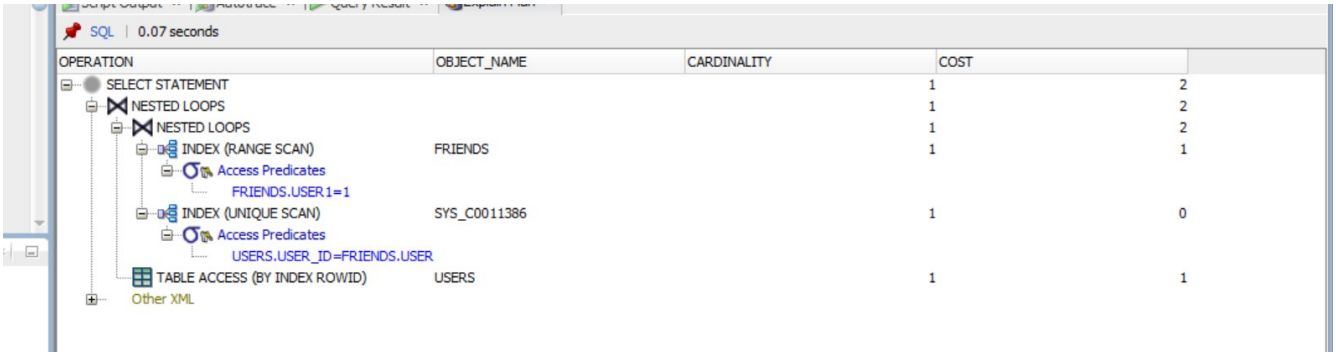
Matthew O'Connor
Explain Plans
And
Queries

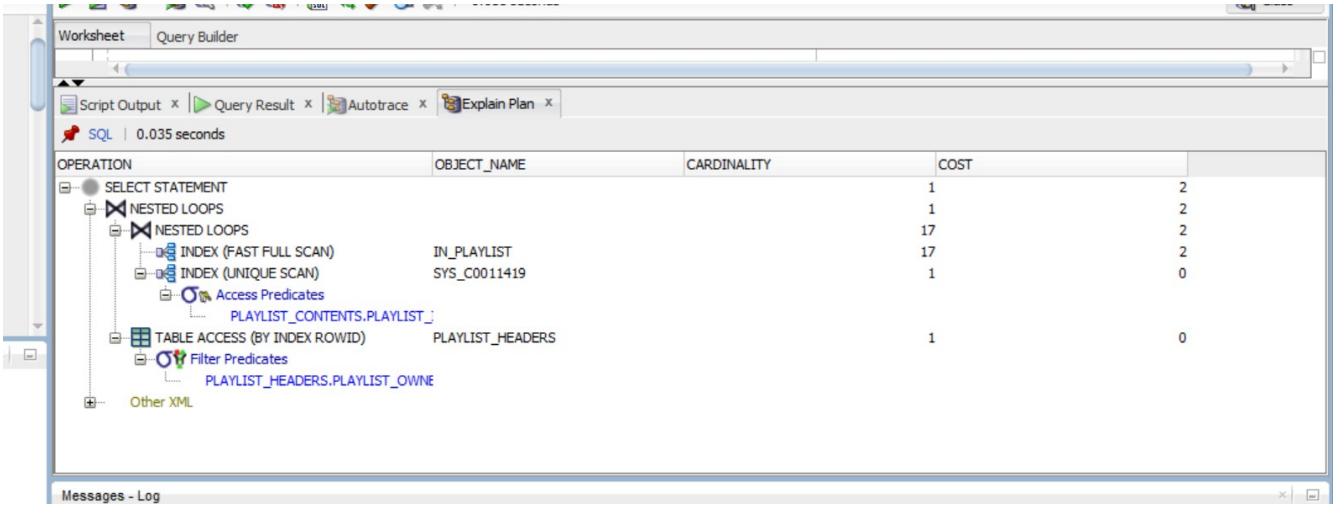
Select USers.First_Name
From USERS, Friends
Where (Friends.User1 = 1) AND (Users.USER_ID = friends.USER2);

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.



