Databases and Advanced Data Techniques

Midterm Coursework Report

Weekly Spotify Charts

Link to lab:
https://hub.labs.coursera.org:443/connect/sharedqzrtjder?forceRefresh=false&path=%2F%3Ffolder%3D%2Fhome%2Fcoder%2Fproject&isLabVersioning=true

WebApp demo video: https://youtu.be/buXXAq0NFtI

Introduction

In this project, I applied what I had learnt from the module thus far to evaluate and model a weekly Spotify charts dataset with the aim of identifying and gaining insights into possible factors contributing to the popularity of top tracks on Spotify.

This report documents the processes I had carried out in order to achieve my aim. This includes the process of choosing a suitable dataset by assessing it against several criterias, modelling that dataset using an E/R model, and cleaning and normalising it before implementing it as a MySQL database. Last but not least, the report will document the process of creating a node web application that connects to the MySQL database created, and queries it to display data on the web application.

Table of Contents

Finding and critiquing a dataset

Being an avid music listener who spends majority of my time listening to music, I often find joy in browsing through music charts in hope of discovering new songs and artists. For this reason, I had decided to search for a dataset related to music charts to work on. As I personally use the Spotify platform to stream music, the ***Spotify tracks chart*** dataset caught my attention immediately.

1.1 Dataset Description

The **Spotify tracks chart [1]** dataset consists of data from Spotify's weekly charts across 77 countries, ranging from the year 2014 to 2023.

Within this dataset, alpha-2 country codes are used to determine the country of charting. While looking through the dataset, I realised that many of the country codes were unfamiliar to me. Therefore, I decided to introduce an additional dataset, a **List of all countries with their 2 digit codes [2]**, which is a very small dataset consisting of country codes and their corresponding country names.

Using this additional set of data to derive a new "country name" column in the Spotify dataset would allow easier identification of the countries, especially for people who are not so familiar with alpha-2 country codes.

Scope

As the dataset is very large (5 million rows in total), I will limit the scope to speed up the processing time.

Below are what I will be working with:

- Only charting data ranging from 2020 to 2023.
- Only charting data ranked within the top 100 position.

Sources of Data

1. The first set of data, **Spotify tracks chart**, was downloaded from Kaggle in CSV format. (source: https://www.kaggle.com/datasets/jfreyberg/spotify-chart-data)

2. The second set of data, **List of all countries with their 2 digit codes**, was download from datahub.io in CSV format
(source: https://datahub.io/core/country-list#resource-country-list_zip)

Terms of use

1. The Spotify tracks chart dataset from Kaggle is licensed under **Attribution 4.0 International (CC BY 4.0) [3]**, which allows use of the data for any purpose as long as the appropriate credits are given to the creator.

Additionally, according to the dataset description on Kaggle, the data was initially taken from Kworb **[4]**, a website that displays music data from several music streaming platforms. Referring to the FAQ page **[5]**, the owner states that free use of data displayed on the website is allowed.

2. As for the additional dataset, it is stated on the “Read me” section that data on the website is “licensed by its maintainers under the Public Domain Dedication and License.”(Datopian)**[6]** allowing free use of the data.

1.2 Dataset Evaluation

In this section, I will evaluate the Spotify dataset based on its quality, level of detail, documentation, interrelation, use, and discoverability.

Quality

The Spotify dataset consists of data that had initially been provided by Spotify through the Spotify API. Spotify is one of the most well-known music streaming platforms with users all over the world, guaranteeing a certain level of accuracy and quality in the data made available to the public.

Based on the “column” tab of the Kaggle page of the dataset, out of all ten columns only the “name” column had 885 missing values. Furthermore, values in all columns were more or less consistent in format. Therefore, this dataset is suitable to be used in terms of quality.

Level of detail

The Spotify dataset contains a high level of detail for example, the date column which represents the date which the song entered the charts not only contains the year and month, it even contains the specific day where the track charted. Also, each track lists down every artist featured in each track, instead of just the main artist. This level of detail allows me to ask more advanced questions to find out even more.

Documentation

Although a description was included for each column of the dataset, it was obvious that it was mismatched. For example, the “date” column has the description “Spotify track id” while the “track_id” column has the description “Position this track appeared in the countries' charts”. Fortunately, the column names were clear in what the values in each column was, and the values stored in each column also matched the column names.

Interrelation

As I had discussed previously, a new dataset was introduced to add a “country name” column to supplement the existing column containing country codes, for the purpose of allowing easier identification of countries.

Use

There are many use cases that relate to the Spotify dataset. For example, it can be used to analyse music trends across different countries, and also measure the rise or drop in popularity of a particular artist.

Discoverability

The Spotify dataset and datasets of a similar type are easily discoverable due to the popularity of the Spotify platform, as well as the artists whose songs are on the platform.

1.3 Areas of Interest

Below are the questions I would like to ask regarding the dataset I had chosen.

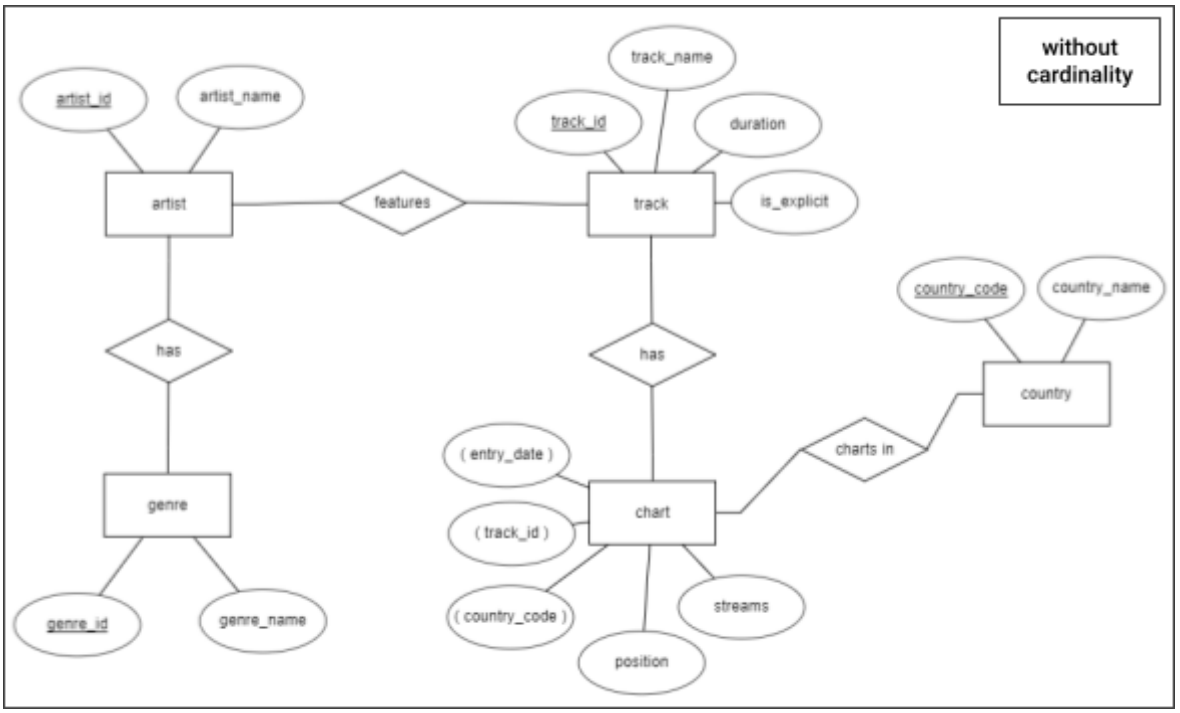
- Which tracks topped the charts the most number of times, and who are the artists featured on those tracks?
- What are the genres of the artists with the highest number of total streams in the first quarter of 2021?
- Between explicit and non-explicit tracks, which one generally has more streams? Is this consistent throughout all countries?

Since the questions above are not very straightforward, it would require more than just basic sorting and filtering. Hence, using a database would be much more efficient as opposed to a spreadsheet for example. Furthermore, using a database to store the dataset would allow modification of all records (i.e. updating artist name whenever an artist changes their stage name) with just a simple query such as: `UPDATE spotifyDB SET artist_name = new_name_value WHERE artist = artistA`.

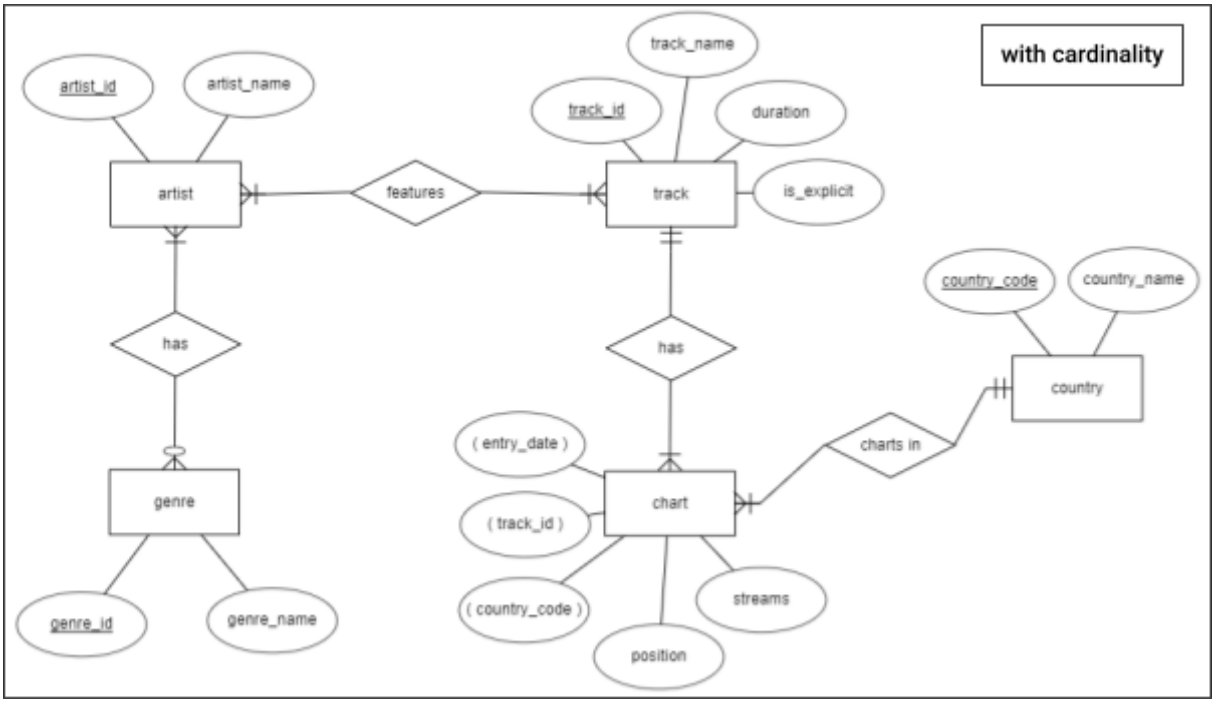
Modelling the Data

2.1 Complete E/R Model

Below shows the Entity/Relationship model of the Spotify dataset with an additional "country_name" attribute from the additional country dataset mentioned previously. Underlined attributes are primary keys, and those in brackets are composite keys.

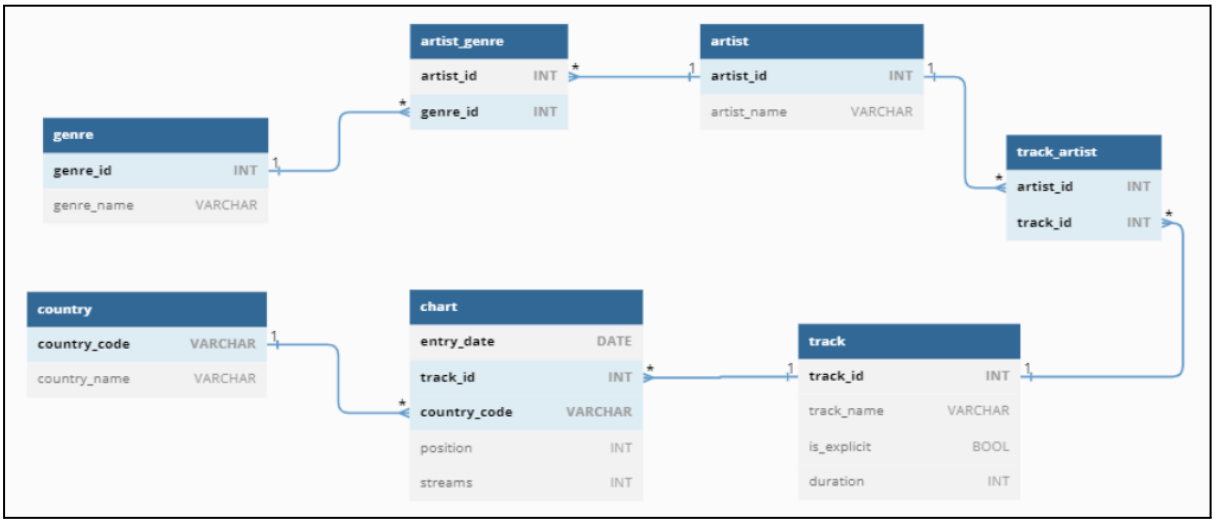


Each track features one or many artists, and each artist is featured on one or many tracks. On the other hand, each artist can be associated with zero or many genres, and each genre is associated with one or many artists.



