COMP 2670-09

Custom SoundCloud Database

Spring 2016 – Most recent Web project

Cannistraro, Derek
4-11-2016

The Problem?

SoundCloud is a great web service that provides what is essentially a social networking site for music. It allows users to upload, share, and discover new sounds from around the world! However, SoundCloud doesn't allow you to find music in just any way you want and you have to use their search options when looking for new music. By bringing SoundCloud tracks into a separate database, a solution can be found that allows users to search through music however they please. Entities that were extracted include the Tracks with all their information, Users and basic info, Genres, Playlists and what tracks are on them, comments for all tracks, playlist and track duration time, Groups and the users that belong to each group. All Groups, Tracks, and Playlists have a User that created them, all Tracks have a specified Genre, and all Groups have a specified number of Users that are kept track of. The database was built up on this general basis.

ER Diagram

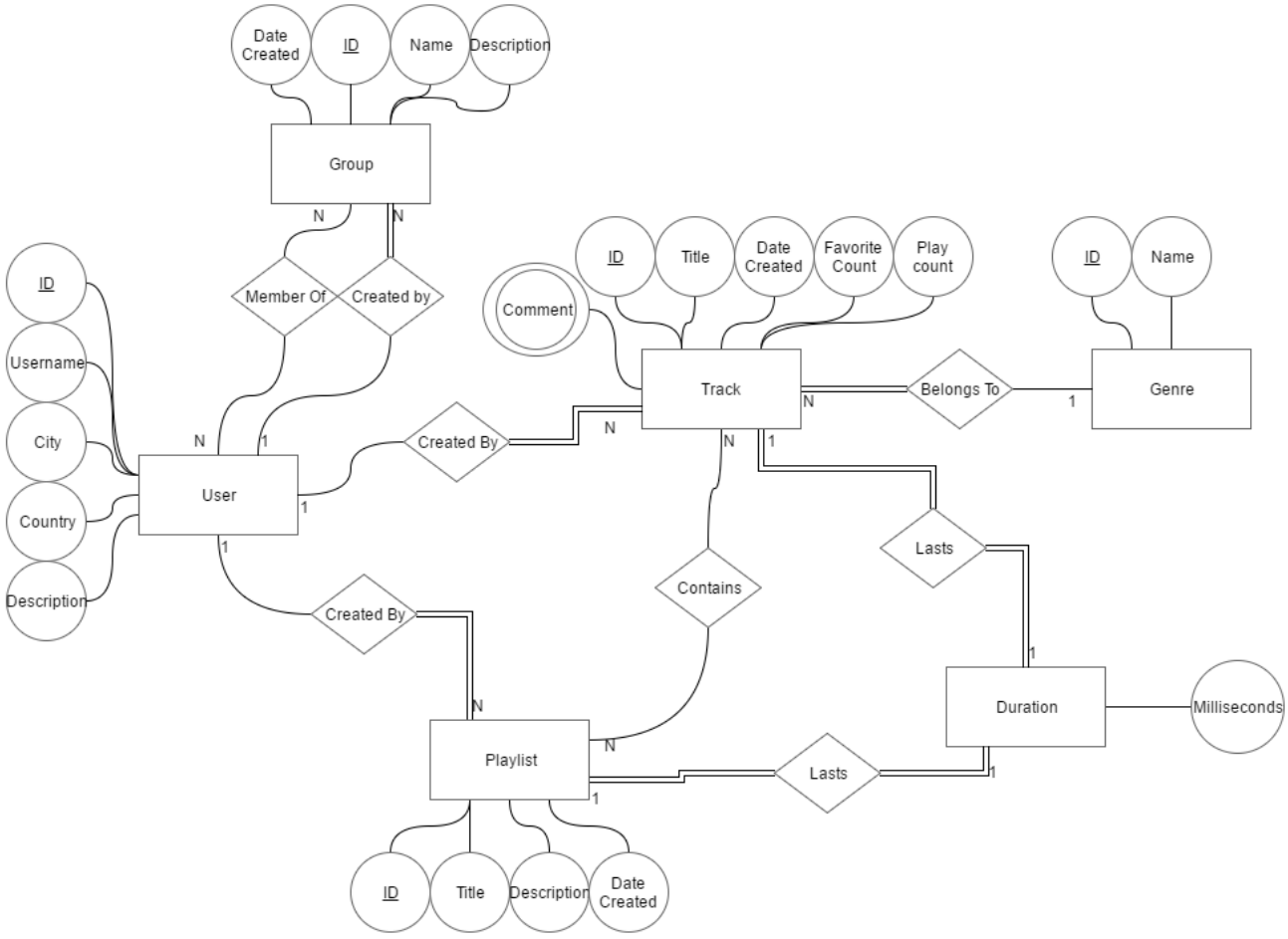


Figure 1

Normalized Relation Table

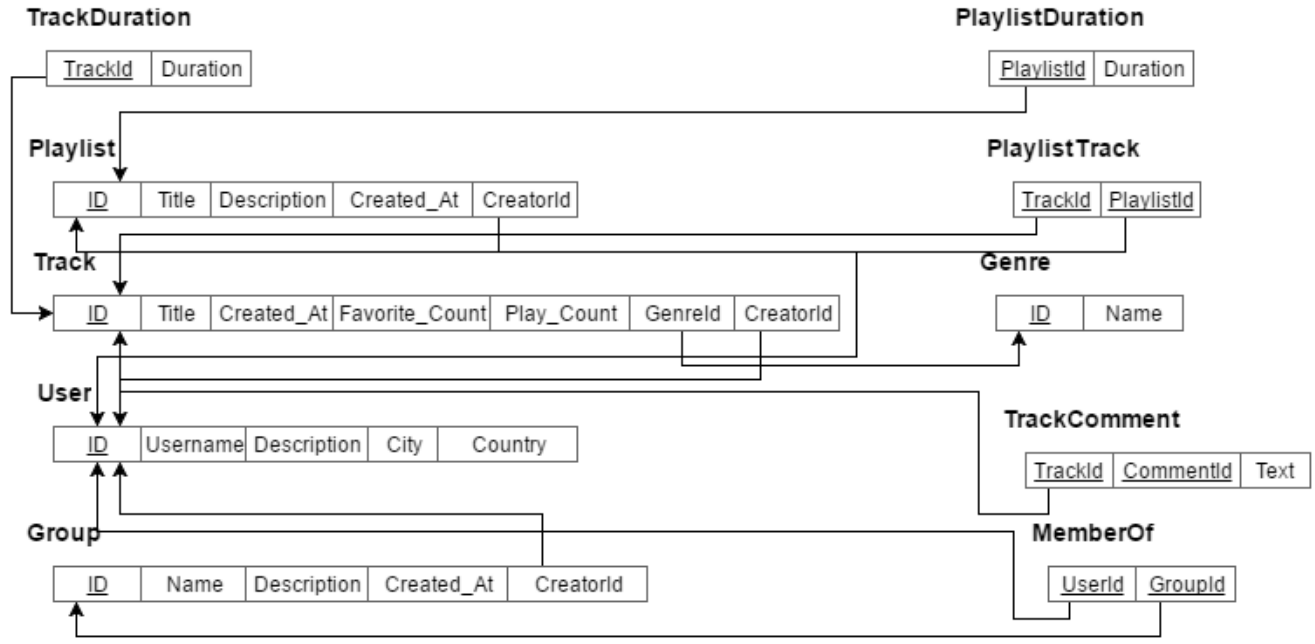


Figure 2

Physical Design

All of the tables were structured just as is shown above in the database. All pages on the site can be accessed from the menu shown below. Data was loaded into the database through the Soundcloud API using JavaScript and PHP Code. Playlists and Tracks can be added in manually by the User on an html page which gives them the ability to load a maximum of 200 tracks or 20 playlists in at a time and select by a keyword search.

