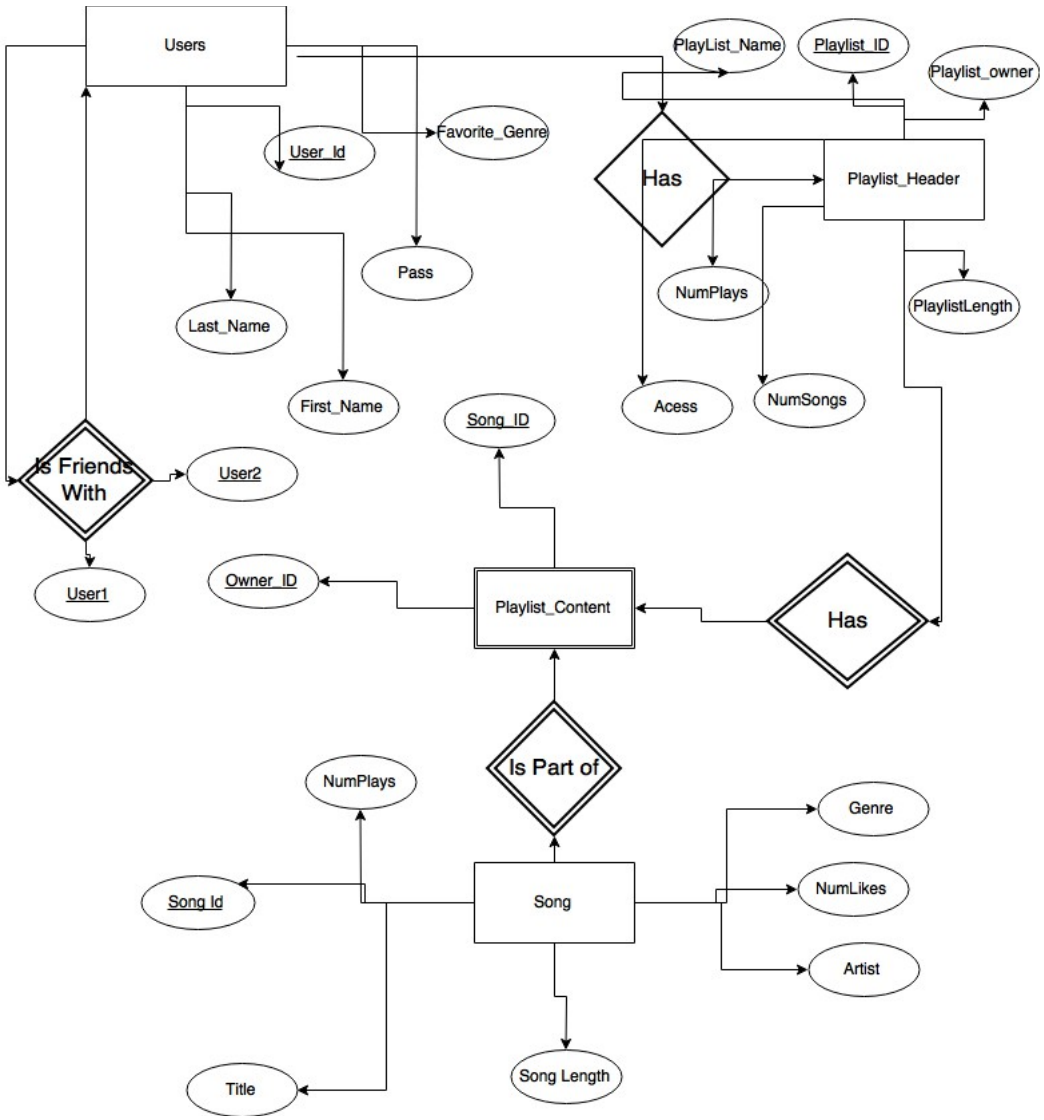
Matthew O'Connor

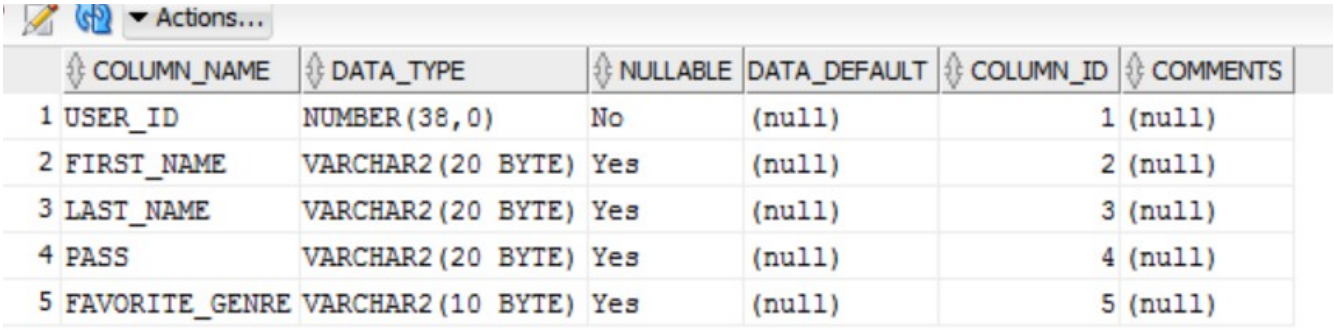
CSE 462
Final Report



There are 6 tables total: Users,Playlist Headers, Playlist Contents, Friends, Songs, and Genre. Users are people who use the system. They have an ID which is generated by the system and is an artificial primary key. They have a favorite genre which is additional functionality I would implement later, and it describes their favorite genre. They also have a first name, last name, and password. Users have friends and they create playlists which have headers. This leads us to two other tables, Friends and Playlist_Contents. A user is a friend with another user, and this is what is captured in the friends table. Every friendship is captured by two entries. If user 5 is friends with user 2, there would be an entry that has 5 as user1 and 2 as user2, and then another one that is flipped. This is to ensure you can run queries on the column user1 in order to get all of their friends. Both the user1 and user2 columns are foreign keys because they are User_ID's from users, thus friends is a weak entity. The reason friends exists is because we want to be able to find all playlists a user can access, and this leads us to the playlist header table. The playlist header table contain all information about a playlist besides the actual songs contained in it. Its primary key is a unique playlist ID that would be artificially generated by the system. It would contain one foreign key, which is the user that owns the playlist. It would also have an access level (spelled acess because access is reserved), of public private or friends. A public playlist would be accessible to anyone, a private playlist would be accessible only to the user who made it, and a friends play list would be accessible to a user and their friends. Playlist_Content simply matches a