2.2 Relational Schema

Below shows the relational schema with 7 tables in total, two of which are junction tables (artist_genre, track_artist). Attributes in bold are the primary keys.



2.3 Database tables and fields

Below are the list of columns from the original datasets, along with a short description of each column.

Dataset	Columns	Description
Spotify tracks chart	date	The date of charting.
	country	Alpha-2 country code of the country of charting.
	position	Charting position.
	streams	Number of streams as of charting date.
	artists	Name of the artists featured on the track.
	artist_genres	Genres of the artists.
	track_id	Unique ID of the track on spotify.
	name	Name of the track.
	duration	Duration of the track (in milliseconds).
	explicit	whether the track contains explicit content.
List of all countries	Code	Alpha-2 country code.
	Name	Country name.

2.4 Data Cleaning

In order to prepare the dataset for carrying out normalisation, data cleaning had to be carried out to handle missing values and outliers.

Using the Pandas library in Python, the following issues were handled:

- Dropping data that are not within my scope.
- Handle missing track names.
- Converting dates to the same format.

The full cleaning process can be found in section 6.1 of the appendix.

2.5 Normalisation process

In this section, the dataset will be evaluated against all the normal forms to achieve the highest normal form possible. Diagrams will be used to illustrate the changes as the table progresses through all the normal forms.

Normalisation of the actual csv data was carried out in a jupyter notebook using Python libraries. The full notebook can be found in section 6.2 of the appendix.

Below shows the table that will be normalised.

1st Normal Form (1NF)

To satisfy the first normal form, there should be no duplicated rows and no missing data. Each cell should only contain one value, and there should be a unique column(s) that identifies each row of data.

My current table does not satisfy 1NF as:

- The “artists” and “artist_genres” contain cells storing more than one value.
- Rows cannot be uniquely identified.

Satisfying 1NF

To satisfy 1NF, the cells containing more than one value are split up into their own row of records. Next, the “entry_date”, “track_id”, and “country_code” are set as the composite key to uniquely identify each row of data.

Below shows the table that now satisfies 1NF.

Charts	
entry_date	DATE
track_id	VARCHAR
country_code	VARCHAR
country_name	VARCHAR
position	INT
streams	INT
artist_names	VARCHAR
artist_genres	VARCHAR
duration	INT
explicit	BOOL
track_name	VARCHAR

2nd Normal Form (2NF)

To satisfy the second normal form, the table should satisfy 1NF, and there should be no partial dependencies. Attributes of the table should be dependent on all key values. The key values in my current table are **entry_date**, **track_id**, and **country_code**.

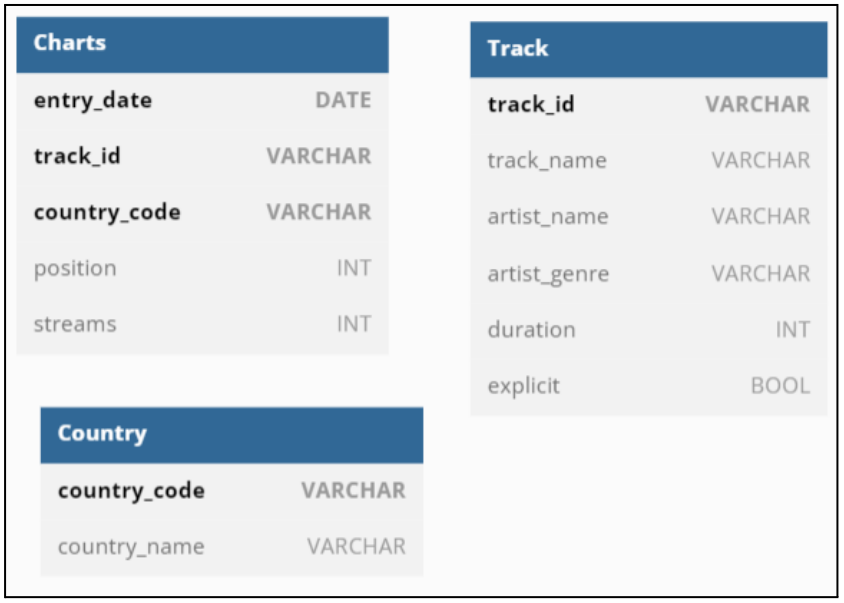
My current table does not satisfy 2NF as:

- *track_name*, *duration*, *explicit*, *artists*, *artist_genres* are dependent on *track_id* but not the other two key values.
- *country_name* is dependant on *country_code* but not the two other keys

Satisfying 2NF

To satisfy 2NF, two new tables “Track” and “Country” should be created. Values dependent on track_id are stored in the Track table, while values dependent on country_code are stored in the county table. This removes the partial dependencies within the chart table.

Below shows the tables that now satisfy 2NF.



3rd Normal Form (3NF)

To satisfy 3NF, the tables should satisfy 2NF, and should not contain transitive dependencies where a non-key value depends on another non-key value.

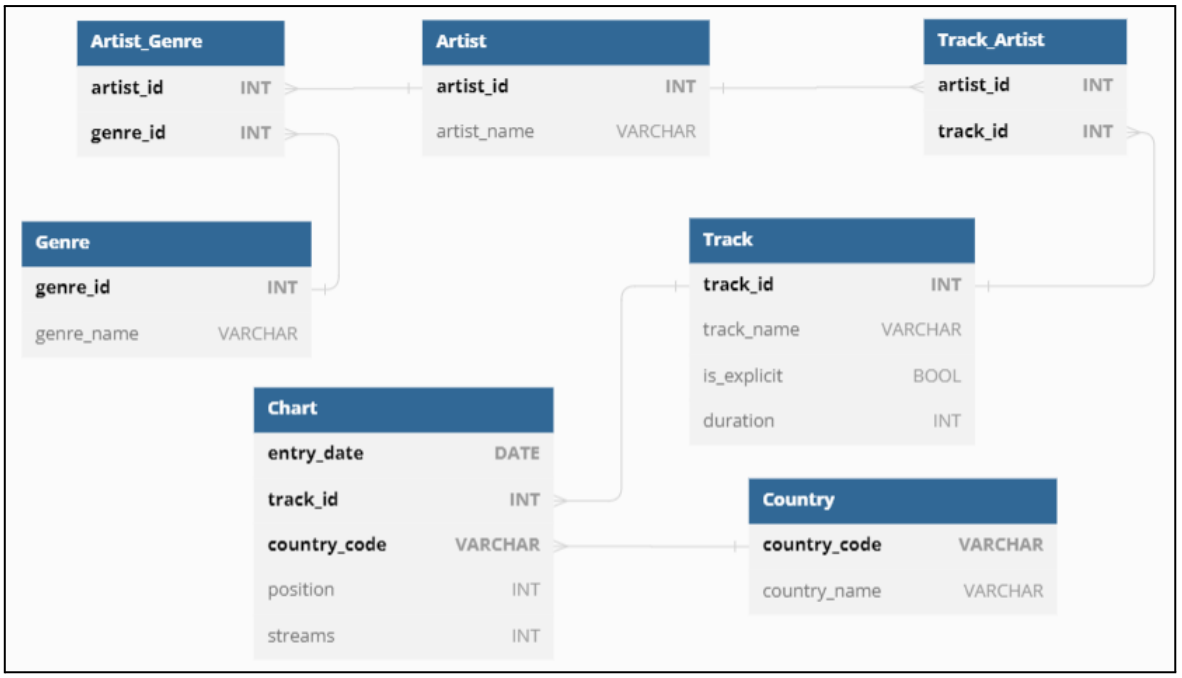
My current tables do no satisfy 3NF as:

- The non-key attribute “artist_genre” in the Track table is dependent on another non-key attribute “artist_name”.
- Many-to-many relationship between track_id and artist_name.
- Many-to-many relationship between artist_name and artist_genres.

Satisfying 3NF

To satisfy 3NF, a “Genre” table should be created to store “artist_genre” and should reference the “artist_name”. Next, the many-to-many relationships can be resolved by creating a junction table.

Below shows the tables that now satisfy 3NF.



Boyce-Codd Normal Form (BCNF)

In the current tables, only the junction tables containing a candidate key could have possible violation of BCNF. For both junction tables “Artist_Genre” and “Track_Artist”, they contain the key-attribute of the left and right table (e.g. artist_id and genre_id are both key attributes of Artist and Genre table respectively). This proves that BCNF has already been satisfied.

Database Creation

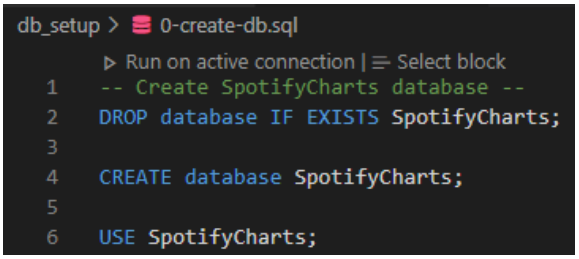
This section will go through the process of how I had created a MySQL database in the lab environment that stores data from the dataset.

To allow reusability, I had written all the SQL statements that are needed, into sql scripts before running those scripts to create the database structure. Below are all the scripts and each of its purposes.

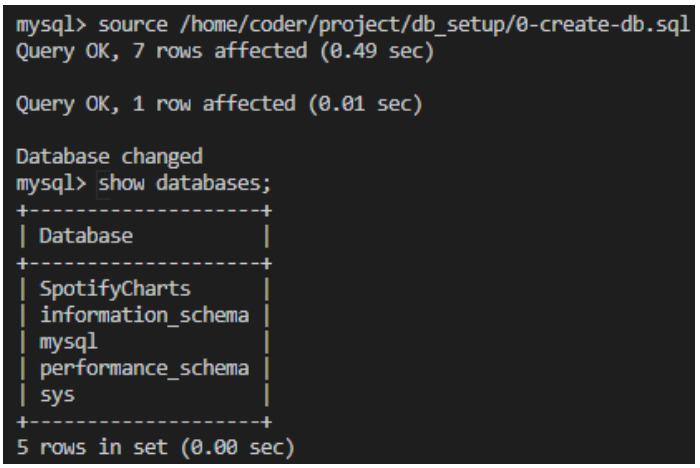
- **0-create-db.sql** : Creates a new SpotifyCharts database.
- **1-create-tables.sql** : Creates all tables needed, with relevant fields.
- **2-ingest-data.sql** : Loads data from CSV files into the respective db tables.
- **3-create-user.sql** : Creates a new user and grants SELECT privileges on all tables to the user.

3.1 Building the database structure

To create the database, I ran the "*0-create-db.sql*" script using the **source** command. This script contains commands that first drops any existing database called "SpotifyCharts". Afterwards, it creates a new "SpotifyCharts" database using a CREATE statement.