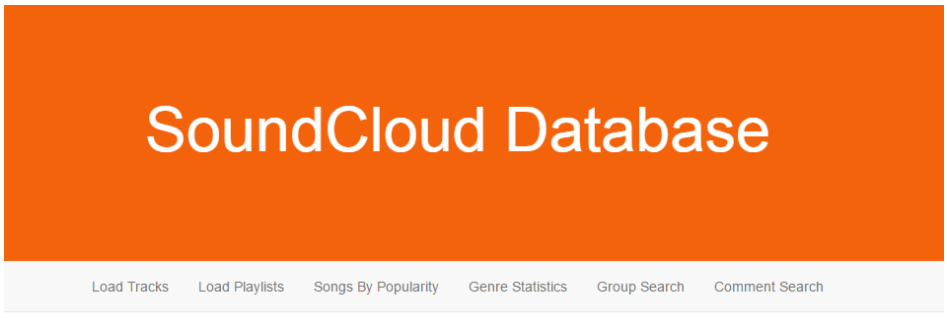


Figure 3

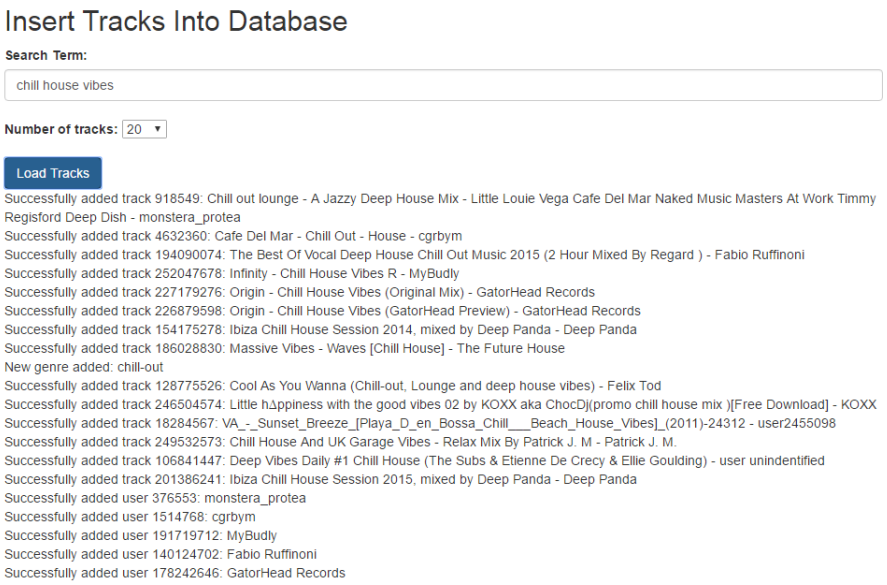
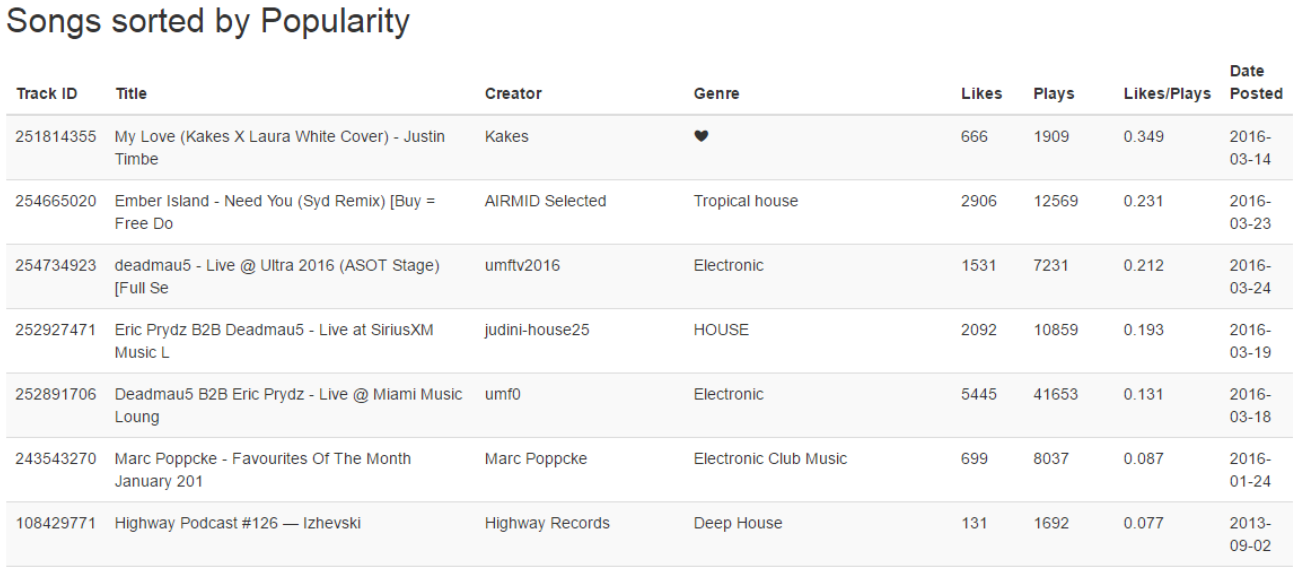


