CSCI620-01 Project

Phase 1 Write Up

Group 5:

Vinod Dalavai | vd1605

Ramprasad Kokkula | rk1668

Samson Zhang | sz7651

March 3, 2023

1 Dataset Description

1.1 Overveiw and Source:

Our dataset contains information on one million playlists created by Spotify users. The dataset is sourced from Kaggle, which can be found here

1.2 Files Description:

1.2.1 Overview

The dataset is stored in a thousand json files, with each file containing a thousand playlists (which totals to a million).

The json files follows the naming pattern of:

$$\text{mpd.slice.[starting playlist number]} - \text{[ending playlist number]}$$

For example, “mpd.slice.0-999” contains the first 1,000 playlists, note the use of 0-based numbering.

1.2.2 Formatting

Each of the json files contains the following:

$$\{\text{info, playlists}\}$$

Where *info* is the metadata of the file and contains the following:

1. “generated_on”: time when the data was generated
2. “slice”: the slice of playlists that this file contains (e.g. 0-999)
3. “version” the version of this file/data

A *playlist* contains the following:

1. “name”: name of the playlist
2. “collaborative”: whether playlist was made by more than one person
3. “pid”: playlist id
4. “modified_at”: the last time the playlist was modified
5. “num_tracks”: number of tracks in the playlist
6. “num_albums”: number of albums in the playlist

7. "num_followers": number of followers of the playlist
8. **"tracks"**: a *list* of all the tracks, **details below**
9. "num_edits": number of times playlist was edited
10. "duration_ms": total duration of playlist in milliseconds
11. "num_artists": number of artists in the playlist

A single *track* contains the following:

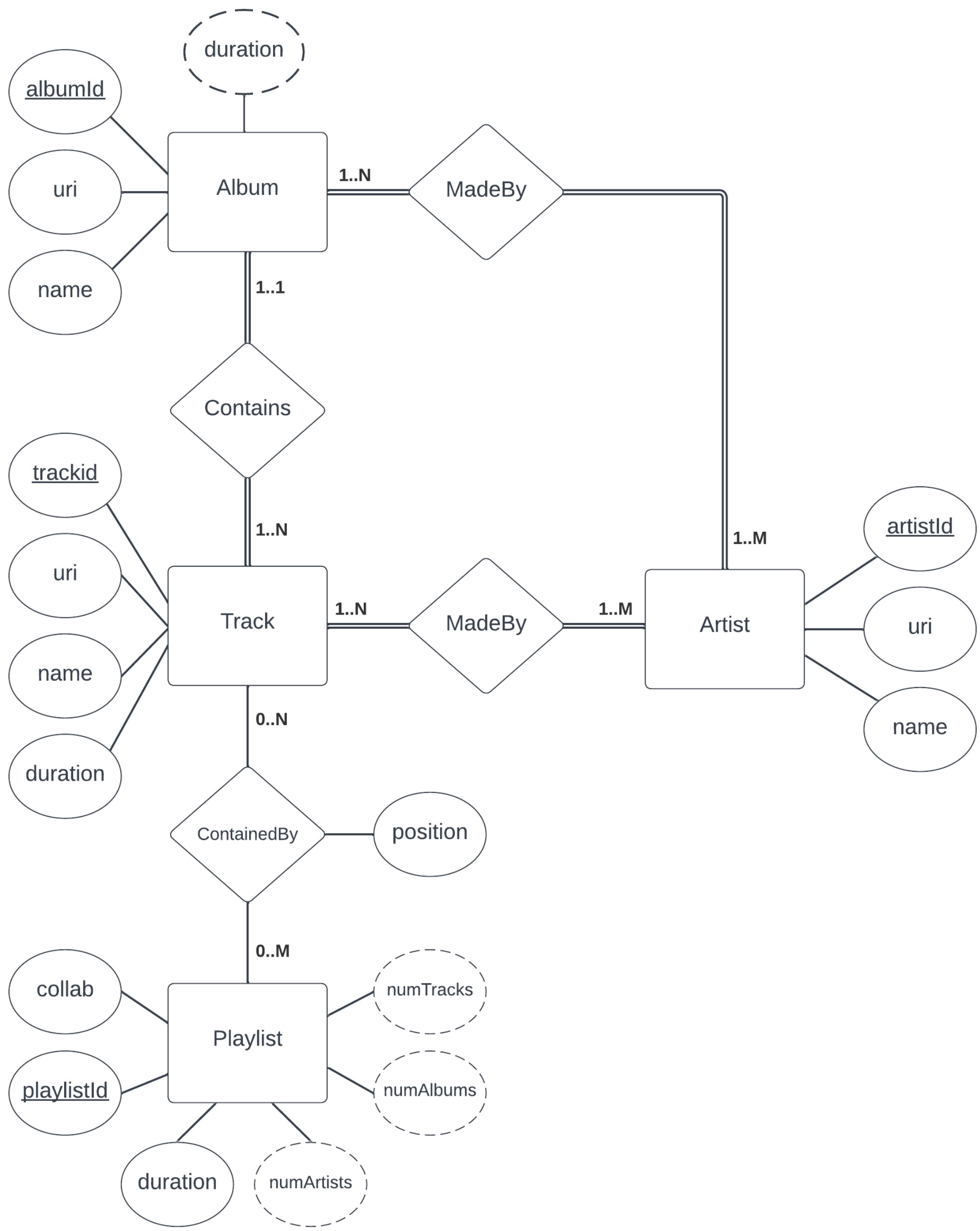
1. "pos": the position of this track in the playlist
2. "artist_name": artist of this track
3. "track_uri": Spotify URI of this track
4. "artist_uri": Spotify URI of the artist of this track
5. "track_name": name of this track
6. "album_uri": Spotify URI of the album of this track
7. "duration_ms": duration of this track in milliseconds
8. "album_name": name of the album of this track