(Contents of the sql script "*0-create-db.sql*")



(Running the script to create new database)

Creating the tables

To create the database tables, I ran the "*1-create-tables.sql*" script which contains CREATE TABLE commands to create the tables: artist, genre, artist_genre, track, track_artist, country, chart. All CREATE commands used are in the next page.

```
mysql> source /home/coder/project/db_setup/1-create-tables.sql
Query OK, 0 rows affected, 1 warning (0.00 sec)

Query OK, 0 rows affected, 1 warning (0.00 sec)

Query OK, 0 rows affected, 1 warning (0.00 sec)

Query OK, 0 rows affected, 1 warning (0.00 sec)

Query OK, 0 rows affected, 1 warning (0.00 sec)

Query OK, 0 rows affected, 1 warning (0.00 sec)

Query OK, 0 rows affected, 1 warning (0.00 sec)

Query OK, 0 rows affected (0.15 sec)

Query OK, 0 rows affected (0.13 sec)

Query OK, 0 rows affected (0.17 sec)

Query OK, 0 rows affected (0.12 sec)

Query OK, 0 rows affected (0.15 sec)

Query OK, 0 rows affected (0.14 sec)

Query OK, 0 rows affected (0.20 sec)

+-------------------------+
| Tables_in_SpotifyCharts |
+-------------------------+
| artist                  |
| artist_genre            |
| chart                   |
| country                 |
| genre                   |
| track                   |
| track_artist            |
+-------------------------+
7 rows in set (0.00 sec)
```

(Running the script to create tables)

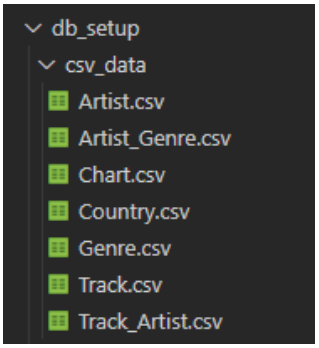
All CREATE commands used

Command	Description
`CREATE DATABASE SpotifyCharts;`	Creates “SpotifyCharts” database.
`CREATE TABLE artist (` `  artist_id INT PRIMARY KEY,` `  artist_name VARCHAR(100)` `);`	Creates an “artist” table with the attributes *artist_id* and *artist_name*. *artist_id* is also set at the primary key.
`CREATE TABLE genre (` `  genre_id INT PRIMARY KEY,` `  genre_name VARCHAR(100)` `);`	Creates a “genre” table with the attributes *genre_id* and *genre_name*. *genre_id* is also set at the primary key.
`CREATE TABLE artist_genre (` `  artist_id INT,` `  genre_id INT,` `  PRIMARY KEY (artist_id, genre_id),` `  FOREIGN KEY (artist_id)` `  REFERENCES artist(artist_id),` `  FOREIGN KEY (genre_id)` `  REFERENCES genre(genre_id)` `);`	Creates an “artist_genre” table with the attributes *artist_id* and *genre_id* that references *artist_id* from “artist” table, and *genre_id* from “genre” table respectively. Both attributes make up the composite key. This table is a junction table connecting the tables “artist” and “genre”.
`CREATE TABLE track (` `  track_id VARCHAR(62) PRIMARY KEY,` `  track_name VARCHAR(255),` `  duration INT,` `  is_explicit BOOLEAN` `);`	Creates a “track” table with the attributes *track_id*, *track_name*, *duration*, and *is_explicit*. *track_id* is also set as the primary key. *track_id* is of type VARCHAR(62) as it contains 62 character spotify track IDs.
`CREATE TABLE track_artist (` `  track_id VARCHAR(62),` `  artist_id INT,` `  PRIMARY KEY (track_id, artist_id),` `  FOREIGN KEY (track_id)` `  REFERENCES track(track_id),` `  FOREIGN KEY (artist_id)` `  REFERENCES artist(artist_id)` `);`	Creates a “track_artist” table with the attributes *track_id* and *artist_id* that references *track_id* from “track” table, and *artist_id* from “artist” table respectively. Both attributes make up the composite key. This table is a junction table connecting the tables “track” and “artist”.
`CREATE TABLE country (` `  country_code CHAR(2) PRIMARY KEY,` `  country_name VARCHAR(255)` `);`	Creates a “country” table with the attributes *country_code* and *country_name*. *country_code* is also set at the primary key.

`` CREATE TABLE chart (entry_date DATE, track_id VARCHAR(62), country_code CHAR(2), position INT, streams INT, PRIMARY KEY (entry_date, track_id,country_code), FOREIGN KEY (track_id) REFERENCES track (track_id), FOREIGN KEY (country_code) REFERENCES country (country_code)); ```	Creates a “chart” table with the attributes *entry_date, track_id, country_code, position, streams.* It references *track_id* from “track” table, and *country_code* from ”country” table. *entry_date*, *track_id*, and *country_code* make up the composite key.

3.2 Loading data into the database

After setting up the database structure, I added the csv files containing the cleaned data into the lab environment, under the “csv_data” folder as seen below.



Afterwards, I ran the “*2-ingest-data.sql*” script containing LOAD DATA statements that loads data from each of the csv files into the corresponding database table.

(snippet of content in sql script “*2-ingest-data.sql*”)

```
mysql> source /home/coder/project/db_setup/2-ingest-data.sql
Query OK, 15239 rows affected (1.28 sec)
Records: 15239  Deleted: 0  Skipped: 0  Warnings: 0

Query OK, 2099 rows affected (0.13 sec)
Records: 2099  Deleted: 0  Skipped: 0  Warnings: 0

Query OK, 69746 rows affected (2.82 sec)
Records: 69746  Deleted: 0  Skipped: 0  Warnings: 0

Query OK, 38556 rows affected (1.63 sec)
Records: 38556  Deleted: 0  Skipped: 0  Warnings: 0

Query OK, 62760 rows affected (3.22 sec)
Records: 62760  Deleted: 0  Skipped: 0  Warnings: 0

Query OK, 76 rows affected (0.02 sec)
Records: 76  Deleted: 0  Skipped: 0  Warnings: 0

Query OK, 1077101 rows affected (1 min 4.08 sec)
Records: 1077101  Deleted: 0  Skipped: 0  Warnings: 0
```

(running the script to load csv data into tables)

I was able to verify that all rows of data had been loaded correctly by comparing the number of rows affected with the actual number of rows in the csv file (excluding the header row in the csv).

Querying the database to display all columns from all tables using the appropriate join statements also returned the expected output.

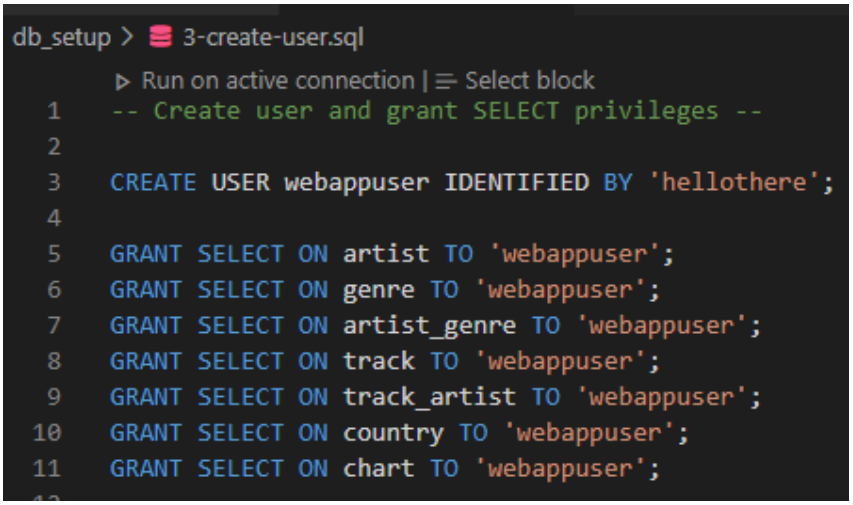
```
mysql> SELECT chart.entry_date, chart.position,
    ->     country.country_code, country.country_name,
    ->     track.track_id, track.track_name,
    ->     artist.artist_id, artist.artist_name,
    ->     genre.genre_id, genre.genre_name,
    ->     chart.streams, track.duration, track.is_explicit
    -> FROM chart
    ->     JOIN country ON chart.country_code = country.country_code
    ->     JOIN track ON chart.track_id = track.track_id
    ->     JOIN track_artist ON track.track_id = track_artist.track_id
    ->     JOIN artist ON track_artist.artist_id = artist.artist_id
    ->     JOIN artist_genre ON artist.artist_id = artist_genre.artist_id
    ->     JOIN genre ON artist_genre.genre_id = genre.genre_id
    -> LIMIT 10;
+------------+----------+--------------+--------------+------------------------+-----------------+-----------+-------------+----------+-------------------------+---------+----------+-------------+
| entry_date | position | country_code | country_name | track_id               | track_name      | artist_id | artist_name | genre_id | genre_name              | streams | duration | is_explicit |
+------------+----------+--------------+--------------+------------------------+-----------------+-----------+-------------+----------+-------------------------+---------+----------+-------------+
| 2020-09-10 |       29 | ad           | Andorra      | 06s3QtMJVXw1AJX3UfvZG1 | Hasta Que Dios Diga |       104 | Anuel AA    |        1 | pop                     |    1377 |   246000 |           1 |
| 2020-09-10 |       29 | ad           | Andorra      | 06s3QtMJVXw1AJX3UfvZG1 | Hasta Que Dios Diga |       104 | Anuel AA    |        3 | pop rap                 |    1377 |   246000 |           1 |
| 2020-09-10 |       29 | ad           | Andorra      | 06s3QtMJVXw1AJX3UfvZG1 | Hasta Que Dios Diga |       104 | Anuel AA    |        4 | melodic rap             |    1377 |   246000 |           1 |
| 2020-09-10 |       29 | ad           | Andorra      | 06s3QtMJVXw1AJX3UfvZG1 | Hasta Que Dios Diga |       104 | Anuel AA    |       10 | dance pop               |    1377 |   246000 |           1 |
| 2020-09-10 |       29 | ad           | Andorra      | 06s3QtMJVXw1AJX3UfvZG1 | Hasta Que Dios Diga |       104 | Anuel AA    |       13 | rap                     |    1377 |   246000 |           1 |
| 2020-09-10 |       29 | ad           | Andorra      | 06s3QtMJVXw1AJX3UfvZG1 | Hasta Que Dios Diga |       104 | Anuel AA    |       17 | north carolina hip hop  |    1377 |   246000 |           1 |
| 2020-09-10 |       29 | ad           | Andorra      | 06s3QtMJVXw1AJX3UfvZG1 | Hasta Que Dios Diga |       104 | Anuel AA    |       18 | trap                    |    1377 |   246000 |           1 |
| 2020-09-10 |       29 | ad           | Andorra      | 06s3QtMJVXw1AJX3UfvZG1 | Hasta Que Dios Diga |       104 | Anuel AA    |       25 | chicago rap             |    1377 |   246000 |           1 |
| 2020-09-10 |       29 | ad           | Andorra      | 06s3QtMJVXw1AJX3UfvZG1 | Hasta Que Dios Diga |       104 | Anuel AA    |       28 | r&b en espanol          |    1377 |   246000 |           1 |
| 2020-09-10 |       29 | ad           | Andorra      | 06s3QtMJVXw1AJX3UfvZG1 | Hasta Que Dios Diga |       104 | Anuel AA    |       38 | miami hip hop           |    1377 |   246000 |           1 |
+------------+----------+--------------+--------------+------------------------+-----------------+-----------+-------------+----------+-------------------------+---------+----------+-------------+
10 rows in set (0.00 sec)
```

(querying the database to show all columns from all tables)

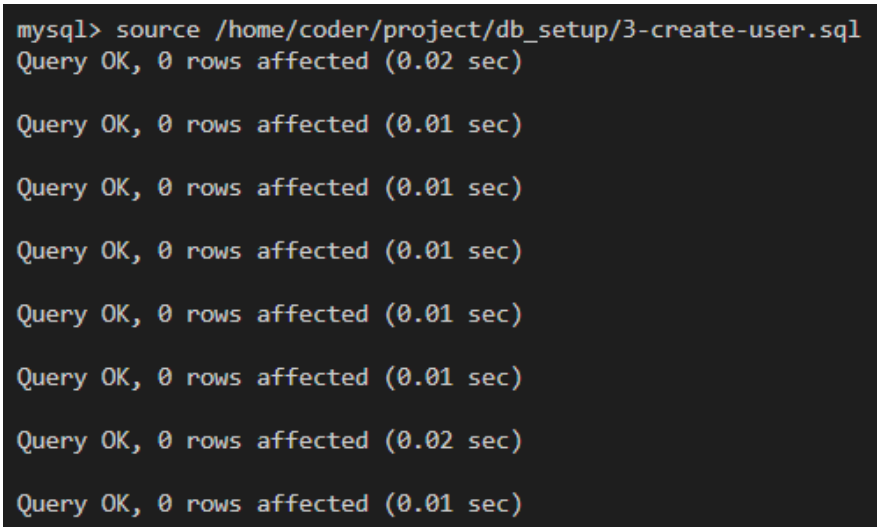
3.3 Creating user and granting privileges

Last but not least, a user has to be created with the appropriate privileges so that it can be used to connect to the database in the node web app later on. Since the web app would only query from the database to display data, the user would only require SELECT privileges.