Figure 4

When adding tracks, all comments, new genres, and new users will be added to the database automatically. Similarly, all of the same data is added automatically when adding playlists to the database as well. When users have finished adding all of the data they want, they can then use one of the several search functions on the website to search through the data on their own terms and find what they're looking for faster.



Track ID	Title	Creator	Genre	Likes	Plays	Likes/Plays	Date Posted
251814355	My Love (Kakes X Laura White Cover) - Justin Timbe	Kakes	♥	666	1909	0.349	2016-03-14
254665020	Ember Island - Need You (Syd Remix) [Buy = Free Do	AIRMID Selected	Tropical house	2906	12569	0.231	2016-03-23
254734923	deadmau5 - Live @ Ultra 2016 (ASOT Stage) [Full Se	umftv2016	Electronic	1531	7231	0.212	2016-03-24
252927471	Eric Prydz B2B Deadmau5 - Live at SiriusXM Music L	judini-house25	HOUSE	2092	10859	0.193	2016-03-19
252891706	Deadmau5 B2B Eric Prydz - Live @ Miami Music Loung	umf0	Electronic	5445	41653	0.131	2016-03-18
243543270	Marc Poppcke - Favourites Of The Month January 201	Marc Poppcke	Electronic Club Music	699	8037	0.087	2016-01-24
108429771	Highway Podcast #126 — Izhevski	Highway Records	Deep House	131	1692	0.077	2013-09-02

Figure 5

In this screenshot, Tracks from the database are listed and ordered by their Like to Play ratio, meaning that a larger percentage of listeners favorited the track. This is a really useful way to find good music quickly that isn't offered to you on SoundCloud. All of the tables on the website also include a table sorter for easier reorganization of data once it's collected. There is also a feature allowing users to simply click on the row the track is listed on to display an embed window and listen to it. This allows for

a much smoother browsing experience because the user doesn't have to jump between websites to listen to the tracks they found. If they are logged into SoundCloud on the current system they can even favorite the track and add it to their own playlists.

