FROM 'file:///umbrellas.csv' AS row
WITH row.umbrella_id as umbrella_id, row.umbrella_name AS umbrella_name
MERGE (u:umbrella {umbrella_id: umbrella_id})
  SET u.umbrella_name=umbrella_name
RETURN u
```

```
USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS FROM 'file:///genre-details.csv' AS row
WITH row.umbrella_id AS umbrella_id, row.genre_id AS genre_id, toInteger(row.total) as total_listens
MATCH (u:umbrella {umbrella_id: umbrella_id})
MATCH (g:genre {genre_id: genre_id})
MERGE (g)-[rel:BelongsTo {total_listens:total_listens}]->(u)
RETURN u,rel,g
```

ARTIST PLAYS GENRE

```
USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS FROM 'file:///artist_profile.csv' AS row
WITH row.artist_name AS artist_name, row.genre_id AS genre_id
MATCH (g:genre {genre_id: genre_id})
MATCH (s:songs {artist_name: artist_name})
MERGE (s)-[rel:Plays]->(g)
RETURN s,rel,g Limit 100
```

FINAL

```
MATCH (u:user)-[rel:ListensTo]->(s:songs)-[pel:Plays]->(g:genre)-[gel:BelongsTo]->(um:umbrella)
RETURN u, rel, s,pel,g, gel, um LIMIT 50
```

QUERIES

```
#Number of songs listed
MATCH (n:songs) RETURN count(*) AS number_songs

#Number of users
MATCH (n:user) RETURN count(*) AS number_users

#Number of artists
MATCH (n:songs) RETURN count(distinct(n.artist_name)) AS number_artists

#Number of genres
MATCH (n:genre) RETURN count(n.genre_id) AS number_genres

Which users listen to a genre
MATCH (u:user)-[rel:ListensTo]->(s:songs)-[pel:Plays]->(g:genre)
RETURN g.genre_id, collect(distinct(u.userid))

#List of genre everyone listens to
MATCH (u:user)-[rel:ListensTo]->(s:songs)-[pel:Plays]->(g:genre)
WITH g.genre_id as genre_a, count(distinct(u.userid)) as number_of_users
WHERE number_of_users=8
RETURN genre_a, number_of_users
ORDER BY number_of_users DESC
```

```
#list of genres noone likes
MATCH (g:genre)
WHERE NOT ((:user)-[:ListensTo]->(:songs)-[:Plays]->(g))
RETURN distinct(g.genre_id);


#artist everyone likes
MATCH (u:user)-[rel:ListensTo]->(s:songs)
WITH s.artist_name as artist, count(distinct(u.userid)) as number_of_users
WHERE number_of_users=8
RETURN artist, number_of_users
ORDER BY number_of_users DESC

#specific user's listens
MATCH (u:user{userid:'bd93'})-[r:ListensTo]->(s:songs) RETURN u.userid, s.artist_name as
artists_user_listens_to, r.total_listens as total_times_listened


#specific genre's listens
MATCH (u:user)-[rel:ListensTo]->(s:songs)-[pel:Plays]->(g:genre { genre_id: 'chamber-pop' })
WITH u.user_location as user_location,g.genre_id as genre_id, u.userid as userid, sum(rel.total_listens)
as total_listens
WHERE total_listens>600
RETURN user_location,genre_id, userid, total_listens


#ADD
CREATE (n:user { userid: 'xj21', username: 'xj21', user_location:'TX' })


CREATE (u:user{userid:'xj21'})-[r:ListensTo{total_listens:'1'}]->(s:songs{artist_name:'Lorde'})


MATCH (u:user{userid:'xj21'})-[r:ListensTo]->(s:songs) RETURN u.userid, s.artist_name as
artists_user_listens_to, r.total_listens as total_times_listened


#DELETE
MATCH (n:genre { genre_id: 'opera' })
DETACH DELETE n


MATCH (u:user)-[rel:ListensTo]->(s:songs)-[pel:Plays]->(g:genre)-[gel:BelongsTo]->(um:umbrella)
RETURN u, rel, s,pel,g, gel, um limit 200
```

MONGO QUERIES IN R