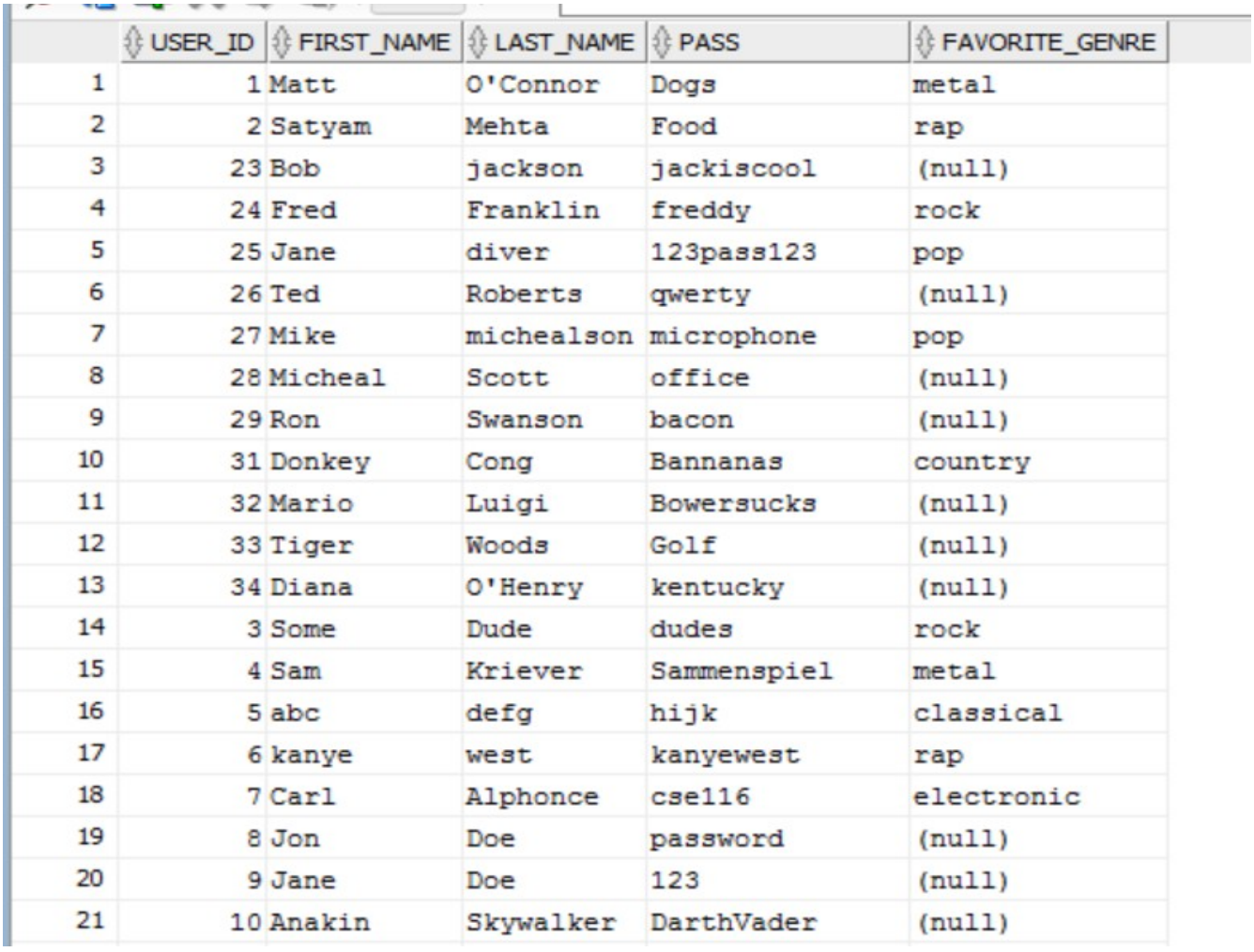
Playlist_ID to the Song_ID of every song it contains. The primary key is the whole relationship and both things in Playlist_Content are foreign keys. Playlist_Content exists because in the first version of my tables I realized I needed to split off the contents from the playlists themselves, as the contents could be rather extensive. Contents combined with playlist headers allow a user to not only look up all songs they own, but what songs their friends are allowing them to see as well. The songs table is simple, it contains data about the songs themselves, but threw in some columns such as song likes and song plays, so users could see which songs were the most popular. Finally an additional genre table exists as a reference for what genres are available for favorite genre and song genre. All my tables are in the class database but ill be providing the first 20 or so rows just for reference.

Users:


Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	USER_ID	NUMBER(38,0)	No	(null)	1	(null)
2	FIRST_NAME	VARCHAR2(20 BYTE)	Yes	(null)	2	(null)
3	LAST_NAME	VARCHAR2(20 BYTE)	Yes	(null)	3	(null)
4	PASS	VARCHAR2(20 BYTE)	Yes	(null)	4	(null)
5	FAVORITE_GENRE	VARCHAR2(10 BYTE)	Yes	(null)	5	(null)

Friends:



	USER_ID	FIRST_NAME	LAST_NAME	PASS	FAVORITE_GENRE
1	1	Matt	O'Connor	Dogs	metal
2	2	Satyam	Mehta	Food	rap
3	23	Bob	jackson	jackiscool	(null)
4	24	Fred	Franklin	freddy	rock
5	25	Jane	diver	123pass123	pop
6	26	Ted	Roberts	qwerty	(null)
7	27	Mike	michealson	microphone	pop
8	28	Micheal	Scott	office	(null)
9	29	Ron	Swanson	bacon	(null)
10	31	Donkey	Cong	Bannanas	country
11	32	Mario	Luigi	Bowersucks	(null)
12	33	Tiger	Woods	Golf	(null)
13	34	Diana	O'Henry	kentucky	(null)
14	3	Some	Dude	dudes	rock
15	4	Sam	Kriever	Sammenspiel	metal
16	5	abc	defg	hijk	classical
17	6	kanye	west	kanyewest	rap
18	7	Carl	Alphonce	cse116	electronic
19	8	Jon	Doe	password	(null)
20	9	Jane	Doe	123	(null)
21	10	Anakin	Skywalker	DarthVader	(null)

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	USER1	NUMBER(38,0)	No	(null)	1	(null)
2	USER2	NUMBER(38,0)	No	(null)	2	(null)

	USER1	USER2
1	1	2
2	1	3
3	1	4
4	1	5
5	1	6
6	1	7
7	1	8
8	1	9
9	1	10
10	1	11
11	2	1
12	3	1
13	4	1
14	5	1
15	6	1
16	7	1
17	8	1
18	9	1
19	10	1
20	11	1
21	10	11
22	11	10

	USER1	USER2
79	20	13
80	20	14
81	20	15
82	20	16
83	20	17
84	20	18
85	20	19
86	1	20
87	2	20
88	3	20
89	4	20
90	5	20
91	6	20
92	7	20
93	8	20
94	9	20
95	10	20
96	11	20
97	12	20
98	13	20
99	14	20
100	15	20

Playlist_Headers

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	PLAYLIST_ID	NUMBER(38,0)	No	(null)	1	(null)
2	PLAYLIST_OWNER	NUMBER(38,0)	Yes	(null)	2	(null)
3	PLAYLIST_NAME	VARCHAR2(50 BYTE)	Yes	(null)	3	(null)
4	NUMSONGS	NUMBER(38,0)	Yes	(null)	4	(null)
5	PLAYLIST_LENGTH	NUMBER(20,0)	Yes	(null)	5	(null)
6	NUMPLAYS	NUMBER(38,0)	Yes	(null)	6	(null)
7	ACESS	VARCHAR2(20 BYTE)	Yes	(null)	7	(null)

