# Overview

Our MySQL database contains information about tracks, creditors (performers, writers, producers), and trending dates. We used the Spotify and Genius APIs to gather information into a JSON file which we then parsed using python scripts. The python scripts produced SQL files used to insert the values into the tables we created manually in the CreateTables.sql file. Views and Queries are also created manually each in their own SQL files. The database contains an ISA relationship (performers, writers, and producers ISA creditor) and a weak relationship between trending information and tracks. The trending information is the weak entity.

# Approach and Challenges

## Gathering Data

For the gathering data process, we used three public APIs to gather enough information for each track. Using rapidapi.com, we found 3 APIs that would satisfy our criteria to create the database: Spotify, Spotify Web API, and Billboard. Spotify allowed us to efficiently gather the top 200 songs of each week from now until 2016. To do so, we created a python script that iterates through each week and grabs all the relevant data of the tracks and remove duplicates. The second API, called Spotify Web API was relevant as it provided description about each person who worked on the track (writer, performer and producer). These attributes allowed us to create an ISA relationship. Since this API also used the same trackIDs’ as the first API, creating a bridge between the two was straight forward. Finally, we used Billboard in order to gather additional information about each track, specifically about the trending attributes (appearances on chart, consecutive appearances and play count)

## Creating Tables, Views, and Queries

To create the tables, views, and queries, we simply wrote MySQL scripts. This was straightforward as it is what we’ve been learning about this entire semester. We decided that to simplify things, we would use a local database that is reset every time the CreateTables.sql script is run. Some challenges were related to inserting content into tables, and the data types had to be modified. More complex queries were also a bit of a challenge, but this was easily remedied by online research. In MySQL, domains can not be created, therefore we included CHECKS instead.

## Inserting Data into Tables

To insert the data into the tables, we wrote some python scripts that would insert the appropriate data into each table by reading the package.json file. The scripts would write the appropriate insert statements in SQL files that would be copied and pasted into the PopulateTables.sql file after ensuring their validity. This method brought some challenges, notably ensuring that ‘relationship’ tables such as TrackCreditors contained accurate information, especially since the primary key ids were auto incremented. We tried using a python script to select already present creditors and tracks and find their ids but there were issues with the connection to the database. Therefore, we resorted to duplicating the process and creating scripts (for example insert\_into\_trackCreditor.py) that mimic the process of inserting creditors and tracks and kept in memory the id at which each track was inserted. There were a lot of duplicates, but the INSERT IGNORE statement fixed that issue. Other challenges included replacing empty strings with NULL statements for dates. This was done inside the python script.