```
library(mongolite)

library(RMySQL)

library(ggplot2)

library(dplyr)

library(tidyr)

mydb <- dbConnect(MySQL(), user = 'testuser', password = ‘pw',

                dbname = 'spotify', host = '127.0.0.1')


rs <- dbSendQuery(mydb,"select * from songs;")

songs=dbFetch(rs)


#CONNECT TO MONGO AND CREATE SONGS COLLECTION

my_collection = mongo(collection = "songs", db = "Spotify")

my_collection$drop()

my_collection$insert(songs)


#ABOUT COLLECTION

my_collection$find()

length(my_collection$distinct("artist_name"))

my_collection$iterate()$one()

#INSERT NEW SONGS TO COLLECTION

my_collection$insert('{"track_id":"x","artist_name":"Bon
Iver","track_name":"test","acousticness":"1","danceability":"1","energy":"1", "liveness":"1"
,"loudness":"1","speechiness":"1","tempo":"1","valence":"1","tally":"1"}')


#check if it is there

my_collection$find('{"artist_name":"Bon
Iver","track_name":"test","acousticness":"1","danceability":"1","energy":"1", "liveness":"1"
,"loudness":"1","speechiness":"1","tempo":"1","valence":"1","tally":"1"}', fields = '{"_id":0,
"track_id":1}')
```

```
#MONGO QUERY - table of artists' average metrics using Aggregate

dfm1=my_collection$aggregate('[{"$group":

                {"_id":"$artist_name", "count": {"$sum":1}, "avg_acousticness":{"$avg":"$acousticness"},
"avg_danceability":{"$avg":"$danceability"}, "avg_energy":{"$avg":"$energy"},
"avg_liveness":{"$avg":"$liveness"}, "avg_loudness":{"$avg":"$loudness"},
"avg_speechiness":{"$avg":"$speechiness"}, "avg_tempo":{"$avg":"$tempo"},
"avg_valence":{"$avg":"$valence"}}}]')

head(dfm1)


#MONGO QUERY - Find supporting artists who have specific metrics using Find

dfm1[,3:10]=round(dfm1[,3:10], 0)

my_collection2 = mongo(collection = "average_metrics", db = "Spotify")

my_collection2$drop()

my_collection2$insert(dfm1)

my_collection2$find('{"avg_acousticness":1,"avg_danceability":0, "avg_energy":0, "avg_liveness":0,
"avg_loudness":-15, "avg_speechiness":0, "avg_valence":0}', fields = '{"_id":1, "artist_name":1}')
```

MONGO QUERIES IN PLAYGROUND

```
db.dropDatabase();


db.spotify_user.insert(

  {

    "userid":"bd93",

    "user_location":"MA"

  }

);


db.spotify_user.insert(

  {

    "userid":"bd89",

    "user_location":"NH"

  }

);


db.spotify_user.insert(

  {

    "userid":"fr89",

    "user_location":"NY"

  }

);


db.spotify_user.insert(

  {

    "userid":"iv93",

    "user_location":"VT"

  }
);

db.spotify_user.insert(
   {
      "userid":"ja93",
      "user_location":"NY"
   }
);

db.spotify_user.insert(
   {
      "userid":"kl93",
      "user_location":"MD"
   }
);

db.spotify_user.insert(
   {
      "userid":"rm93",
      "user_location":"FL"
   }
);

db.spotify_user.insert(
   {
      "userid":"st92",
      "user_location":"MA"
   }
);
```

```
db.spotify_songs.insert(
   {
       "track_id":"4uexcsJVOIsqiEZgshqKUy",
       "artist_name":"070 Shake",
       "track_name":"Accusations",
       "acousticness":0.453,
       "danceability":0.901,
       "energy":0.283,
       "liveness":0.0858,
       "loudness":-10.156,
       "speechless":0.0769,
       "tempo":118.963,
       "valence":0.333
   }
);

db.spotify_songs.insert(
   {
       "track_id":"2DnOFuJwSSZc1bxUqsLCMr",
       "artist_name":"22-20s",
       "track_name":"Devil In Me",
       "acousticness":0.00167,
       "danceability":0.439,
       "energy":0.932,
       "liveness":0.264,
       "loudness":-6.127,
       "speechless":0.106,
       "tempo":109.72,
       "valence":0.342
   }
);