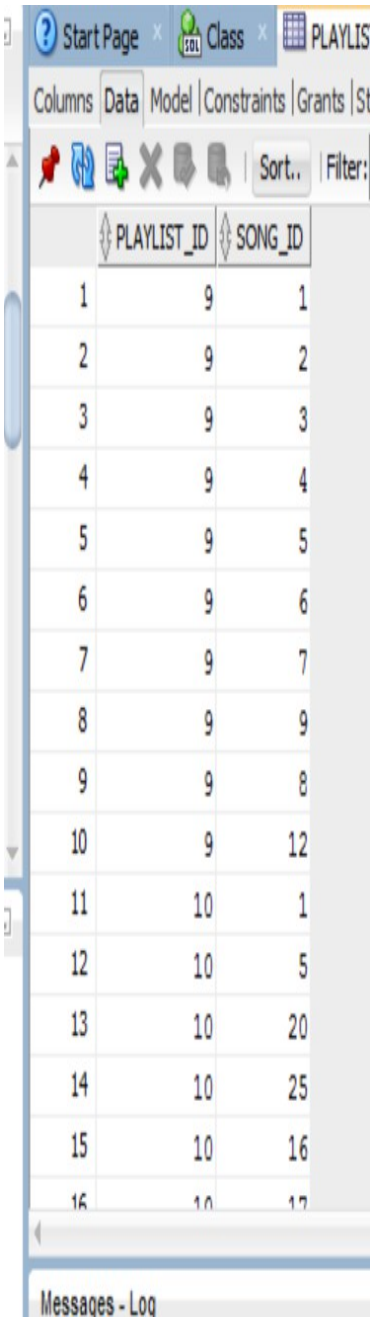
Sort... Filter:

	PLAYLIST_OWNER	PLAYLIST_ID	PLAYLIST_NAME	NUMSONGS	PLAYLIST_LENGTH	NUMPLAYS	ACESS
1	1	1	Jams	(null)	(null)	2	Public
2	1	2	More jams	(null)	(null)	3	Public
3	1	3	Stuff	(null)	(null)	4	Private
4	2	4	Songs	(null)	(null)	5	Friends
5	2	5	Songz	(null)	(null)	6	Private
6	3	6	Pop	(null)	(null)	1	Public
7	4	7	Rock	(null)	(null)	3	Private
8	6	8	Metal	(null)	(null)	10	(null)
9	9	9	Plays	(null)	(null)	123	Public
10	22	10	rock	(null)	(null)	12	Public
11	12	11	musiclist	(null)	(null)	100	Private
12	15	12	my songs	(null)	(null)	9001	Friends
13	18	13	lotsofsongs	(null)	(null)	10	Friends
14	30	14	untitled	(null)	(null)	1123	Private
15	27	15	wubs	(null)	(null)	111	Public

Playlist_Contents

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	PLAYLIST_ID	NUMBER(38,0)	No	(null)	1	(null)
2	SONG_ID	NUMBER(38,0)	No	(null)	2	(null)



	PLAYLIST_ID	SONG_ID
1	9	1
2	9	2
3	9	3
4	9	4
5	9	5
6	9	6
7	9	7
8	9	9
9	9	8
10	9	12
11	10	1
12	10	5
13	10	20
14	10	25
15	10	16

	PLAYLIST_ID	SONG_ID
37	14	22
38	15	1
39	15	2
40	15	3
41	15	4
42	15	5
43	15	6
44	15	7
45	15	8
46	15	9
47	15	10
48	15	11
49	15	12
50	15	13
51	15	14
52	15	15
53	15	16
54	15	17
55	15	18
56	15	19
57	15	20
58	15	21

Songs

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	SONG_ID	NUMBER(38,0)	No	(null)	1	(null)
2	SONGLENGTH	NUMBER(20,0)	Yes	(null)	2	(null)
3	TITLE	VARCHAR2(100 BYTE)	Yes	(null)	3	(null)
4	ARTIST	VARCHAR2(100 BYTE)	Yes	(null)	4	(null)
5	ALBUM	VARCHAR2(100 BYTE)	Yes	(null)	5	(null)
6	GENRE	VARCHAR2(100 BYTE)	Yes	(null)	6	(null)
7	NUMPLAYS	NUMBER(38,0)	Yes	(null)	7	(null)
8	NUMLIKES	NUMBER(38,0)	Yes	(null)	8	(null)

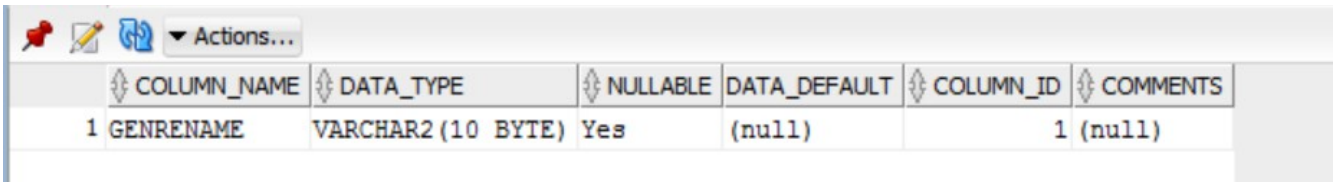
Start Page | Class | SONGS

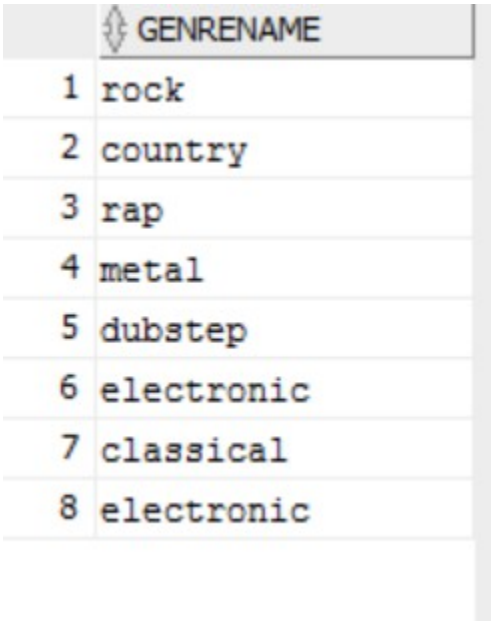
Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Sort.. | Filter: | Actions...

	SONG_ID	TITLE	ARTIST	ALBUM	SONGLEN...	GENRE	NUMPLAYS	NUMLIKES
1	21	God Went North	Nothing...	Nothing More	211	rock	247	89
2	22	War Of the Gods	Amon Am...	Surtur Rising	777	metal	399	111
3	23	Dollhouse	Melanie...	Dollhouse EP	357	(null)	999999	999
4	24	Diamond Eyes	Shinedown	Somewhere in the st...	400	rock	12321	1232
5	25	If I Fall	Five Fi...	American Capitalist	567	rock	123213213	12321390
6	12	Weapon	Bastille	VS	400	rock	222	5
7	13	Axe to Grind	Bastille	VS	500	rock	333	80
8	14	Flaws	Bastille	VS	436	rock	222222	440
9	15	Her Kiss	In This...	The Dream	347	rock	999	99
10	16	Bully	Shinedown	Bully	400	pop	298329	276
11	17	Hear Me Now	Hollywo...	American Tragedy	411	pop	99996	42
12	18	So Cold	Breakin...	Phobia	499	rock	1234	123
13	19	X Rated	Excision	X Rated	600	dubstep	90	1
14	20	Kingpin	Skream	All Trap Music 2	663	dubstep	456	23
15	1	Shots	Imagine...	Smoke+Mirrors	187	rock	1000	100
16	2	Gold	Imagine...	Smoke+Mirrors	120	rock	200	50
17	3	Radioactive	Imagine...	Radioactive	300	rock	20000	400
18	4	Savior	Rise Ag...	Appeal to Reason	320	rock	99999	999
19	5	Re Education (Through...	Rise Ag...	Appeal to Reason	333	rock	12312	123
20	6	abc	ghi	def	222	pop	111	11
21	7	Angel of Death	Slayer	Reign in blood	333	metal	200	22
22	8	Drones	Muse	Drones	345	rock	234	1000