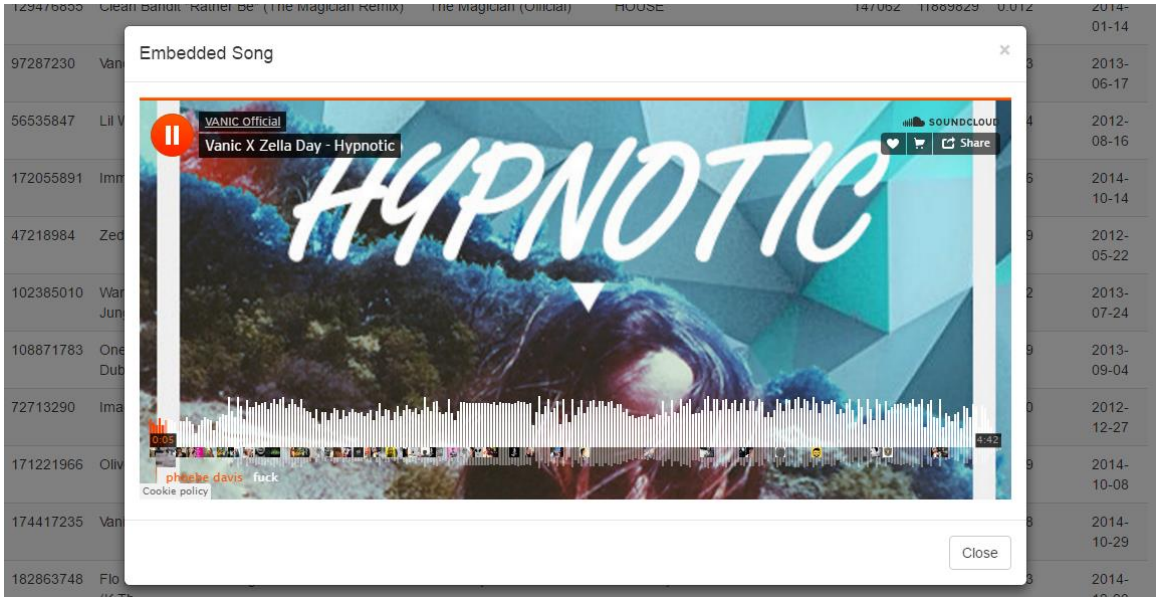


Figure 6

Another Important thing to note is that the database does not include any indexes mainly because the database is small enough to where having an index wouldn't make a noticeable impact on performance speed. Data is also almost always gathered by using the primary key of the table as well, which already includes an index. Because the database is constantly changing as well, it makes it more convenient to not have indexes because they would have to be updated constantly as well and take up additional hard drive storage.

Query Reports

1) SELECT Track.ID AS track_id, Track.title AS title, User.Username AS creator, Genre.Name AS genre, Track.Favorite_count AS likes, Track.Play_count AS plays,(Track.favorite_count/Track.play_count) AS popularity_ratio, Track.Created_at AS date_posted FROM Track INNER JOIN Genre ON Genre.ID = Track.GenreId INNER JOIN User ON User.ID = Track.CreatorId WHERE Track.favorite_count >= 100 AND Track.play_count >= 1000 GROUP BY creator ORDER BY popularity_ratio DESC, likes DESC, plays DESC, date_posted DESC, creator, title

   This Query gives us the output shown in Figure 4 above. It is essential for giving the user search options not available on the original site.
2) SELECT Genre.ID AS genre_id, Genre.Name AS genre, COUNT(*) AS num_of_tracks, (ROUND(SUM(duration.duration)/1000)) AS total_duration FROM Track INNER JOIN Genre ON Genre.ID = Track.GenreId INNER JOIN duration ON duration.trackid = track.id GROUP BY genre HAVING num_of_tracks >= 20 ORDER BY num_of_tracks DESC, genre, duration DESC

This query shows the following:

Genre Statistics

Genre ID	Genre	Number of Tracks	Total Duration of all Tracks (seconds)
2	No Genre	657	755576
14	HOUSE	522	756548
24	Deep House	438	745191
1	Electro House	190	63858
6	Progressive House	179	126580
80	Dance	106	30089
93	Electronic	97	108167
10	Trap	95	23461
52	techno	58	94527
438	Tropical house	58	18883

Figure 7

This lets the user sort by genre statistics including the number of tracks for each genre and the duration of all tracks within the genre.