To create the user account, I ran the "*3-create-user.sql*" script which contains a CREATE USER statement that creates a new user, and grants SELECT on all tables to the user.



(content of sql script "*3-create-user.sql*")



(running the script to create user and grant privileges)

To verify if the user account was created successfully, I connected to mysql using the details of the newly created user and displayed the grants for the user.

```
coder@3b7594b4f7c6:~/project$ mysql -u webappuser -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 11
Server version: 8.0.22 MySQL Community Server - GPL

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.
```

(successful login to new user account)

```
mysql> show grants;
+-----------------------------------------------------------------------------+
| Grants for webappuser@%                                                     |
+-----------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO `webappuser`@`%`                                      |
| GRANT SELECT ON `SpotifyCharts`.`artist_genre` TO `webappuser`@`%`          |
| GRANT SELECT ON `SpotifyCharts`.`artist` TO `webappuser`@`%`                |
| GRANT SELECT ON `SpotifyCharts`.`chart` TO `webappuser`@`%`                 |
| GRANT SELECT ON `SpotifyCharts`.`country` TO `webappuser`@`%`               |
| GRANT SELECT ON `SpotifyCharts`.`genre` TO `webappuser`@`%`                 |
| GRANT SELECT ON `SpotifyCharts`.`track_artist` TO `webappuser`@`%`          |
| GRANT SELECT ON `SpotifyCharts`.`track` TO `webappuser`@`%`                 |
+-----------------------------------------------------------------------------+
```

(list of privileges)

Using an INSERT statement on the track table indeed gave an error telling me that I do not have the required privilege.

```
mysql> USE SpotifyCharts;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> INSERT INTO track VALUES (231ghjg3j, "new track");
ERROR 1142 (42000): INSERT command denied to user 'webappuser'@'localhost' for table 'track'
```

3.4 Reflection

Overall, the database reflects most of the data well.

For example, the “entry_date” column stores the date where the track entered the charts. Since the values are stored in DATE format, it allows retrieval of the YEAR, MONTH and DAY easily as demonstrated below.

```
mysql> SELECT *
    -> FROM chart
    -> WHERE YEAR(entry_date) = 2023
    -> AND MONTH(entry_date) < 02
    -> AND DAY(entry_date) < 10
    -> LIMIT 10;
+------------+------------------------+--------------+----------+---------+
| entry_date | track_id               | country_code | position | streams |
+------------+------------------------+--------------+----------+---------+
| 2023-01-05 | 00TO3hVgOAgfKrRjrKEZxx | eg           |       33 |   82864 |
| 2023-01-05 | 00xIHj5ZaQ2jLAa5eXaOhy | by           |       34 |   28457 |
| 2023-01-05 | 00xIHj5ZaQ2jLAa5eXaOhy | kz           |       86 |   25536 |
| 2023-01-05 | 01ovvmz6UHFeuCh6pLcBe4 | kz           |       37 |   38185 |
| 2023-01-05 | 03qkDom9tLmAt1e8tBijkK | ve           |       92 |   26214 |
| 2023-01-05 | 03yKEFruN3BS2coglBtt2N | in           |       75 | 1508763 |
| 2023-01-05 | 04TshWXkhV1qkqHzf31Hn6 | jp           |       94 |  375329 |
| 2023-01-05 | 04TshWXkhV1qkqHzf31Hn6 | tw           |       92 |   87894 |
| 2023-01-05 | 05sqcYfU2wMlKwPVJ0rotq | pk           |       71 |   43355 |
| 2023-01-05 | 06IeMRkIhGUJY5kvhvYiT9 | sk           |       44 |   28668 |
+------------+------------------------+--------------+----------+---------+
```

On the other hand, an element not reflected well is the “is_explicit” column storing BOOLEAN values indicating whether a track contains explicit content. MySQL stores boolean types as tinyint (e.g. 1 or 0) as seen in the screenshot below. Since this database would be queried to display content on a web application, displaying 0s and 1s might confuse users without programming knowledge. Fortunately, using a CASE statement to return “Yes/No” instead of “1/0” would easily solve this issue.

```
mysql> SELECT * FROM track LIMIT 15;
+------------------------+-------------------------------------+----------+-------------+
| track_id               | track_name                          | duration | is_explicit |
+------------------------+-------------------------------------+----------+-------------+
| 001b8t3bYPfnabpjpfG1Y4 | Geen Stof                           |   167865 |           1 |
| 002wiS64p2qXaYYVAP7KOW | Set Myself On Fire                  |   157986 |           0 |
| 003VDDA7J3Xb2ZFlNx7nIZ | YELL OH                             |   236778 |           1 |
| 003vvx7Niy0yvhvHt4a68B | Mr. Brightside                      |   222973 |           0 |
| 0041QJFD2EWfKWdAZpX2RB | Diaspora Night                      |   142875 |           0 |
| 004nEmGTeP5Mus4qXxz3x2 | Phoyisa - Live                      |   356414 |           1 |
| 004zGvrc84enQHFitC6auy | Mercy On Me (In Sbatti)             |   184186 |           1 |
| 006u4CXVO2cusq08RrME1Q | 10月無口な君を忘れる                  |   332285 |           0 |
| 0074XGa4ytWx2s1o48jO3y | Boîte à Gants                       |   167960 |           1 |
| 0078peqORcMMXbcYEfeLkk | Vet Hon Om?                         |   184773 |           0 |
| 007VTHj6roW99wKcMbUsBR | Egyptian Cotton                     |   362339 |           1 |
| 0087fDVeNhuyMtST91RMjv | Kundiman                            |   339053 |           0 |
| 008A8MMPCshycPlNewvCPn | Top                                 |   161926 |           1 |
| 008aqmngiiKe5jrPSNyV6n | G Wagon                             |   209680 |           0 |
| 008hCrBLW4a7BbC6c6UUBL | GOBLIN (Feat. Homies) (Prod. R.Tee) |   239543 |           0 |
+------------------------+-------------------------------------+----------+-------------+
```

3.5 Answering the questions

Q1. Which tracks topped the charts the most number of times, and who are the artists featured on those tracks?

The first half of the question can be answered using the query below. It does a count of the number of rows with a position of 1, with track_id as the basis of the count. The result is then sorted from highest to lowest count.

```
SELECT
    track.track_name AS "Track Name",
    COUNT(chart.track_id) AS "Number of times"
FROM chart
JOIN track ON chart.track_id = track.track_id
WHERE chart.position = 1
GROUP BY chart.track_id
ORDER BY 2 DESC;
```

The second part of the question requires retrieving artists featured on each track. As a track can feature one or more artists, the GROUP_CONCAT function has to be used to concatenate the names of all artists featured on a track into a single string if that track contains more than one artist. Below shows the full SQL query that answers question 1, as well as an example output.

```
SELECT
    track.track_name AS "Track Name",
    COUNT(chart.track_id) AS "Number of times",
    GROUP_CONCAT(DISTINCT artist.artist_name) AS "Artists"
FROM chart
JOIN track ON chart.track_id = track.track_id
JOIN track_artist ON track.track_id = track_artist.track_id
JOIN artist ON track_artist.artist_id = artist.artist_id
WHERE chart.position = 1
GROUP BY chart.track_id
ORDER BY 2 DESC;
```

Track Name	Number of times	Artists
STAY (with Justin Bieber)	550	Justin Bieber,The Kid LAROI
Lo Siento BB:/ (with Bad Bunny & Julieta Venegas)	471	Bad Bunny,Julieta Venegas,Tainy
DÁKITI	372	Bad Bunny,Jhay Cortez
Tusa	298	KAROL G,Nicki Minaj
Mood (feat. iann dior)	292	24kGoldn,iann dior
ROCKSTAR (feat. Roddy Ricch)	278	DaBaby,Roddy Ricch
Fiel	249	Jhayco,Los Legendarios,Wisin
Quevedo: Bzrp Music Sessions, Vol. 52	248	Bizarrap,Quevedo
La Jeepeta - Remix	240	Anuel AA,Brray,Juanka,Myke Towers,Nio Garcia

Q2. What are the genres of the artists with the highest number of total streams in the first quarter of 2021?

The second part of the question can be answered by the query below. The sum of all stream counts are added up for each artist, but only the streams from rows where the entry_date is within the first 4 months of the year 2021. This returns the total stream count of each artist for the first quarter of the year 2021.

```
SELECT
    artist.artist_name AS "Artist Name",
    SUM(chart.streams) AS "Total streams"
FROM artist
JOIN track_artist ON artist.artist_id = track_artist.artist_id
JOIN track ON track_artist.track_id = track.track_id
JOIN chart ON track.track_id = chart.track_id
WHERE YEAR(chart.entry_date) = 2021
AND MONTH(chart.entry_date) < 5
GROUP BY artist.artist_id
ORDER BY 2 DESC;
```

As each artist can have more than one genre, the GROUP_CONCAT function has to be used to concatenate the genres associated with each artist into a single string if there is more than one. Below shows the full SQL query that answers question 2, as well as an example output produced by the query.

```
SELECT
    artist.artist_name AS "Artist name",
    SUM(chart.streams) AS "Total streams",
    GROUP_CONCAT(DISTINCT genre.genre_name) AS "Artist genres"
FROM artist
JOIN artist_genre ON artist.artist_id = artist_genre.artist_id
JOIN genre ON artist_genre.genre_id = genre.genre_id
JOIN track_artist ON artist.artist_id = track_artist.artist_id
JOIN track ON track_artist.track_id = track.track_id
JOIN chart ON track.track_id = chart.track_id
WHERE YEAR(chart.entry_date) = 2021
AND MONTH(chart.entry_date) < 5
GROUP BY artist.artist_id
ORDER BY 2 DESC;
```

Artist name	Total streams	Artist genres
Bad Bunny	64889202168	afro dancehall,afropop,alternative hip hop,alt...
Justin Bieber	61771832626	afro dancehall,afro r&b,afropop,alabama rap,...
Myke Towers	50306598567	argentine hip hop,bachata,basshall,colombian...
J Balvin	39450198028	afro dancehall,afropop,argentine hip hop,bas...
Rauw Alejandro	32301097752	argentine hip hop,basshall,colombian hip hop,...
The Weeknd	32112388698	art pop,atl hip hop,atl trap,canadian contemp...

Q3: Between explicit and non-explicit tracks, which one generally has more streams? Is this consistent throughout all countries?

This question can be solved using the query shown below. With the country_code set as the basis, for every country, the total number of streams is calculated for explicit tracks and non-explicit tracks separately. The CONCAT function is then used to display both sums within a single string.

```
SELECT
    country.country_name AS "Country",
    CONCAT(
      'Explicit: ',
        SUM(CASE WHEN track.is_explicit = 1 THEN chart.streams ELSE 0 END),
      ' || ',
      'Non-Explicit: ',
        SUM(CASE WHEN track.is_explicit = 0 THEN chart.streams ELSE 0 END)
    ) AS "Streams"
FROM track
JOIN chart ON track.track_id = chart.track_id
JOIN country ON chart.country_code = country.country_code
GROUP BY country.country_code;
```

Below shows the output returned by the query.

Country	Streams
Netherlands	Explicit: 2473039180 || Non-Explicit: 5970536229
Korea, Republic of	Explicit: 46930119 || Non-Explicit: 326182531
United States	Explicit: 33132119196 || Non-Explicit: 23911748874
Australia	Explicit: 3456205712 || Non-Explicit: 5146510051
United Kingdom	Explicit: 6477979274 || Non-Explicit: 8466459314
Ireland	Explicit: 629970774 || Non-Explicit: 1207829676
New Zealand	Explicit: 621436812 || Non-Explicit: 1019452622
Canada	Explicit: 4322973246 || Non-Explicit: 4166792282
Austria	Explicit: 510896839 || Non-Explicit: 835609670
Belgium	Explicit: 764596667 || Non-Explicit: 1347304644
Luxembourg	Explicit: 50567816 || Non-Explicit: 67923945

Creating a web application

4.1 Presenting the web application

A simple node.js web application was created to display data queried from the database that was created earlier. EJS was used as the templating engine together with the Express framework. Additionally, the mysql2 module was used to connect to the database so that it can be queried to retrieve data to display on the html page.