Genre

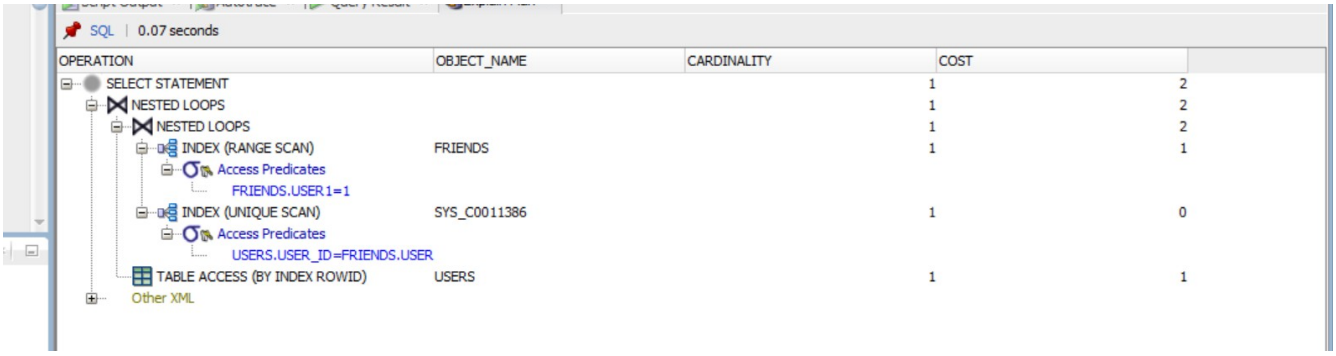




Functionality. A lot of the functionality was described in the previous report so I included it in here. The functionality of the project so far really just keeps track of songs, users,playlists, and friends. I know what users are friends with each other. I know what songs are in each playlist. I know who owns what playlist. I know what the most and least liked songs are (since I artificially inputted it). I know how many songs are in each playlist. But the main functionality is users, their friends, their playlists, and the songs in each playlist. The following queries highlight some of this functionality.

Select USers.First_Name
From USERS, Friends
Where (Friends.User1 = 1) AND (Users.USER_ID = friends.USER2);

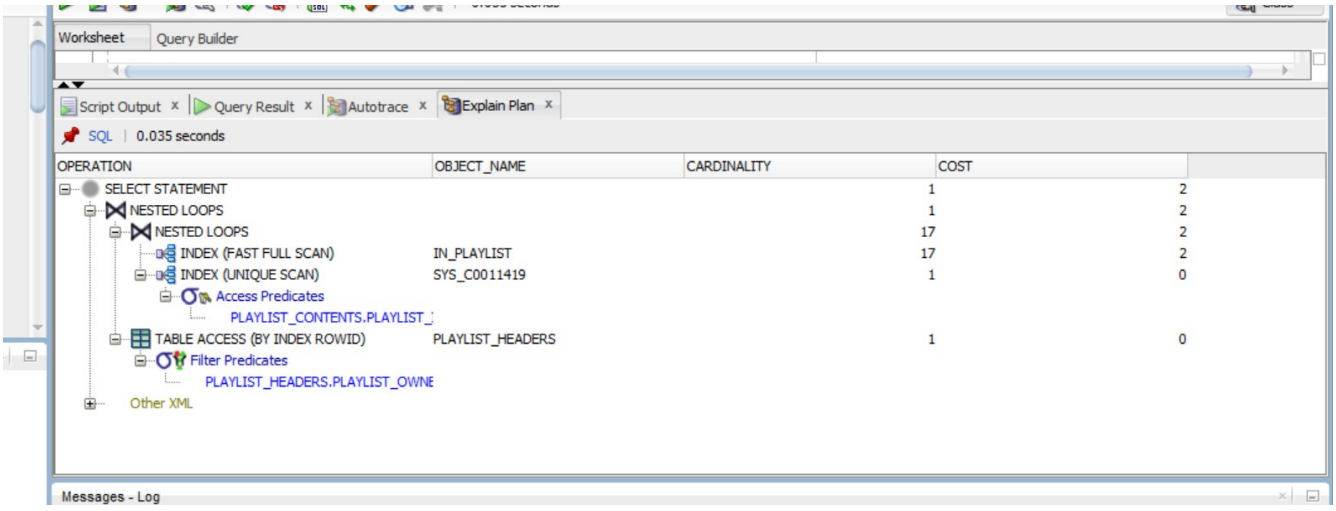
This Query finds the name of all users that the user with USER_ID 1 is friends with. It uses the friends table to get all the users that user 1 is friends with and then it matches those ID's up with the users table to get the name of those friends. The friends table is over 100 values long. Since no full table scans are being performed and all the costs are 2 or lower, this is a very efficient query. It is accessing the user 1 column in friends and the user_id column in users as well as the user 2 column in friends. The rang scan and unique scan indexes are being used to make this very fast.



Select Playlist_Contents.Song_ID
From Playlist_Headers, Playlist_Contents
Where (Playlist_Headers.Playlist_Owner = 1) AND (Playlist_Contents.Playlist_ID = Playlist_Headers.Playlist_ID);

This query returns all songs that a certain user (In this case the user with user_id=1) has in their play lists. It finds all playlists that a person owns and then matches up all songs in those play lists. This scan is optimized since all the actions are of a low cost. Playlist_Contents only has two values , and these values are indexed, so the full table scan over the index is very cost effective (cost 2) despite its high cardinality of 17. This is due to the fast full scan index on playlist contents, and this makes the query nice and fast.

We care about all songs in all playlists by a user had because I was going to run some additional metrics about a users favorite song, but I didn't manage to implement the functionality. I was thinking about building on this, because if I had all songs a user has in playlists, I could then reference ACESS in the play list column to to find all songs a user had and all songs their friends had, that was the idea anyways.