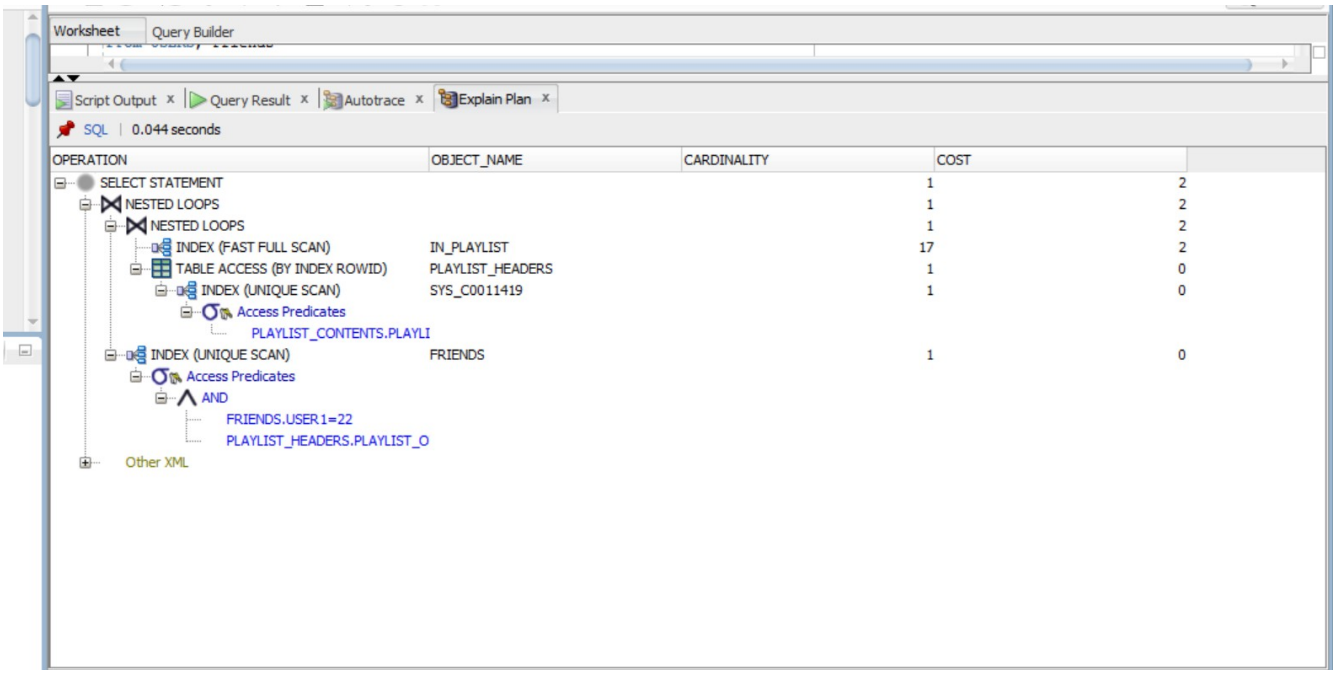
Select Playlist_Contents.Song_ID
From Playlist_Headers, Playlist_Contents
Where (Playlist_Headers.Playlist_Owner = 1) AND (Playlist_Contents.Playlist_ID = Playlist_Headers.Playlist_ID);

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.



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);

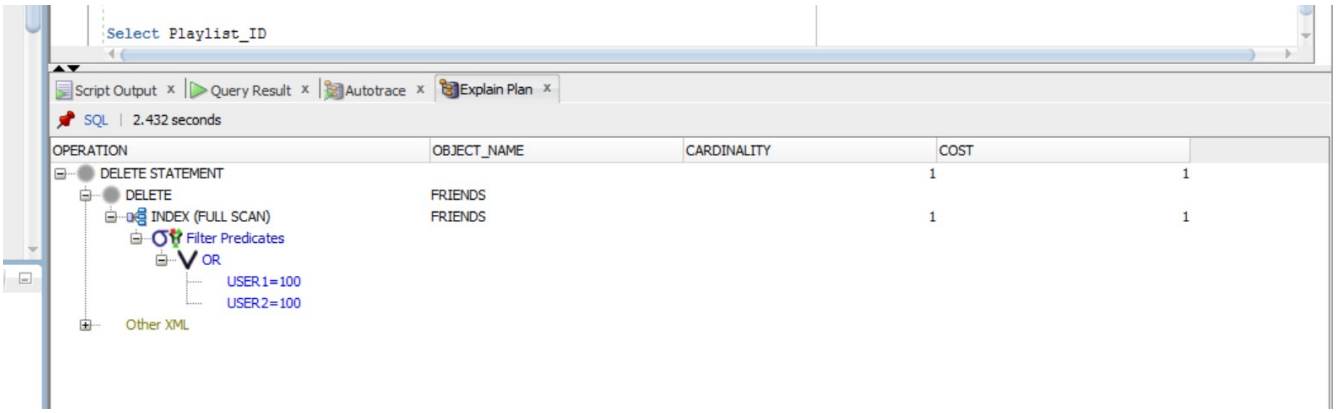
This is a more advanced version of the previous query. Instead of having the Owner_Id decided for it, the owner_id it is using are all the friends of the user (User 22 in this case). This query returns all songs and the playlists they belong to of user 22. It is still running everything in very low costs 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.



Delete From Friends

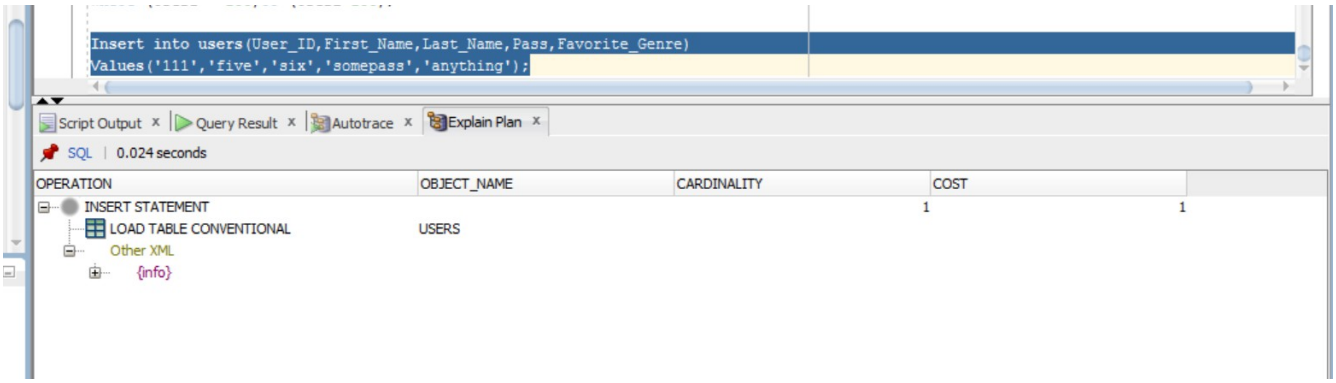
Where (User1 = 100)or (User2=100);

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.



So that's five queries with explain plans and I believe that's what you asked for, so as far as grading goes I think this is all I am supposed to submit? I'm just gonna throw in some more queries as bonus(without explain plans) feel free to ignore this.

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).