Below shows the code used to create a connection to the database using the "*webappuser*" account that was created earlier.

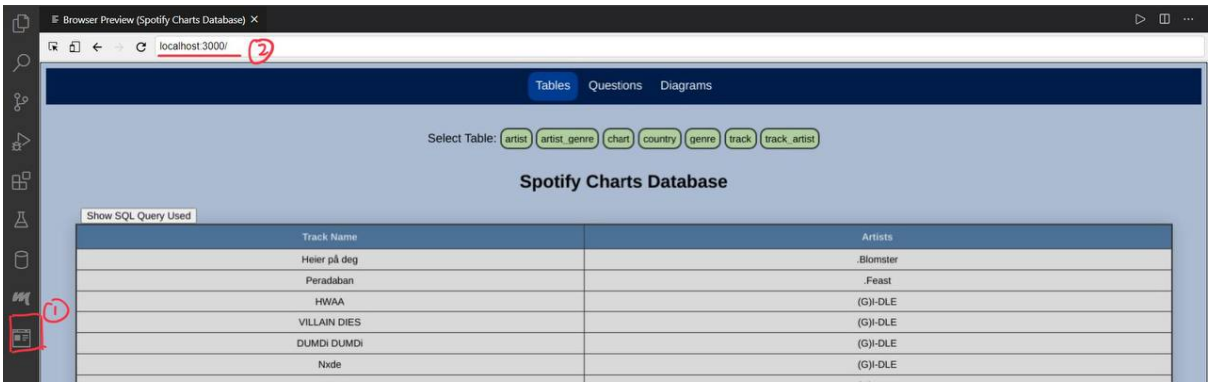
```
// connect to database
const db = mysql.createConnection({
  host: 'localhost',
  user: 'webappuser',
  password: 'hellothere',
  database: 'SpotifyCharts',
});

db.connect((err) => {
  if (err) throw err;
  console.log('Database connection successful.');
});
```

To start up the web application in the lab environment, enter "node index.js" into the terminal like shown below. Upon successful connection to the database and startup of the web application, a message is printed.

```
coder@3f9c25464770:~/project$ node index.js
App is listening on http://localhost:3000/
Database connection successful.
```

Clicking on the browser preview icon (last icon on the left sidebar) and navigating to the URL "localhost:3000" will display the main screen of the web application.

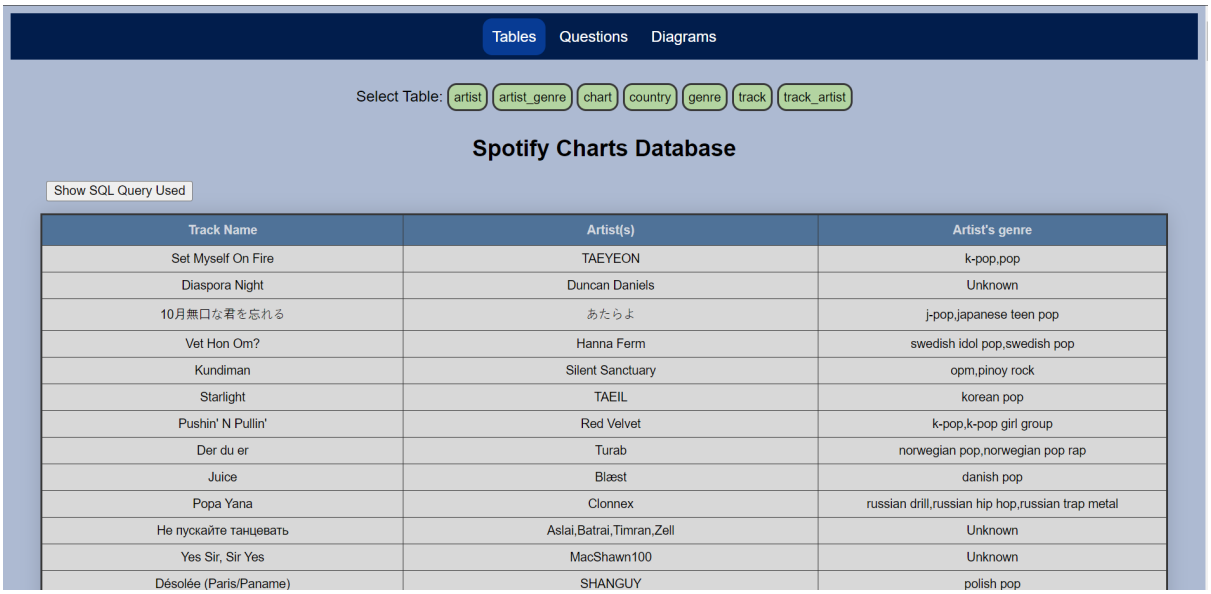


4.2 Main Screens of the web application

Note: All queries have been limited to output only up to 300 rows of records. This is to handle the lag, and to reduce the rendering time of the html pages in the lab environment.

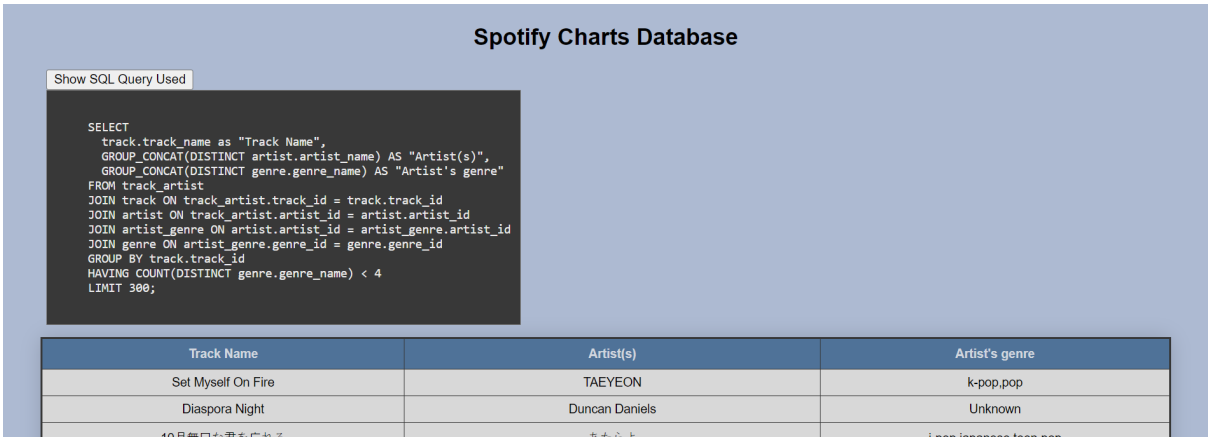
Homepage (index.html)

The homepage of the web application (URL: http://localhost:3000/), when first opened, displays the name of all tracks in the database, with the names of artists featured on each track as well as the genres associated with the artists. Clicking on the “Show SQL Query Used” button shows the query that has been used in the backend to retrieve the data that is displayed in the table.



Track Name	Artist(s)	Artist's genre
Set Myself On Fire	TAEYEON	k-pop,pop
Diaspora Night	Duncan Daniels	Unknown
10月無口な君を忘れる	あたらよ	j-pop,japanese teen pop
Vet Hon Om?	Hanna Ferm	swedish idol pop,swedish pop
Kundiman	Silent Sanctuary	opm,pinoy rock
Starlight	TAEIL	korean pop
Pushin' N Pullin'	Red Velvet	k-pop,k-pop girl group
Der du er	Turab	norwegian pop,norwegian pop rap
Juice	Blæst	danish pop
Popa Yana	Clonnex	russian drill,russian hip hop,russian trap metal
Не пускайте танцевать	Aslai,Batrai,Timran,Zell	Unknown
Yes Sir, Sir Yes	MacShawn100	Unknown
Désolée (Paris/Paname)	SHANGUY	polish pop

(Homepage)

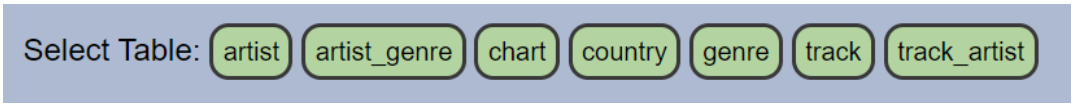


Track Name	Artist(s)	Artist's genre
Set Myself On Fire	TAEYEON	k-pop,pop
Diaspora Night	Duncan Daniels	Unknown

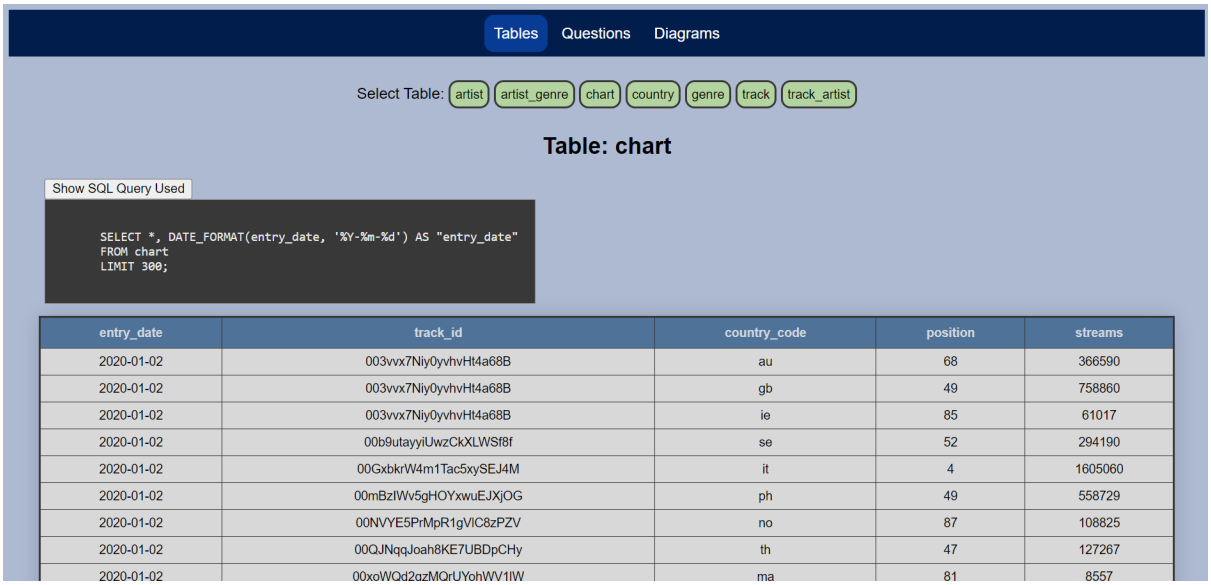
(after clicking “show SQL Query Used” button)

Homepage (index.html)

Also on the homepage, buttons have been created that correspond to each table in the database. Clicking on any of the buttons shown below would re-render the same index.html page, and the html table will then be filled with data retrieved from the database table of whichever button has been clicked.



Below shows the index.html page after the “chart” button has been clicked.



entry_date	track_id	country_code	position	streams
2020-01-02	003vvx7Niy0yvhvHt4a68B	au	68	366590
2020-01-02	003vvx7Niy0yvhvHt4a68B	gb	49	758860
2020-01-02	003vvx7Niy0yvhvHt4a68B	ie	85	61017
2020-01-02	00b9utayyiUwzCkXLWSf8f	se	52	294190
2020-01-02	00GxbkrW4m1Tac5xySEJ4M	it	4	1605060
2020-01-02	00mBzIWv5gHOYxwuEJXjOG	ph	49	558729
2020-01-02	00NVYE5PrMpR1gVIC8zPZV	no	87	108825
2020-01-02	00QJNqqJoah8KE7UBDpCHy	th	47	127267
2020-01-02	00xoWQd2azMQrUYohWV1IW	ma	81	8557

This was achieved using the “show tables” command to retrieve all table names in the backend. The table names are then sent to the html page where a button is created for each table. When a button is pressed, the corresponding table name is sent back to the backend to retrieve data from the database.