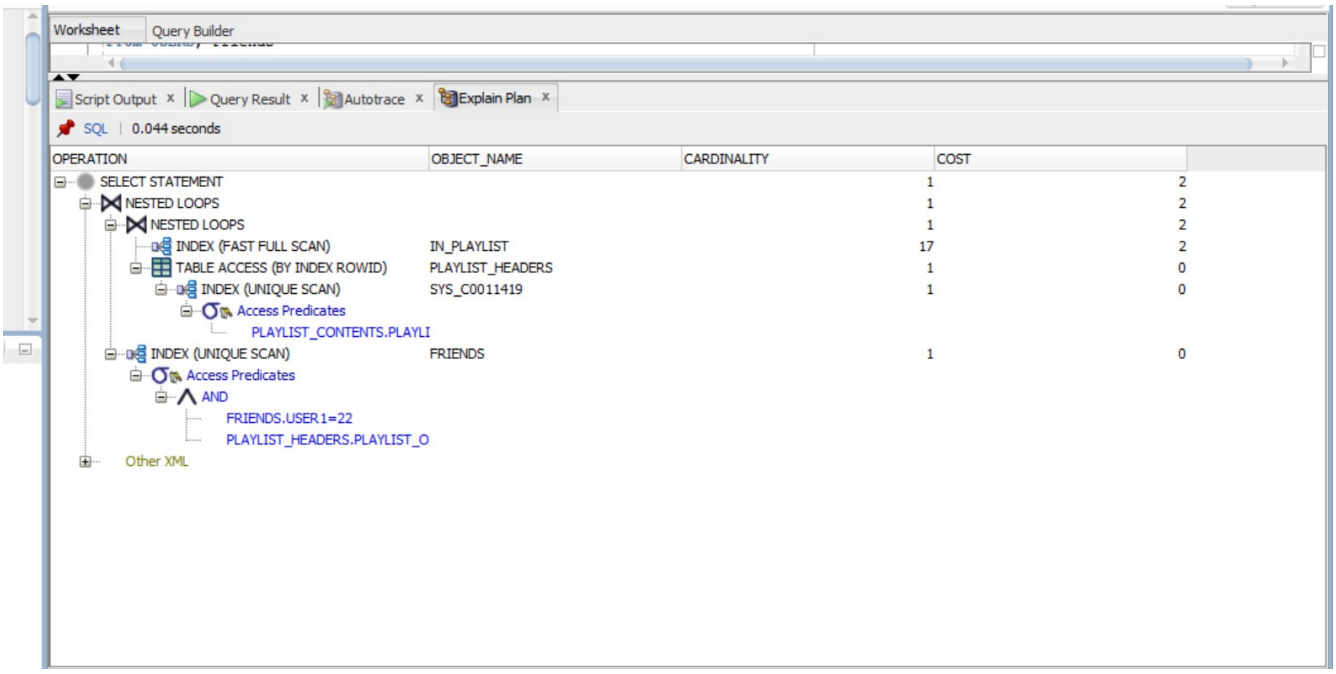


Select Playlist_Contents.Song_ID,Playlist_Contents.PLAYLIST_ID
From Playlist_Headers, Playlist_Contents, Friends
Where (FRIENDS.User1=22) AND (Playlist_Headers.Playlist_Owner =USER2) AND (Playlist_Contents.Playlist_ID = Playlist_Headers.Playlist_ID);

This is a more advanced version of the previous query. Instead of having the Owner_Id decided for it, the owner_id it is using are all the friends of the user (User 22 in this case). This query returns all songs

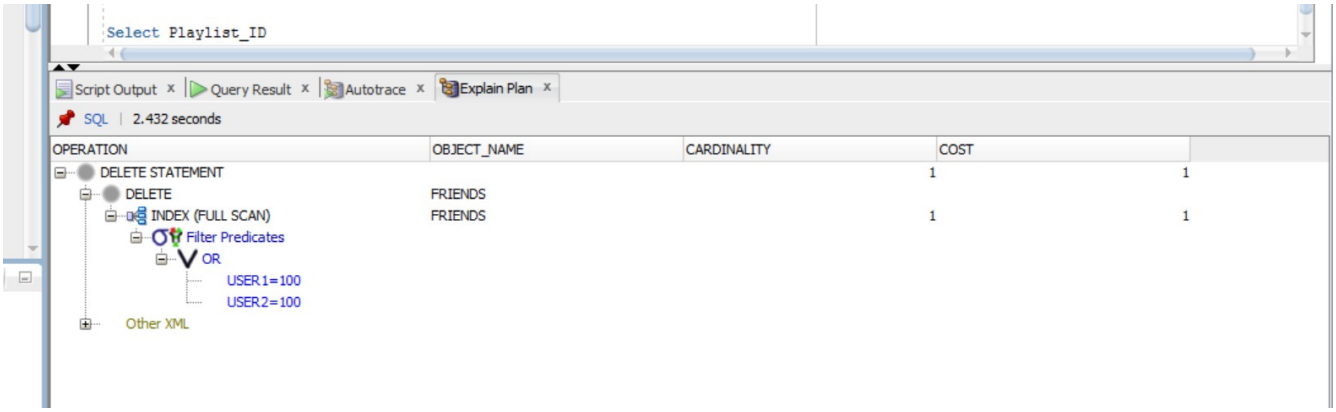
and the playlists they belong to of user 22. It is still running everything in very low costs because this scan is using indexes. Everything is cost 2 or lower. The friends table is being accessed, and a full table scan on that would cost over 100, so this query is optimized, due to using similar indexes and rows to the previous query.

This is definitely not optimal but the idea here was to find all songs your friends had, hence why we care about song id because from song id song name can be given. If you found all songs your friends had in playlists, a method could be written to then sort the songs by number of plays or number of likes. I am not very good at Pl/SQL but I think something could be written in it that would allow me to return all songs a user and their friends had, store this group of songs as some kind of song network, and then find the most popular and trending songs in the network.



Delete From Friends
Where (User1 = 100)or (User2=100);

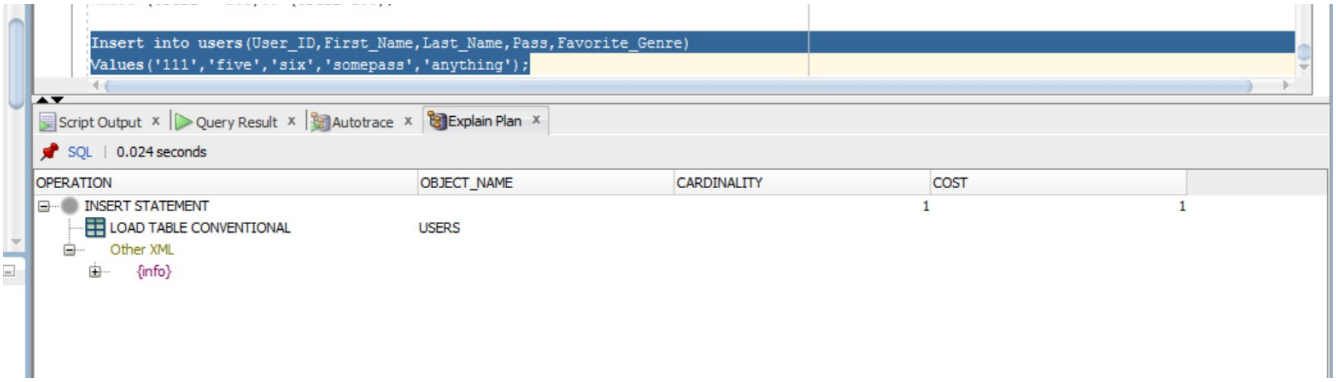
A delete query to remove all friendships involving user with user_Id=100 in the friendship table. This query is very optimized due to friends being indexed, despite a full scan taking place, it takes place on an index, thus preventing the full scan and making this fast.



Insert into users(User_ID,First_Name,Last_Name,Pass,Favorite_Genre)
Values('111','five','six','somepass','anything');

Inserts a new user into the system. Inserting a new user is just one action so this is very optimal. I can't really get faster than one here.

I Created a Genres table for reference but I need to add functionality so that only genres from the genre table can inserted, as of now any value can be inserted.



More queries besides the original 5:

Select User2
From friends
Where user1=2;

Gets all of user 1's friends.