Note that in our relational model, we leave out certain attributes. For example: we do not store "num_edits" and "num_followers" for a playlist. We are excluding these attributes because we don't think they represent useful information for our use case.

Additionally, some of the attributes can be derived. Therefore, these attributes are also not directly stored. For example, "num_trakcs" and "num_albums" in a playlist can be derived through queries.

While a playlist's duration can also be derived, we will need to join tables in order to get the duration of each track contained in the playlist. Because the dataset already provides playlists' duration, we store it for ease of access.

2 ER Diagram



duration
albumId
uri
name
Album
1..N
MadeBy
1..1
Contains
trackid
uri
name
duration
1..N
Track
1..N
MadeBy
1..M
Artist
1..M
artistId
uri
name
0..N
ContainedBy
position
0..M
collab
playlistId
Playlist
numTracks
numAlbums
duration
numArtists

3 Relational Model

3.1 Reduction to Tables

album(id, album_uri, name)
track(id, track_uri, name, duration, *album_id*)
artist(id, artist_uri, name)
playlist(playlist_id, collab, duration)
album_artist(*album_id*, *artist_id*)
track_artist(*track_id*, *artist_id*)
track_playlist(*track_id*, *playlist_id*, position)

3.2 Entities DDL

```
CREATE TABLE album(
id int PRIMARY KEY,
album_uri char(255),
name varchar)
)
```

```
CREATE TABLE track(
id int PRIMARY KEY,
track_uri char(255),
name varchar,
duration int,
album_id int,
FOREIGN KEY (album_id) REFERENCES album(id)
)
```

```
CREATE TABLE artist(
id int PRIMARY KEY,
artist_uri char(255),
name varchar
)
```

```
CREATE TABLE playlist(
playlist_id int PRIMARY KEY
collab boolean,
duration int
)
```

For each uri, we assume a max length of 255 characters. Because some names are longer than 255 characters, we use varchar to store them insead.

Note that because the "Album Contains Track" relationship is one to many, we just need to store albumid as a Foreign Key in the Track table (we only need to create a new table if the relationship is many to many).