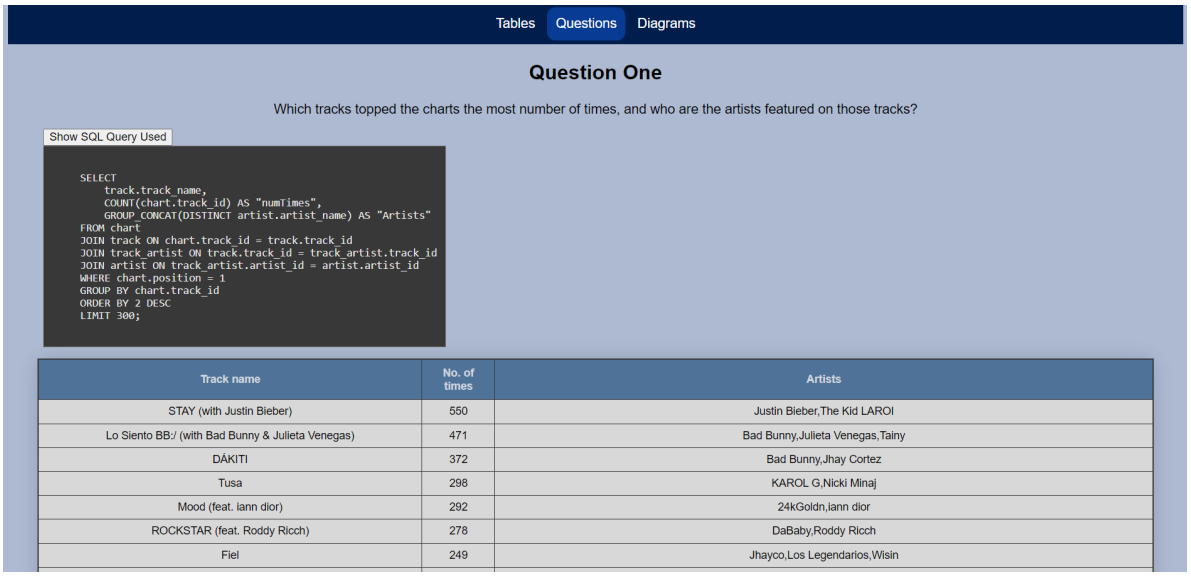
```
<!-- buttons to select which table to display -->
<div class="buttonRow">
  <form method="POST" action="/">
    Select Table:
    <% tablenames.forEach((table)=> { %>
      <button name="userOption" type="submit" value=<%=table %>>
        <%= table %>
      </button>
      <% }) %>
  </form>
</div>
```

(code to create a button for each database table)

Questions

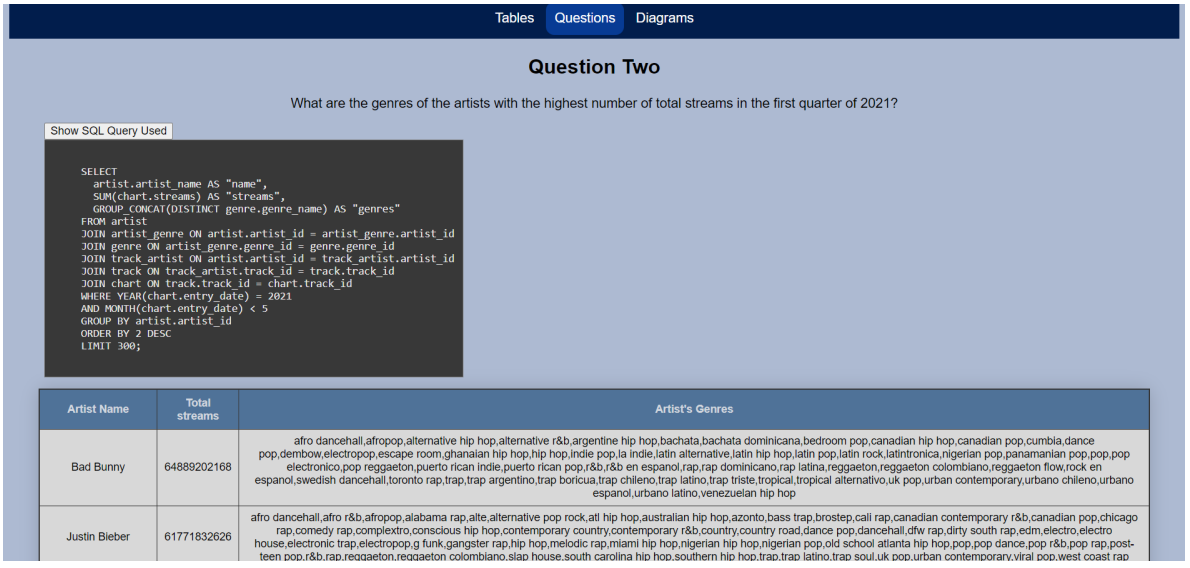
Under the “Questions” tab of the navigation bar are 3 different pages displaying the output of the queries that answer each of the questions discussed previously. Do note that these 3 pages might take some time to load due to the long query.

The first page “**question1.html**” displays the output of the query that answers the question “Which tracks topped the charts the most number of times, and who are the artists featured on those tracks?”



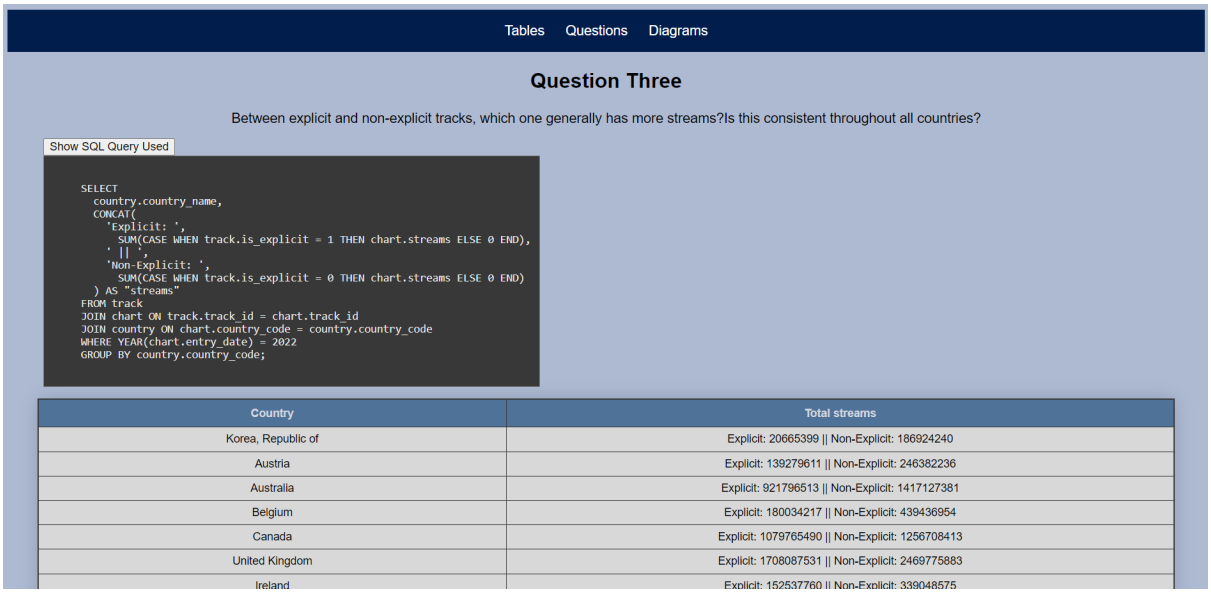
(question1.html, URL: http://localhost:3000/question1)

The second page “**question2.html**” displays the output of the query that answers the question “What are the genres of the artists with the highest number of total streams in the first quarter of 2021?”



(question2.html, URL: http://localhost:3000/question2)

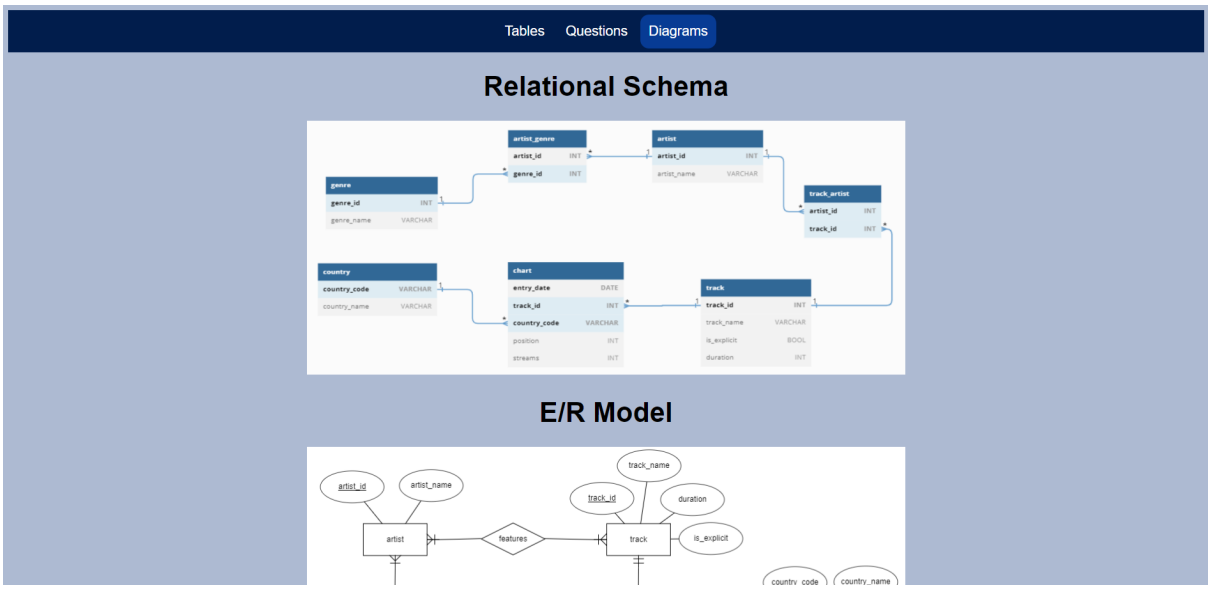
Lastly, the page “**question3.html**” displays the output of the query that answers the question “Between explicit and non-explicit tracks, which one generally has more streams? Is this consistent throughout all countries?”.



(question3.html, URL: http://localhost:3000/question3)

Diagrams (models.html)

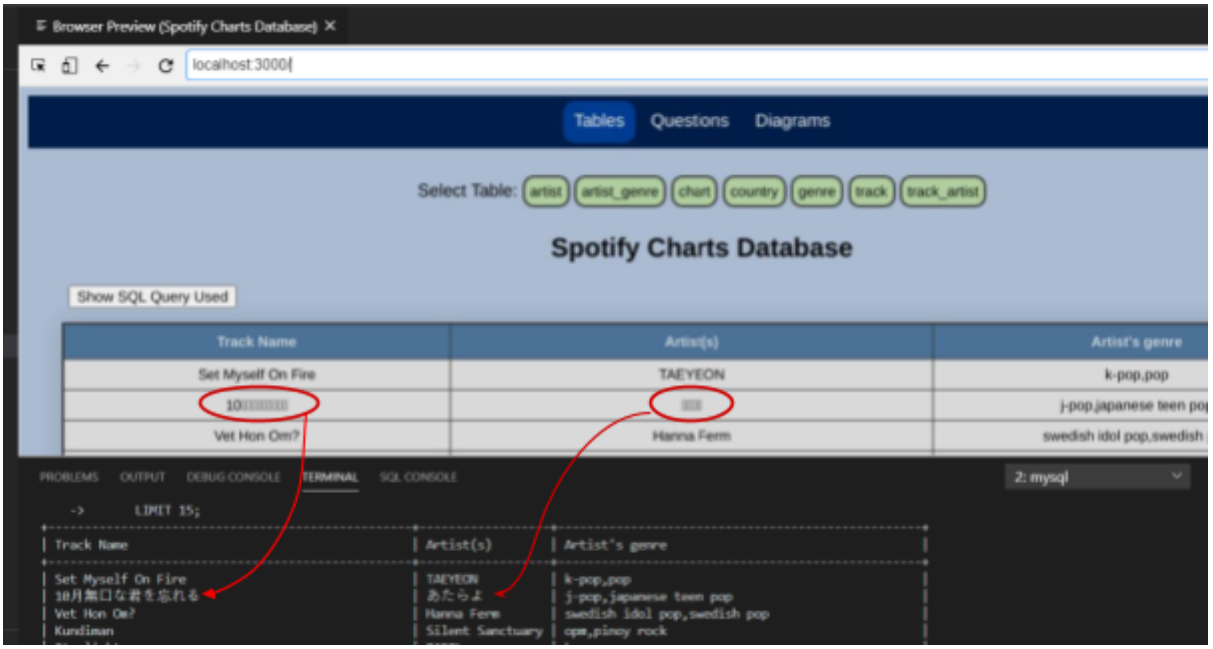
Lastly, the “Diagrams” tab simply displays the relations scheme and E/R model of the database so that it can be referred to easily.