Select USers.First_Name, Users.Last_Name
From USERS, Friends
Where (Friends.User1 = 21) AND (Users.USER_ID = friends.USER2);
Gets all the first names and last names of user 1's friends.

Select Playlist_ID
From Playlist_Headers
Where Playlist_Owner = 1;

Gets all playlists owned by user 1.

```
Select Title
from songs
where (Artist like '%grace');
```

Gets all songs by all artists ending in grace (In this case Three days grace).

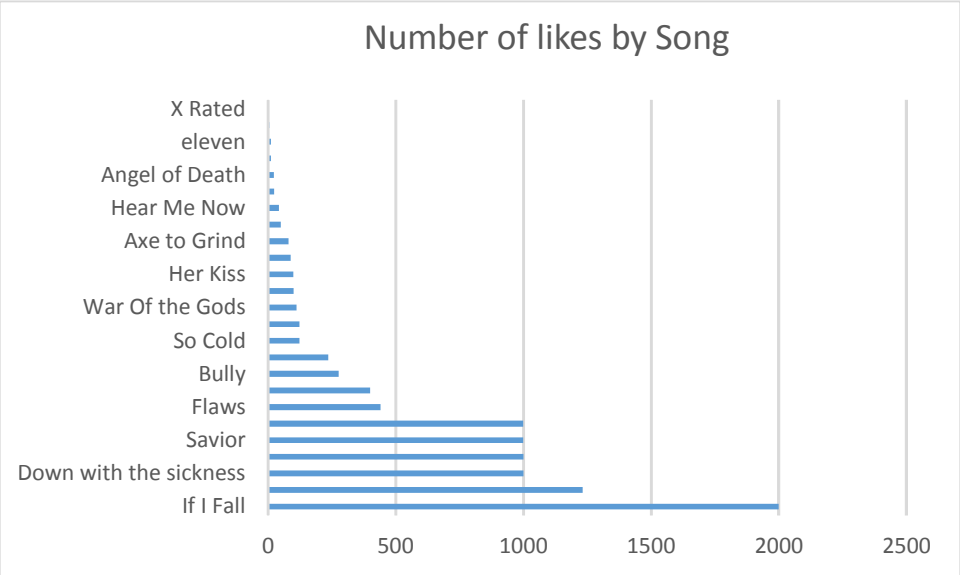
```
Select Title, numlikes
from songs
order by numlikes DESC;
```

Gets number of likes by song.

Implementation.

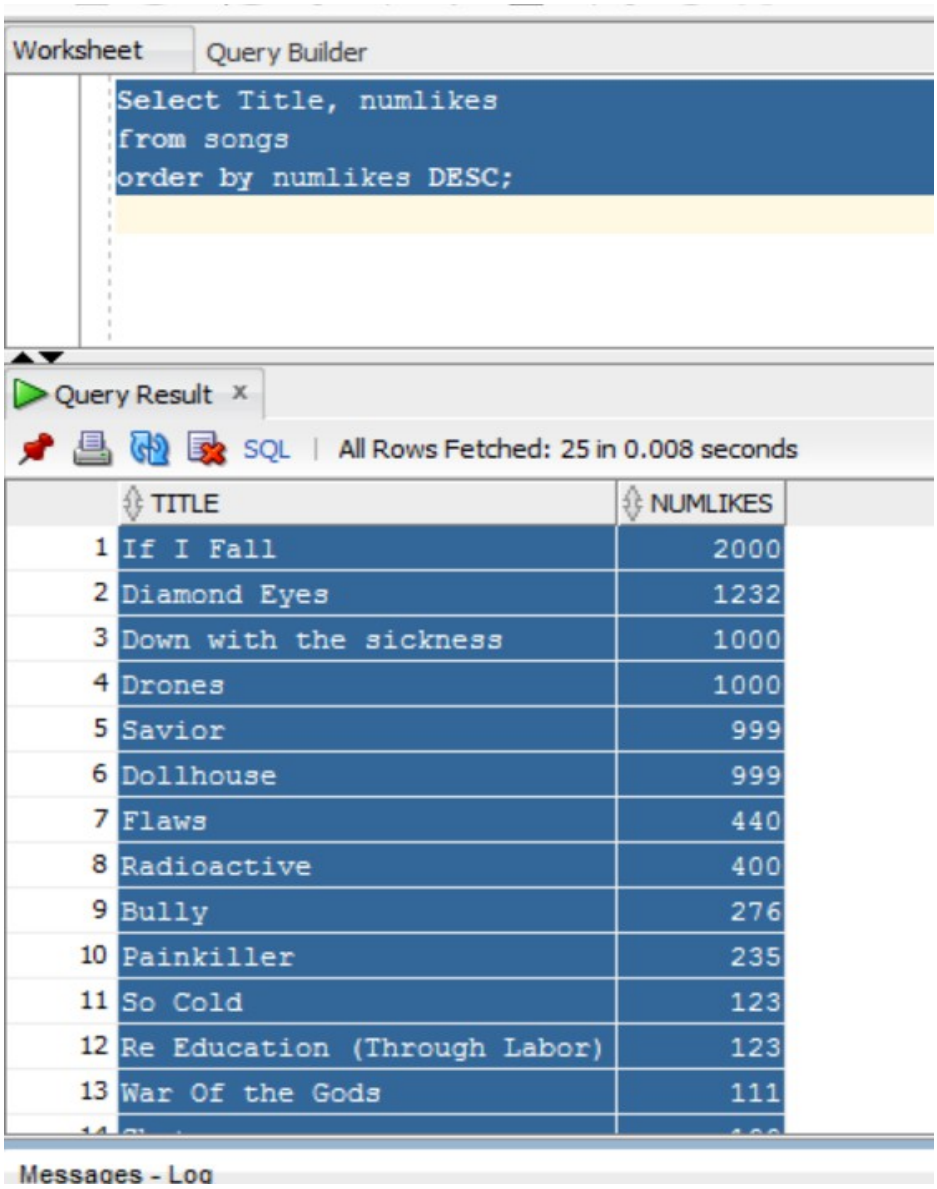
The graphs are mock ups of what I would like the system to be able to output. These are mock ups reports that I would like the system to be able to generate. I put them in graph form in order to nicely show data in the system.

Here is a query of the number of songs by their number of likes.



TITLE	NUMLIKES
If I Fall	2000
Diamond Eyes	1232
Down with the sickness	1000
Drones	1000
Savior	999
Dollhouse	999
Flaws	440
Radioactive	400
Bully	276
Painkiller	235
So Cold	123
Re Education (Through Labor)	123
War Of the Gods	111
Shots	100
Her Kiss	99
God Went North	89
Axe to Grind	80
Gold	50
Hear Me Now	42