db.spotify_songs.insert(
   {
       "track_id":"2WWRiGaZ6MwvAMPJHSQfps",
       "artist_name":"Aaron Taylor",
       "track_name":"Lay My Troubles Down",
       "acousticness":0.176,
       "danceability":0.799,
       "energy":0.429,
       "liveness":0.102,
       "loudness":-6.449,
       "speechless":0.0592,
       "tempo":92.002,
       "valence":0.621
   }
);

db.spotify_songs.insert(
   {
       "track_id":"0iSJuL1AKxnHiDqmQaHlwQ",
       "artist_name":"Ahmet Kilic",
       "track_name":"Good Ones Go - Original Mix",
       "acousticness":0.0317,
       "danceability":0.841,
       "energy":0.623,
       "liveness":0.185,
       "loudness":-9.994,
       "speechless":0.0642,
       "tempo":120.013,
       "valence":0.408
   }
);

db.spotify_songs.insert(
   {
       "track_id":"586wnNE1CaMcp5UrgkzXzV",
       "artist_name":"Akora",
       "track_name":"Eyes of Love - Toly Braun Remix",
       "acousticness":0.0725,
       "danceability":0.847,
       "energy":0.652,
       "liveness":0.0919,
       "loudness":-6.982,
       "speechless":0.0801,
       "tempo":118.016,
       "valence":0.822
   }
);

db.spotify_songs.insert(
   {
       "track_id":"2lE7oRoKssULAtbWViL385",
       "artist_name":"Alanis Morissette",
       "track_name":"Hand in My Pocket - 2015 Remaster",
       "acousticness":0.135,
       "danceability":0.657,
       "energy":0.655,
       "liveness":0.102,
       "loudness":-8.3,
       "speechless":0.0248,
       "tempo":92.259,
       "valence":0.668
   }
);

db.spotify_songs.insert(
   {
       "track_id":"1d6KS9GH06JAd19uiBy9IE",
       "artist_name":"Alanis Morissette",
       "track_name":"Ironic - 2015 Remaster",
       "acousticness":0.218,
       "danceability":0.408,
       "energy":0.582,
       "liveness":0.159,
       "loudness":-8.305,
       "speechless":0.0508,
       "tempo":114.926,
       "valence":0.365,
   }
);

db.spotify_songs.insert(
   {
       "track_id":"3jS7bB0oXVOwGFZn3aE5NV",
       "artist_name":"Alanis Morissette",
       "track_name":"You Oughta Know - 2015 Remaster",
       "acousticness":0.21,
       "danceability":0.665,
       "energy":0.834,
       "liveness":0.452,
       "loudness":-7.737,
       "speechless":0.0576,
       "tempo":105.292,
       "valence":0.411,
   }
);

db.spotify_songs.insert(
   {
       "track_id":"5erTWXdADowSkh825UUOho",
       "artist_name":"Alina Baraz",
       "track_name":"Can I",
       "acousticness":0.312,
       "danceability":0.363,
       "energy":0.536,
       "liveness":0.128,
       "loudness":-8.124,
       "speechless":0.0632,
       "tempo":200.173,
       "valence":0.106,
   }
```

```
);

db.spotify_songs.insert(
  {
     "track_id":"20TYNq9o5sdBAbkCWE9ih7",
     "artist_name":"Alina Baraz",
     "track_name":"Electric (feat. Khalid)",
     "acousticness":0.738,
     "danceability":0.599,
     "energy":0.396,
     "liveness":0.102,
     "loudness":-10.489,
     "speechless":0.0392,
     "tempo":111.072,
     "valence":0.134,
  }
);

db.spotify_songs.insert(
  {
     "track_id":"17YuXw2ScwLLL1sUrRKhoW",
     "artist_name":"Alina Baraz",
     "track_name":"Fantasy",
     "acousticness":0.353,
     "danceability":0.68,
     "energy":0.747,
     "liveness":0.138,
     "loudness":-6.056,
     "speechless":0.091,
     "tempo":113.933,
     "valence":0.331,
  }
);

db.spotify_songs.insert(
  {
     "track_id":"3n69hLUdIsSa1WlRmjMZlW",
     "artist_name":"alt-J",
     "track_name":"Breezeblocks",
     "acousticness":0.096,
     "danceability":0.616,
     "energy":0.656,
     "liveness":0.205,
     "loudness":-7.298,
     "speechless":0.0344,
     "tempo":150.071,
     "valence":0.286,
  }
);