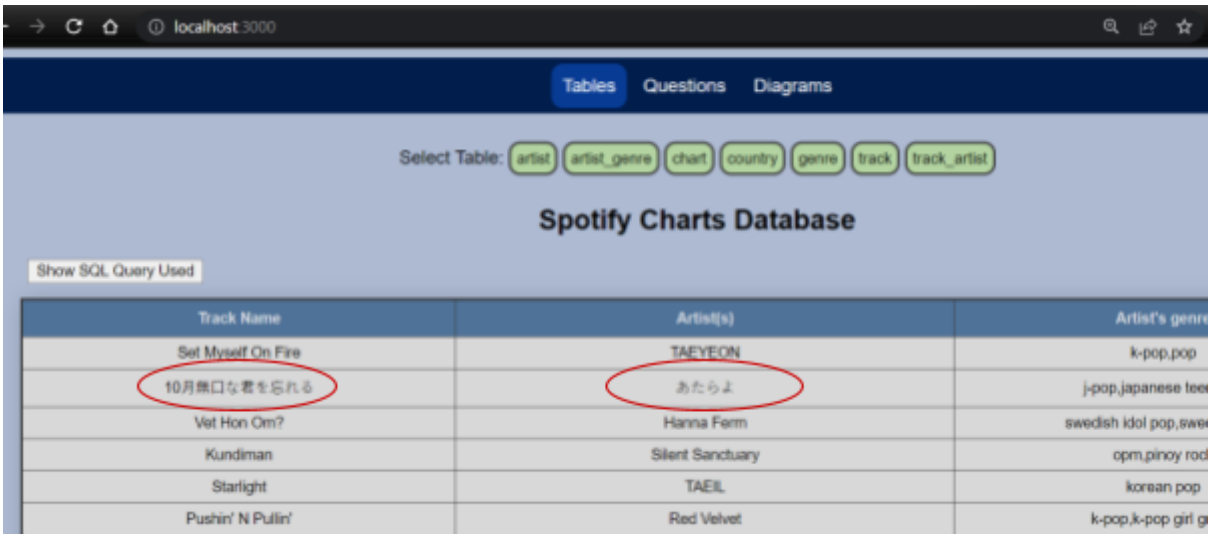
(URL: http://localhost:3000/models)

To note

When the web app is viewed on the lab environment's browser preview, non-English characters are displayed as boxes even though it displays correctly when queried from the terminal, and on my local machine's browser.



(lab's browser preview showing boxes, terminal showing actual values)



(the same page viewed on my own machine's browser preview)

References

[1] Freyberg, J. (2023) Spotify Tracks Chart Dataset (2014-2022), Kaggle. Available at: https://www.kaggle.com/datasets/jfreyberg/spotify-chart-data.

[2] Datopian (2018) List of all countries with their 2 digit codes (ISO 3166-1), DataHub. Available at: https://datahub.io/core/country-list#resource-country-list_zip.

[3] Creative Commons License Deed (no date) Creative Commons - Attribution 4.0 International - CC BY 4.0. Available at: https://creativecommons.org/licenses/by/4.0/.

[4] Kworb.net - all your music data needs in one place (no date) Kworb.net - All your music data needs in one place. Available at: https://kworb.net/.

[5] Kworb FAQ: (No date) Kworb.net - all your music data needs in one place. Available at: https://kworb.net/faq.html.

[6] Open data commons (no date) Open Data Commons Public Domain Dedication and License (PDDL) - Open Data Commons: legal tools for open data. Available at: https://opendatacommons.org/licenses/pddl/.

Appendices

6.1 Data Cleaning

Cleaning of the dataset carried out in a jupyter notebook.

cleaning

June 26, 2023

[ ]:
```
# import required libraries
import pandas as pd
import ast
```

[ ]:
```
# read csv dataset into dataframe
spotifydf = pd.read_csv("./datasets/spotify_charts.csv")

# show information
spotifydf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5428021 entries, 0 to 5428020
Data columns (total 10 columns):
 #   Column         Dtype
---  ------         -----
 0   date           object
 1   country        object
 2   position       int64
 3   streams        int64
 4   track_id       object
 5   artists        object
 6   artist_genres  object
 7   duration       int64
 8   explicit       bool
 9   name           object
dtypes: bool(1), int64(3), object(6)
memory usage: 377.9+ MB
```

I will first drop rows that I do not need from the `position` column based on my scope of only working with tracks within the top 100 position.

[ ]:
```
# drop rows with 'position' more than 100
spotifydf.drop(spotifydf[spotifydf['position'] > 100].index, inplace = True)

spotifydf['position'].sort_values().unique()
```

[ ]:
```
array([  1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,  13,
        14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,  26,
        27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,  39,
```

1

6.1 Data Cleaning (page 2 / 6)

```
            40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,  52,
            53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,  65,
            66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,  78,
            79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,  91,
            92,  93,  94,  95,  96,  97,  98,  99, 100], dtype=int64)
```

[ ]:
```
# check for null values
spotifydf.isnull().sum()
```

[ ]:
```
date              0
country           0
position          0
streams           0
track_id          0
artists           0
artist_genres     0
duration          0
explicit          0
name            407
dtype: int64
```

Based on the outputs above, the `name` column (name of track) contains 407 rows of missing data. According to Spotify's API documentation, the URL `https://open.spotify.com/track/<track_id>` can be used to open a track on the Spotify web player.

With this method, I confirmed that the tracks with a missing *name* value in my data did not show a track name even on the Spotify web player. (For example, the first *track_id* from the cell output below gives this link: https://open.spotify.com/track/60pysSgEslc7i5blU5zZbS which opens the Spotify web player with no track name displayed).

Rows without a track name is not useful in my project since it is one of the main content I will be looking at. As such, I will drop all rows with a missing `name` column.

[ ]:
```
# get first 10 rows with null name
spotifydf[spotifydf['name'].isnull()].iloc[:10]
```

[ ]:

	date	country	position	streams	track_id	\
359584	2020/05/28	br	93	765026	60pysSgEslc7i5blU5zZbS	
359585	2020/06/04	br	94	841386	60pysSgEslc7i5blU5zZbS	
438039	2020/11/26	in	36	592997	461JKAn7H6Sbx0ql9IvRUG	
438040	2020/12/03	in	18	770648	461JKAn7H6Sbx0ql9IvRUG	
438041	2020/12/10	in	15	861712	461JKAn7H6Sbx0ql9IvRUG	
438042	2020/12/17	in	13	947256	461JKAn7H6Sbx0ql9IvRUG	
438043	2021/01/14	in	15	788177	461JKAn7H6Sbx0ql9IvRUG	
755735	2017/12/07	de	2	29893815	5cjecvXOCmC9gK0Laf5EMQ	
755736	2017/12/14	de	9	13748254	5cjecvXOCmC9gK0Laf5EMQ	
755737	2017/12/21	de	25	7725621	5cjecvXOCmC9gK0Laf5EMQ	

6.1 Data Cleaning (page 3 / 6)

	artists	artist_genres	duration	explicit	name
359584	['Various Artists']	[]	0	False	NaN
359585	['Various Artists']	[]	0	False	NaN
438039	['Various Artists']	[]	0	False	NaN
438040	['Various Artists']	[]	0	False	NaN
438041	['Various Artists']	[]	0	False	NaN
438042	['Various Artists']	[]	0	False	NaN
438043	['Various Artists']	[]	0	False	NaN
755735	['Various Artists']	[]	0	True	NaN
755736	['Various Artists']	[]	0	True	NaN
755737	['Various Artists']	[]	0	True	NaN

[ ]:
```
# drop rows with null name
spotifydf.drop(spotifydf[spotifydf['name'].isnull()].index, inplace = True)

spotifydf.rename(columns={'name': 'track_name'}, inplace=True)

# verify if drop is successful
spotifydf.isnull().sum()
```

[ ]:
```
date             0
country          0
position         0
streams          0
track_id         0
artists          0
artist_genres    0
duration         0
explicit         0
track_name       0
dtype: int64
```

The `date` column stores the date where each song was charted. As my scope only limits to the year 2020 to 2023, I will drop rows not within that range. Before doing so, all dates will first have to be converted to the same datetime format.

[ ]:
```
# convert dates with '-' to '/'
spotifydf['date'] = spotifydf['date'].apply(lambda x: pd.to_datetime(x).
  ↪strftime('%Y/%m/%d') if '-' in x else x)

# convert to datetime format
spotifydf['date'] = spotifydf['date'].astype('datetime64[ns]')
spotifydf.rename(columns={'date': 'entry_date'}, inplace=True)

spotifydf['entry_date'].sort_values().unique()
```

[ ]:
```
<DatetimeArray>
['2013-04-28 00:00:00', '2013-05-05 00:00:00', '2013-05-12 00:00:00',
```

6.1 Data Cleaning (page 4 / 6)

```
 '2013-05-19 00:00:00', '2013-05-26 00:00:00', '2013-06-02 00:00:00',
 '2013-06-09 00:00:00', '2013-06-16 00:00:00', '2013-06-23 00:00:00',
 '2013-06-30 00:00:00',
 …
 '2023-02-02 00:00:00', '2023-02-09 00:00:00', '2023-02-16 00:00:00',
 '2023-02-23 00:00:00', '2023-03-02 00:00:00', '2023-03-09 00:00:00',
 '2023-03-16 00:00:00', '2023-03-23 00:00:00', '2023-03-30 00:00:00',
 '2023-04-06 00:00:00']
Length: 515, dtype: datetime64[ns]
```

[ ]:
```
# get unique years
spotifydf['entry_date'].dt.year.sort_values().unique()
```

[ ]:
```
array([2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023])
```

[ ]:
```
# drop rows with year before 2020
spotifydf.drop(spotifydf[spotifydf['entry_date'].dt.year < 2020].index, inplace
 ↪= True)

spotifydf['entry_date'].dt.year.sort_values().unique()
```

[ ]:
```
array([2020, 2021, 2022, 2023])
```

Another issue that can be observed is that the columns `artists` and `artist_genres` both contain rows with multiple values in a single cell. These two columns will have to be exploded later on during the normalisation process.

In order to ensure that it will not cause any unintended behaviour, I will convert values in the column into python list datatype. For rows containing empty lists, I will not be dropping them as those rows still contains information like the track name and its position which is useful for my research. Instead, I will be filling them with 'Unknown'.

[ ]:
```
# convert items into python list
spotifydf[['artists', 'artist_genres']] = spotifydf[['artists',
 ↪'artist_genres']].applymap(ast.literal_eval)
spotifydf[['artists', 'artist_genres']] = spotifydf[['artists',
 ↪'artist_genres']].applymap(lambda x: x if isinstance(x, list) else [])

# fill empty lists with 'Unknown'
spotifydf['artist_genres'] = spotifydf['artist_genres'].apply(lambda x: x if
 ↪len(x) > 0 else ['Unknown'])
```

In `country` column, there are rows with a value of "global". According to artists.spotify.com, global charts represents the most listened tracks throughout the Spotify platform at that time period. This information is not relevant for my area of interest of how the chartings differ by region. Therefore, I will drop rows where the country value is "global".

[ ]:
```
spotifydf.drop(spotifydf[spotifydf['country'] == 'global'].index, inplace =
 ↪True)
```

6.1 Data Cleaning (page 5 / 6)

As mentioned in the report, an additional dataset will be used to create a new `country_name` column to store the name of each country. This is so that the country can be easily identified even by people who are unfamiliar with Alpha-2 country codes.

[ ]:
```
# read dataset into dataframe
countries = pd.read_csv("./datasets/country_data.txt")

countries.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 249 entries, 0 to 248
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Name    249 non-null    object
 1   Code    248 non-null    object
dtypes: object(2)
memory usage: 4.0+ KB
```

To create new `country_name` column: 1. Compare `spotifydf[country_code]` with `countries[Code]` 2. Insert the respective country name into the new `spotifydf[Country_name]` column where the country code matches.