3) SELECT Track.ID AS track_id, Track.title AS title, User.Username AS creator, Genre.id AS genre, Track.Favorite_count AS likes, Track.Play_count AS plays, Track.Created_at AS date_posted FROM Track INNER JOIN Genre ON Genre.ID = Track.GenreId INNER JOIN User ON User.ID = Track.CreatorId WHERE Track.GenreId IN (SELECT Genre.ID FROM Track INNER JOIN Genre ON Genre.ID = Track.GenreId GROUP BY genre.id HAVING COUNT(track.id) > 20) ORDER BY likes DESC, plays DESC, date_posted DESC, creator, title

This query displays the tracks within the previous query for each genre when clicked on. Ex:

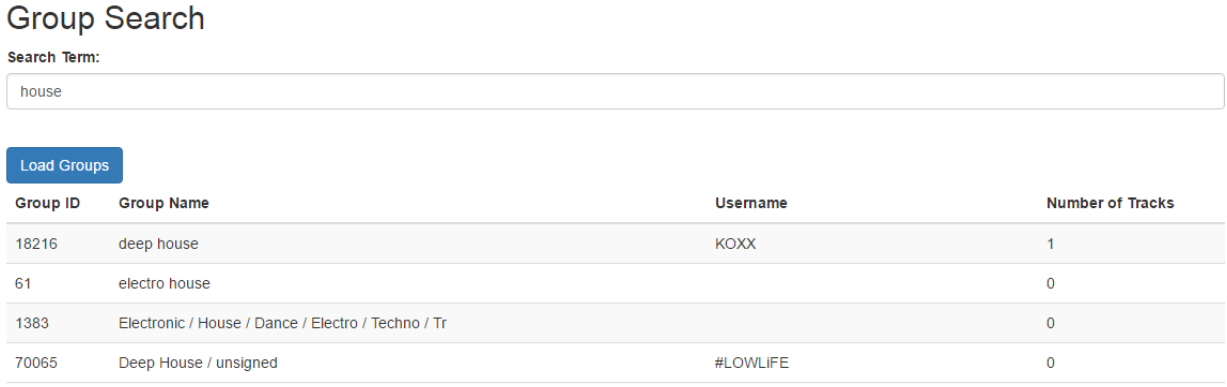
Track ID	Title	Creator	Likes	Plays	Date Posted
114305296	Of Monsters and Men - Little Talks (Thomas Jack Re	Thomas Jack.	117193	7276806	2013-10-07
182274373	Bastille - Flaws (Deep Chills Remix)	Deep Chills	95013	5446433	2014-12-19
168714250	Mattafix - Big City Life (LEEX Remix)	LEEX	78596	4142958	2014-09-21
157948343	Sander Van Doorn & Firebeatz - Guitar Track (Sam F	Sam Feldt	65206	3238360	2014-07-09

Figure 8

This screenshot shows songs within the Tropical house genre and appears after the Tropical House row is clicked from Figure 6.

4) SELECT Groups.ID AS group_id, Groups.Name AS group_name, User.username AS user_name, (SELECT COUNT(*) FROM Track WHERE Track.creatorid = User.ID) AS num_tracks FROM Groups INNER JOIN memberof ON memberof.groupid = Groups.ID INNER JOIN User ON User.ID = memberof.userid WHERE Groups.Name LIKE "%[SearchTerm]%" ORDER BY num_tracks DESC, user_name

This query displays all users within groups matching the search term and shows how many tracks they have uploaded. The user input is also sanitized so that they cannot escape the string and perform an SQL Injection Attack.



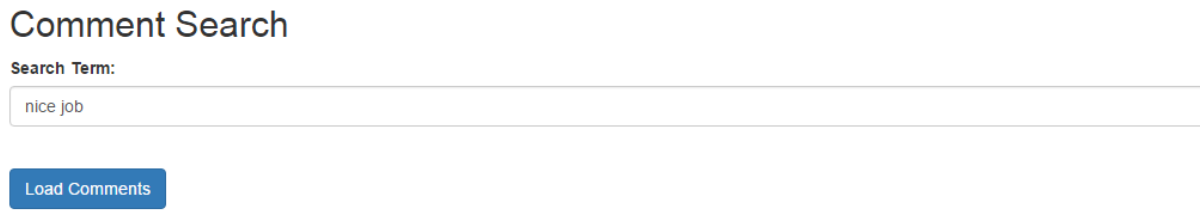
Group ID	Group Name	Username	Number of Tracks
18216	deep house	KOXX	1
61	electro house		0
1383	Electronic / House / Dance / Electro / Techno / Tr		0
70065	Deep House / unsigned	#LOWLiFE	0

Figure 9

In the figure above we see sample output for groups with the search term “house”.