db.spotify_songs.insert(
  {
     "track_id":"2mkv1b3dRFyiJ4Ybq31owf",
     "artist_name":"alt-J",
     "track_name":"Hunger Of The Pine",
     "acousticness":0.785,
     "danceability":0.6,
     "energy":0.413,
     "liveness":0.109,
     "loudness":-9.572,
     "speechless":0.0258,
     "tempo":93.618,
     "valence":0.0741,
  }
);

db.spotify_songs.insert(
  {
     "track_id":"7plLvN3xOrNFCnZX1SrUpj",
     "artist_name":"alt-J",
     "track_name":"n Cold Blood (feat. Pusha T) - Twin Shadow Version",
     "acousticness":0.133,
     "danceability":0.794,
     "energy":0.478,
     "liveness":0.358,
     "loudness":-8.44,
     "speechless":0.0987,
     "tempo":143.003,
     "valence":0.276,
  }
);

db.spotify_songs.insert(
  {
     "track_id":"3aA5fk4c6a7e5HM4rJqkSF",
     "artist_name":"alt-J",
     "track_name":"Matilda",
     "acousticness":0.779,
     "danceability":0.576,
      "energy":0.653,
     "liveness":0.113,
     "loudness":-9.132,
     "speechless":0.03,
     "tempo":147.867,
     "valence":0.209,
  }
);

db.spotify_songs.insert(
  {
     "track_id":"1o22EcqsCANhwYdaNOSdwS",
     "artist_name":"alt-J",
     "track_name":"Tessellate",
     "acousticness":0.364,
     "danceability":0.702,
     "energy":0.607,
     "liveness":0.123,
     "loudness":-6.509,
     "speechless":0.0405,
     "tempo":116.961,
     "valence":0.463,
  }
);
```

```
db.getCollectionNames();

db.spotify_user.find();

db.spotify_songs.find();



var mapFunction = function(){

   key=this.artist_name;

   value={

      count:1,

      acousticness:this.acousticness,

      danceability:this.danceability,

      energy:this.energy,

      liveness:this.liveness,

      loudness:this.loudness,

      speechless:this.speechless,

      tempo:this.tempo,

      valence:this.valence,

   };

   emit(key,value);

};



var reduceFunction=function(key,value){

   reduce_val={count:0, acousticness:0, danceability:0, energy:0, liveness:0, loudness:0, speechless:0,
tempo:0, valence:0};


   for (var i = 0; i<value.length; i++) {

      reduce_val.count+=value[i].count;

      reduce_val.acousticness+=value[i].acousticness;
```

```
        reduce_val.danceability+=value[i].danceability;

        reduce_val.energy+=value[i].energy;

        reduce_val.liveness+=value[i].liveness;

        reduce_val.loudness+=value[i].loudness;

        reduce_val.speechless+=value[i].speechless;

        reduce_val.tempo+=value[i].tempo;

        reduce_val.valence+=value[i].valence;

    }


    return reduce_val;

};


var finalize_func=function(key,reduce_val){

    reduce_val.avg_acousticness=(reduce_val.acousticness/reduce_val.count).toFixed(4);

    reduce_val.avg_danceability=(reduce_val.danceability/reduce_val.count).toFixed(4);

    reduce_val.avg_energy=(reduce_val.energy/reduce_val.count).toFixed(4);

    reduce_val.avg_liveness=(reduce_val.liveness/reduce_val.count).toFixed(4);

    reduce_val.avg_loudness=(reduce_val.loudness/reduce_val.count).toFixed(4);

    reduce_val.avg_speechless=(reduce_val.speechless/reduce_val.count).toFixed(4);

    reduce_val.avg_tempo=(reduce_val.tempo/reduce_val.count).toFixed(4);

    reduce_val.avg_valence=(reduce_val.valence/reduce_val.count).toFixed(4);

    return reduce_val;

};



db.getCollection('spotify_songs').mapReduce(

    mapFunction,

    reduceFunction,

    {
        out: "songs_avg",

        finalize: finalize_func

    }
);

db.songs_avg.find();
```