[ ]:
```
# convert to lowercase
countries['Code'] = countries['Code'].str.lower()

# rename for better representation
countries.rename(columns={'Name': 'country_name'}, inplace=True)
spotifydf.rename(columns={'country': 'country_code'}, inplace=True)
```

[ ]:
```
# do a left join to insert country name based on country codes
spotifydf = pd.merge(spotifydf, countries[['Code', 'country_name']],␣
 ↪how='left', left_on='country_code', right_on='Code')
spotifydf.drop('Code', axis=1, inplace=True)
```

[ ]:
```
# get unique country name and code pairs to check
spotifydf[['country_name', 'country_code']].drop_duplicates().head()
```

[ ]:

	country_name	country_code
0	Germany	de
1	Colombia	co
2	Egypt	eg
38	Brazil	br
62	Israel	il

Now that the data has been cleaned, all that is left to do is to sort the values based on the entry_date, country_code, and position. This will allow better visualisation of the data moving forward.

6.1 Data Cleaning (page 6 / 6)

[ ]:
```
# sort values
spotifydf.sort_values(['entry_date', 'country_code', 'position'], inplace =␣
↪True)
spotifydf.head()
```

[ ]:
```
       entry_date country_code  position  streams                track_id  \
420463 2020-01-02           ae         1    56126  1rgnBhdG2JDFTbYkYRZAku
28156  2020-01-02           ae         2    52210  2rRJrJEo19S2J82BDsQ3F7
389975 2020-01-02           ae         3    46625  696DnlkuDOXcMAnKlTgXXK
629265 2020-01-02           ae         4    34453  2b8fOow8UzyDFAE27YhOZM
323651 2020-01-02           ae         5    31003  6v3KW9xbzN5yKLt9YKDYA2

                              artists  \
420463                  [Tones And I]
28156                 [Trevor Daniel]
389975                [Arizona Zervas]
629265                     [Maroon 5]
323651  [Shawn Mendes, Camila Cabello]

                                        artist_genres  duration  explicit  \
420463                          [pop, australian pop]    209754     False
28156                      [pop, pop rap, melodic rap]    159381     False
389975       [pop rap, rhode island rap, pop, viral rap]    163636      True
629265                                            [pop]    189486     False
323651  [viral pop, pop, canadian pop, uk pop, dance p…    190799     False

          track_name          country_name
420463  Dance Monkey  United Arab Emirates
28156        Falling  United Arab Emirates
389975       ROXANNE  United Arab Emirates
629265      Memories  United Arab Emirates
323651      Señorita  United Arab Emirates
```

Lastly, save the cleaned data into a new csv file.

[ ]:
```
# save cleaned dataframe into a new csv file
pd.DataFrame.to_csv(spotifydf, './datasets/spotify_charts_cleaned.csv',␣
↪index=False)
```

6.2 Normalisation

Normalisation of the cleaned dataset carried out in a jupyter notebook.

normalisation

June 27, 2023

1 Normalising the Dataset

[ ]:
```
# import required libraries
import pandas as pd
import ast
import csv
```

[ ]:
```
# read csv of cleaned dataset into a dataframe
Chart = pd.read_csv("./datasets/spotify_charts_cleaned.csv")
Chart.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1077101 entries, 0 to 1077100
Data columns (total 11 columns):
 #   Column         Non-Null Count    Dtype
---  ------         --------------    -----
 0   entry_date     1077101 non-null  object
 1   country_code   1077101 non-null  object
 2   position       1077101 non-null  int64
 3   streams        1077101 non-null  int64
 4   track_id       1077101 non-null  object
 5   artists        1077101 non-null  object
 6   artist_genres  1077101 non-null  object
 7   duration       1077101 non-null  int64
 8   explicit       1077101 non-null  bool
 9   track_name     1077101 non-null  object
 10  country_name   1077101 non-null  object
dtypes: bool(1), int64(3), object(7)
memory usage: 83.2+ MB
```

1.1 1NF

To satisfy 1NF:

- For columns containing multiple values, split each value into their own row of data.
- Set `entry_date`, `track_id`, and `country_code` as the index.

1

6.2 Normalisation (page 2/6)

[ ]:
```
# convert into python list object
Chart[['artists', 'artist_genres']] = Chart[['artists', 'artist_genres']].
 ↪applymap(ast.literal_eval)
Chart[['artists', 'artist_genres']] = Chart[['artists', 'artist_genres']].
 ↪applymap(lambda x: x if isinstance(x, list) else [])
Chart.reset_index(drop=True, inplace=True)
```

[ ]:
```
# explode artists and artist_genres column
Chart = Chart.explode('artists').explode('artist_genres')
Chart.rename(columns={'artists': 'artist_name', 'artist_genres' :␣
 ↪'genre_name'}, inplace=True)
Chart.head()
```

[ ]:
```
   entry_date country_code  position  streams                track_id  \
0  2020-01-02           ae         1    56126  1rgnBhdG2JDFTbYkYRZAku
0  2020-01-02           ae         1    56126  1rgnBhdG2JDFTbYkYRZAku
1  2020-01-02           ae         2    52210  2rRJrJEo19S2J82BDsQ3F7
1  2020-01-02           ae         2    52210  2rRJrJEo19S2J82BDsQ3F7
1  2020-01-02           ae         2    52210  2rRJrJEo19S2J82BDsQ3F7

     artist_name      genre_name  duration  explicit    track_name  \
0    Tones And I             pop    209754     False  Dance Monkey
0    Tones And I  australian pop    209754     False  Dance Monkey
1  Trevor Daniel             pop    159381     False       Falling
1  Trevor Daniel         pop rap    159381     False       Falling
1  Trevor Daniel     melodic rap    159381     False       Falling

          country_name
0  United Arab Emirates
0  United Arab Emirates
1  United Arab Emirates
1  United Arab Emirates
1  United Arab Emirates
```

[ ]:
```
# ensure no null values
Chart.isnull().sum()
```

[ ]:
```
entry_date      0
country_code    0
position        0
streams         0
track_id        0
artist_name     0
genre_name      0
duration        0
explicit        0
track_name      0
```

6.2 Normalisation (page 3/6)

```
country_name    0
dtype: int64
```

The code below sets 'entry_date', 'track_id', 'country_code' as the composite key of the Chart table, which would allow unique identification of each row of data. I will not run this code as it would add constraints which makes splitting of the dataframe difficult in the next steps.

[ ]:
```
# Chart.set_index(['entry_date', 'track_id', 'country_code'], inplace=True)
```

1.2 2NF & 3NF

To satify both 2NF and 3NF:

- Create Track table (columns: `track_id`, `track_name`, `duration`, `explicit`).
- Create Country table (columns: `country_code`, `country_name`).
- Create Artist table (columns: `artist_id`, `artist_name`);
- Create Genre table (columns: `genre_id`, `genre_name`).
- Create Artist_Genre junction table (columns: `artist_id`, `genre_id`).
- Create Track_Artist junction table (columns: `track_id`, `artist_id`).
- Drop all partial dependencies from Chart table.

Track Table

[ ]:
```
# create Track table with track_id as primary key
Track = Chart.drop_duplicates('track_id').copy()

Track = Track[['track_id', 'track_name', 'duration', 'explicit']]
Track['explicit'] = Track['explicit'].astype(int)

Track.to_csv('csvTables/Track.csv', index=False, quoting=csv.QUOTE_ALL)

Track.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 38556 entries, 0 to 1077093
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   track_id    38556 non-null  object
 1   track_name  38556 non-null  object
 2   duration    38556 non-null  int64
 3   explicit    38556 non-null  int32
dtypes: int32(1), int64(1), object(2)
memory usage: 1.3+ MB
```

Country Table

[ ]:
```
# create Country table with country_code as primary key
Country = Chart.drop_duplicates('country_code').copy()
```

6.2 Normalisation (page 4/6)

```
Country = Country[['country_code', 'country_name']]

Country.to_csv('csvTables/Country.csv', index=False, quoting=csv.QUOTE_ALL)

Country.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 76 entries, 0 to 850698
Data columns (total 2 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   country_code  76 non-null     object
 1   country_name  76 non-null     object
dtypes: object(2)
memory usage: 1.8+ KB
```

Artist Table

[ ]:
```
# create Artist table with artist_id as primary key
Artist = Chart.drop_duplicates('artist_name').copy()

# create new artist_id column to use as primary key
Artist['artist_id'] = range(1, len(Artist)+1)

Artist = Artist[['artist_id', 'artist_name']]

Artist.to_csv('csvTables/Artist.csv', index=False, quoting=csv.QUOTE_ALL)

Artist.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 15239 entries, 0 to 1077093
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   artist_id    15239 non-null  int64
 1   artist_name  15239 non-null  object
dtypes: int64(1), object(1)
memory usage: 357.2+ KB
```

Genre Table

[ ]:
```
# create Genre table with genre_id as primary key
Genre = Chart.drop_duplicates('genre_name').copy()

# create new artist_id column to use as primary key
Genre['genre_id'] = range(1, len(Genre)+1)

Genre = Genre[['genre_id', 'genre_name']]
```

6.2 Normalisation (page 5/6)

```
Genre.to_csv('csvTables/Genre.csv', index=False, quoting=csv.QUOTE_ALL)

Genre.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2099 entries, 0 to 1077085
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   genre_id    2099 non-null   int64
 1   genre_name  2099 non-null   object
dtypes: int64(1), object(1)
memory usage: 49.2+ KB
```

```
[ ]: # drop track_name, duration, explicit from Chart table
Chart.drop(['track_name', 'duration', 'explicit', 'country_name'], axis=1,␣
 ↪inplace=True)
print(Chart.shape)
```

```
(7543681, 7)
```

Artist_Genre Table

```
[ ]: # derive `artist_id` from `artist_name`
Chart = Chart.merge(Artist[['artist_id', 'artist_name']], on='artist_name',␣
 ↪how='left')
# derive `genre_id` from `genre_name`
Chart = Chart.merge(Genre[['genre_id', 'genre_name']], on='genre_name',␣
 ↪how='left')

Chart.drop(['artist_name', 'genre_name'], axis=1, inplace=True)
```

```
[ ]: # Create Artist_Genre table
Artist_Genre = Chart.drop_duplicates(subset=['artist_id', 'genre_id']).copy()
Artist_Genre = Artist_Genre[['artist_id', 'genre_id']]

Artist_Genre.to_csv('csvTables/Artist_Genre.csv', index=False, quoting=csv.
 ↪QUOTE_ALL)

Artist_Genre.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 69746 entries, 0 to 7543607
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   artist_id  69746 non-null  int64
 1   genre_id   69746 non-null  int64
```

6.2 Normalisation (page 6/6)

```
dtypes: int64(2)
memory usage: 1.6 MB
```

Track_Artist Table

[ ]:
```
# Create Track_Artist table
Track_Artist = Chart.drop_duplicates(subset=['track_id', 'artist_id']).copy()
Track_Artist = Track_Artist[['track_id', 'artist_id']]

Track_Artist.to_csv('csvTables/Track_Artist.csv', index=False, quoting=csv.
 ↪QUOTE_ALL)

Track_Artist.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 62760 entries, 0 to 7543606
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   track_id   62760 non-null  object
 1   artist_id  62760 non-null  int64
dtypes: int64(1), object(1)
memory usage: 1.4+ MB
```

Chart Table

[ ]:
```
# drop artist_id and genre_id
Chart.drop(['artist_id', 'genre_id'], axis=1, inplace=True)
Chart.drop_duplicates(inplace=True)

# rearrange columns
Chart = Chart.reindex(columns=['entry_date', 'track_id', 'country_code',␣
 ↪'position', 'streams'])

Chart.to_csv('csvTables/Chart.csv', index=False, quoting=csv.QUOTE_ALL)

Chart.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1077101 entries, 0 to 7543671
Data columns (total 5 columns):
 #   Column        Non-Null Count    Dtype
---  ------        --------------    -----
 0   entry_date    1077101 non-null  object
 1   track_id      1077101 non-null  object
 2   country_code  1077101 non-null  object
 3   position      1077101 non-null  int64
 4   streams       1077101 non-null  int64
dtypes: int64(2), object(3)
```


- END OF REPORT -