5) SELECT text, CONCAT(User.username, " / ", Track.title) AS title, commentid FROM TrackComment INNER JOIN Track ON Track.ID = TrackComment.TrackId INNER JOIN User ON User.ID = Track.CreatorId WHERE text LIKE "%[SearchTerm]%"

This query is similar to the last one except instead of searching through groups, we are searching through track comments. This user input is sanitized here as well.



Comment ID	Poster / Track	Text
11837391	HouSE / TALKIE FLAK Original mix_electro house	solid electro sond full power indeed , nice job
208620093	Brad Kavanagh / Lego House - Cover of Ed Sheeran Original	nice job
156786402	Brian's House / Brian's House #2: Holiday Edition	Nice job
208619831	Brian's House / Brian's House #5: Dicks In The News / Fan Emails /	nice job
108739418	DJ Eddie One / DJ Eddie One - Don't Stop The Party (Heavy Hitters	Nice Job Bro where is the download link.....

Figure 10

The figure above shows the sample output that includes the comment id, Track title and poster username, and the text of the comment.

Source Code

All source code can be found in the SourceCode folder in the .zip archive. DDL Code is in a text file along with the htdocs folder with all the application code. All DML Code is integrated into the site itself.

User Manual

The only program you will need is XAMPP and a Web Browser (chrome preferrably).

1. Assuming XAMPP is already installed, copy all files within the htdocs folder in the SourceCode folder into your htdocs folder located in C:\xampp\htdocs or wherever XAMPP is installed.
2. Open the XAMPP control panel (C:\xampp\xampp-control.exe) and start Apache and MySQL.
3. Open your web browser and goto localhost/phpmyadmin and create a new database named "soundcloud".
4. Go into the soundcloud database in phpMyAdmin and run the DDL code from the SourceCode folder in the SQL tab to create the tables.
5. Goto localhost/menu.html and use the navbar to load some tracks and playlists into the database.
6. Once you have loaded stuff into the database you need to run php scripts manually to load information for groups. First open localhost/group.php press F12 so you can see the browser console and then click on the "Load Data" button and wait for the console to finish logging events.
7. Repeat the above process at localhost/groupmember.php next and finally localhost/user.php.
8. The database is now ready to be searched! Return to localhost/menu.html and begin searching through data! Feel free to load in more tracks and playlists whenever you feel the need to

Looking Back on the Project

I really enjoyed working with the SoundCloud API in this project as it is one of my personal interests and it made me more motivated while working on the project. The only dislikes I had about the project were the requirements of having a minimum of ten tables and 5 complex queries. I found it more stressful to go out of my way to create a few pointless tables that would be used only minimally in the project just to meet the requirement. I also had to make lots of uneccessary complex queries when there are easier options for working with the database in the PHP and JavaScript code as opposed to html. It was kind of just an inconvenience.

The easiest part of the project was just simply creating the DDL code for the tables and making just a basic html design for the site. The harder parts of the project were definitely creating extra complex queries, which seemed unneccessary just to meet the minimum requirement and initially getting the JavaScript and PHP Code to work with each other because you couldn't rerun PHP code on the same page without reloading it.

From doing this large-scale project I learned a lot about the architecture of big company databases like SoundClouds and I learned how useful having all of that information can be. I really enjoyed using my own application because it made it more convenient for me to find the music that I was looking for because I got to use my own queries as opposed to SoundClouds. There's literally so many ways you can organize your table data it's insane!

Conclusion

To conclude with this project, all of the original goals I made when starting this project were met and the problems I sought out to fix have been solved by my application. I would say that there's nothing this project is missing at this point to be considered complete, however, I will definitely be using this project in the future for my own personal needs and will probably change it up a bit along the way!