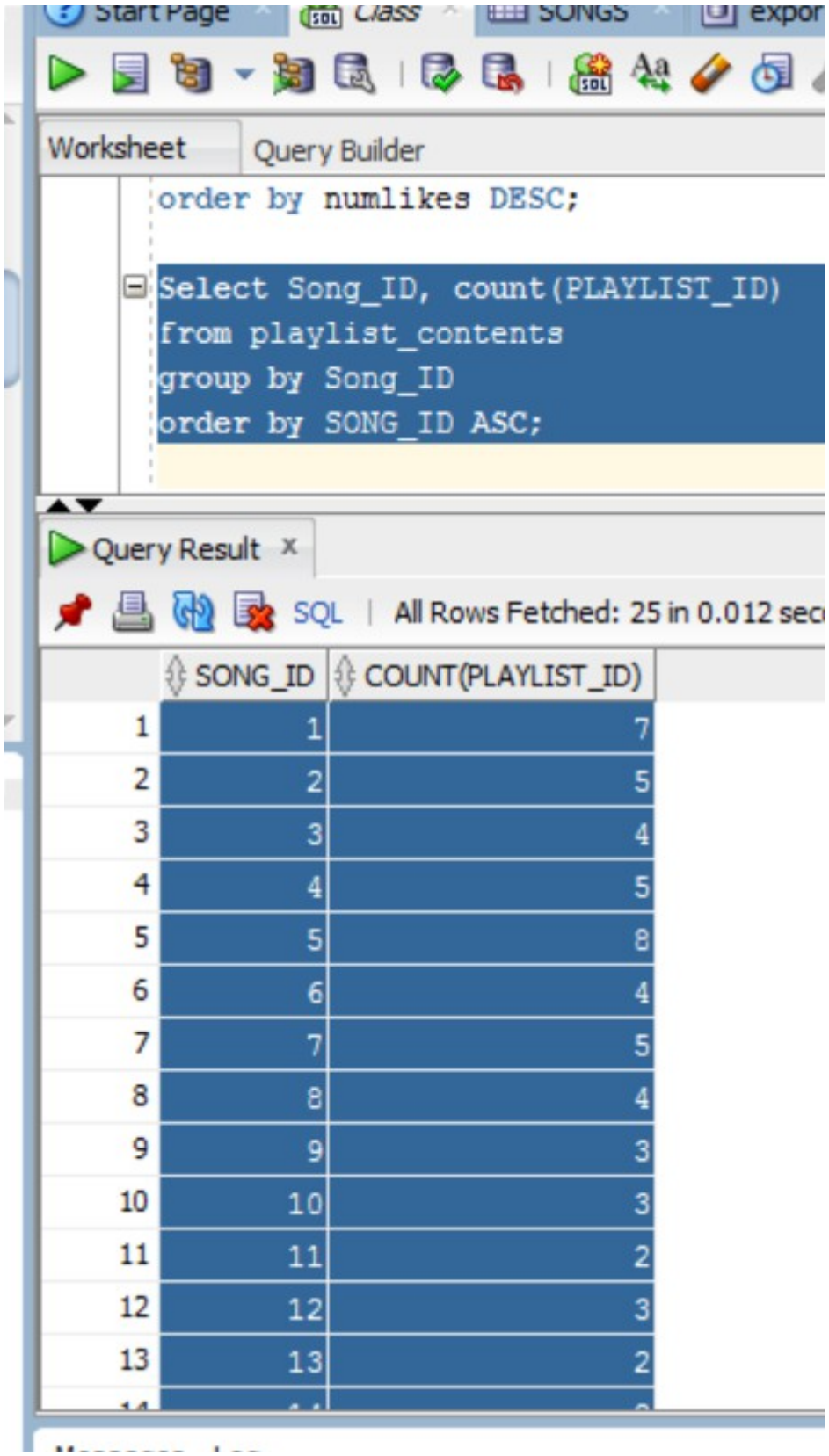
Kingpin	23
Angel of Death	22
abc	11
eleven	11
Weapon	5
X Rated	1



```
Select Song_ID, count(PLAYLIST_ID)
from playlist_contents
group by Song_ID
order by SONG_ID ASC;
```

Query to get a count of how many playlists a song appears in. Takes songs id's gets a count of the playlists they appear in and then groups them in an ascending order. The idea of this query would be to use this for further analytics of the most popular song to put in a playlist, and then maybe make some kind of public system generated playlist of the most popular songs or something.

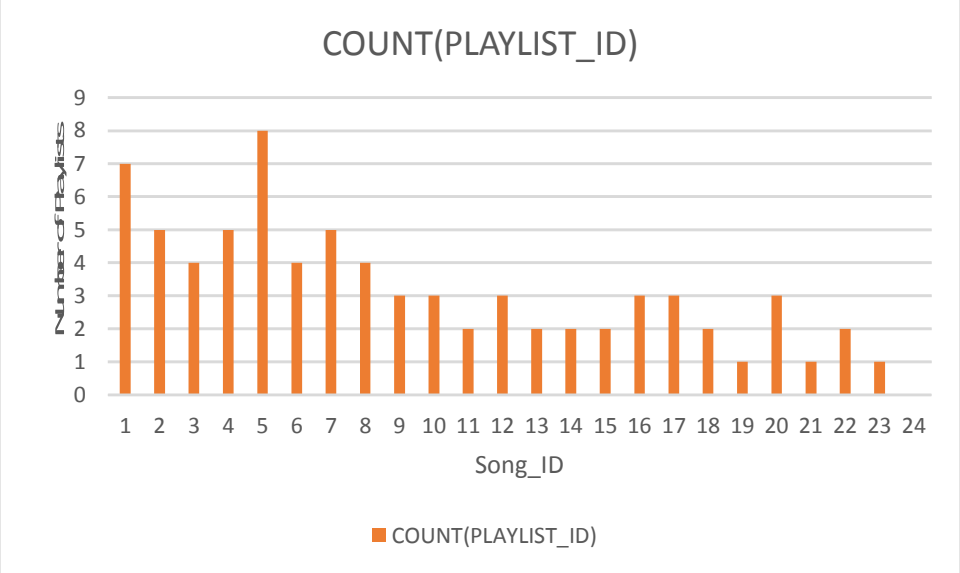
Screenshot


Start Page
Class
SONGS
Worksheet
Query Builder
order by numlikes DESC;
Select Song_ID, count(PLAYLIST_ID)
from playlist_contents
group by Song_ID
order by SONG_ID ASC;
Query Result
SQL
All Rows Fetched: 25 in 0.012 sec
SONG_ID
COUNT(PLAYLIST_ID)
1 1 7
2 2 5
3 3 4
4 4 5
5 5 8
6 6 4
7 7 5
8 8 4
9 9 3
10 10 3
11 11 2
12 12 3
13 13 2

data:

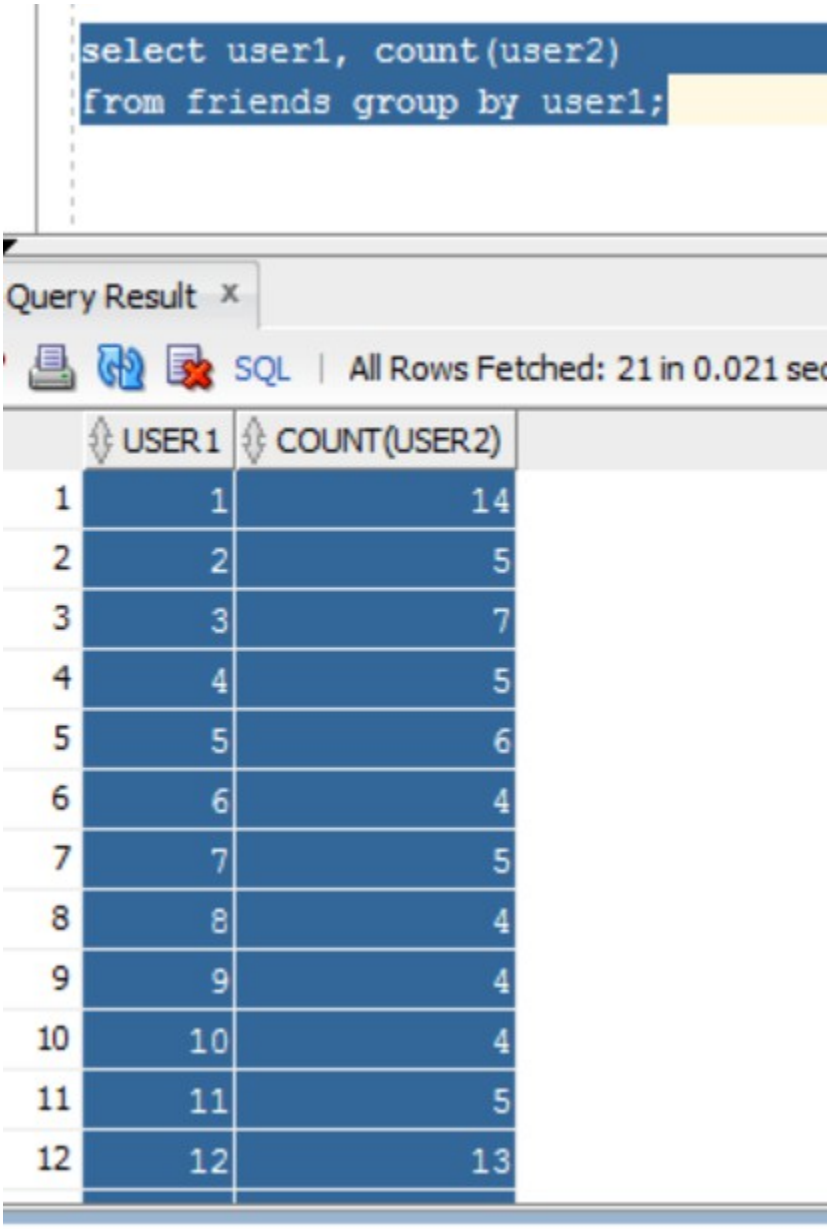
SONG_ID	COUNT(PLAYLIST_ID)
1	7
2	5
3	4
4	5
5	8
6	4
7	5
8	4
9	3
10	3
11	2
12	3
13	2
14	2
15	2
16	3
17	3
18	2
19	1
20	3
21	1
22	2
23	1

UI Chart mockup



```
select user1, count(user2)
from friends group by user1;
```

Query to get how many friends each user_id has, and therefore each user. Users that do not appear have no friends.

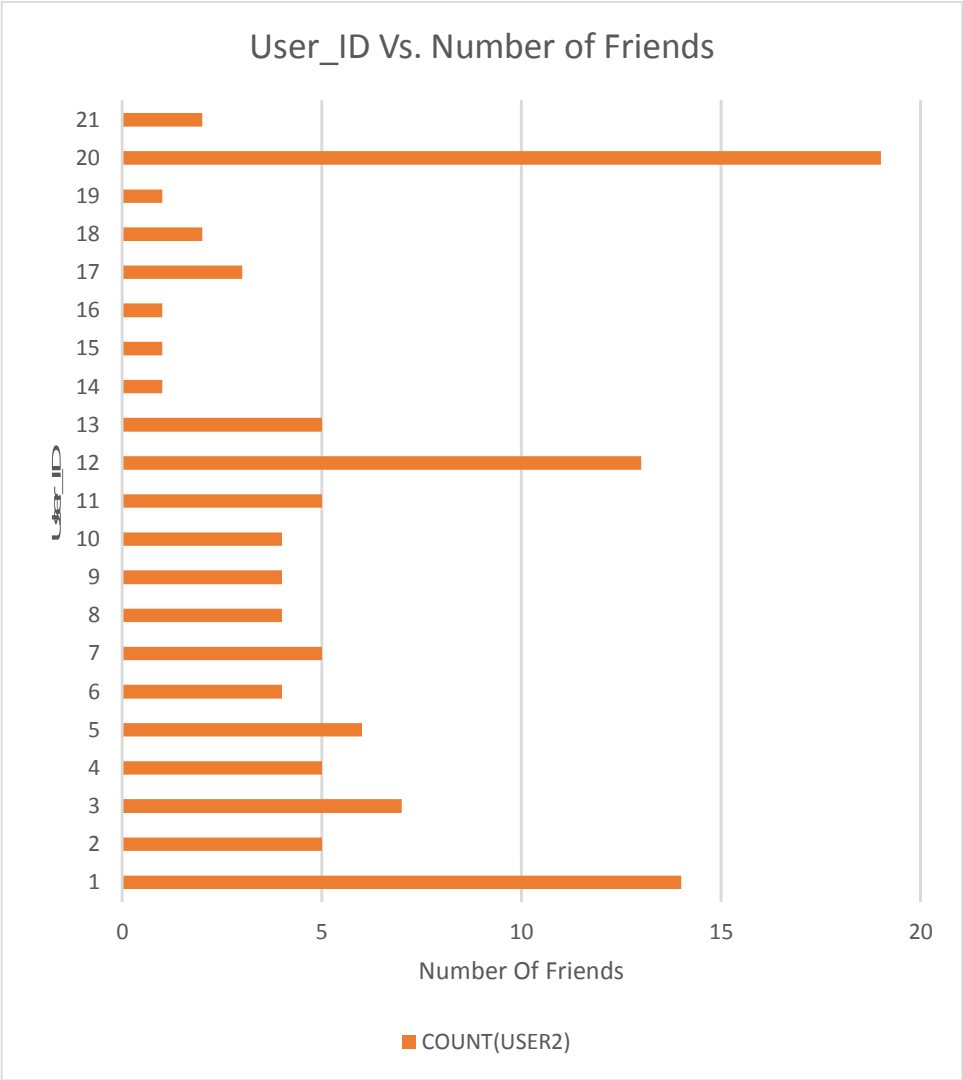


Query Result

SQL | All Rows Fetched: 21 in 0.021 sec

	USER1	COUNT(USER2)
1	1	14
2	2	5
3	3	7
4	4	5
5	5	6
6	6	4
7	7	5
8	8	4
9	9	4
10	10	4
11	11	5
12	12	13

USER1	COUNT(USER2)
1	14
2	5
3	7
4	5
5	6
6	4
7	5
8	4
9	4
10	4
11	5
12	13
13	5
14	1
15	1
16	1
17	3
18	2
19	1
20	19
22	2



Future implementation Overall there is a lot more I had in mind for the project. The overall idea was not just to store users and their friends and playlists (and the songs that made up that playlists), but to be able to analyze which songs were the most most popular, who liked what kinds of songs, what songs were trending within groups of users,what users favorite genres were(without inputting it manually), and to be able to keep some nice stats on everything. As of now the system really just keeps track of users, playlists, who is friends with who, who owns what playlist, and what is in each playlist. I would have really liked to have implemented the trending in your network feature that Spotify has. It lists which songs are popular with your friends so you can check out what your friends are listening to. I also would have liked to implement the discover feature. Discover helps you find new songs based on what you already like. I also wanted to create a PL/SQL function to create playlists, but as of now now it would require manual entry. I would have also liked to have made triggers to update when a song is played and when a song is liked and to manually send that data to the database through the user interface.