3.3 N:M Relationships DDL

```
CREATE TABLE album_artist(
album_id int,
artist_id int,
PRIMARY KEY (album_id, artist_id),
FOREIGN KEY (album_id) REFERENCES album(id)
FOREIGN KEY (artist_id) REFERENCES artist(id)
)

CREATE TABLE track_artist(
track_id int,
artist_id int,
PRIMARY KEY (track_id, artist_id),
FOREIGN KEY (track_id) REFERENCES track(id)
FOREIGN KEY (artist_id) REFERENCES artist(id)
)

CREATE TABLE track_playlist(
track_id int,
playlist_id int,
position int,
PRIMARY KEY (track_id, playlist_id, position),
FOREIGN KEY (track_id) REFERENCES track(id)
FOREIGN KEY (playlist_id) REFERENCES playlist(playlist_id)
)
```

Our schema has a total of four relationships, three of them are many to many relationships. For each of the many to many relationships, we create a new table. The new table will store foreign keys referencing the ids of the entities that participates in the relationship.

Additionally, the relationship between Track and Playlist has its own attribute (position), which is included in the table that represents the relationship (Track_Playlist).

4 Loading The Data

4.1 Overview

The data loading process is split into multiple steps.

1. The json files are read, and a single csv file containing all the data is created.
2. Specific columns from the csv file is used to create temporary csv files corresponding to each table of the schema.
3. Temporary tables are created, these tables do not have any constraints to make initial data loading easier.
4. The content of the temporary csv files is inserted into the corresponding temporary tables.
5. The temporary tables are used to create tables that match the actual schema.
6. The temporary csv files are deleted and the temporary tables are dropped.

The program files (along with this write up) can also be found here

4.2 main.py

This is the entry point to the program. This file runs both converter.py and database_initializer.py. Credentials for the database as well as the path to the data folder are passed in as arguments.

Note that, if needed, both converter.py and database_initializer.py can be run on their own, as they have their own main functions.

4.3 converter.py

The converter reads the json files containing the dataset, and creates a single csv file using it.

Given a path to the folder containing input data files, the program will read and process each of the files one by one. For each file, the json file is converted into a python dictionary using json.load(), which is then used to create a pandas data frame. The data frame is used to create the "output_playlist_data.csv" file. For each files being processed after the first, the data is appended to the end of the csv file (as opposed to overwriting the csv file).

The resulting output csv file contains all of the attributes/columns.

4.4 database_initializer.py

The initializer first connects to a database with the passed in arguments as credentials.

All of the scripts that will be executed by the initializer are stored in the sql_scripts folder, and they are referenced through the scripts.py file, which is an enum class that stores the paths to each of the sql scripts. The enum classes are used to make code easier to understand, for example, we can refer to 'sql_scripts/create_schema.sql' as just 'SCHEMA'.

The schema is created first by running the script create_schema.sql. This script is similar to the DDL statements seen in the previous section (Relational Model), however, to make data loading easier, every attribute of every table is initialized as CHAR or VARCHAR type. This is because we use the COPY command to bulk load the data into the tables, and we treat every data entry from the files as strings. Additionally, the COPY command skips integrity checks so it is faster. However, because there are no constraints initially, when a table is populated, it is altered so that its attributes are of the correct types (specified in the previous section).

Then, to populate each of the tables, the csv file created by converter.py is read in as a pandas data frame. Here we have both table_columns.py and temp_files.py as enum classes to make code more readable. table_columns.py contains information on what columns are needed for a specific table, whereas temp_files.py contains information on the path of each of the temporary csv files to be created.

By specifying which columns of the pandas data frame we want, we create temporary csv files that matches a specific table. For example: the playlist table has columns 'pid', 'name', 'collaborative', and 'duration_ms', therefore, when creating the playlist temporary csv file, only those columns from the data frame are included.

We then copy the data from the temporary csv file into a temporary table with no constraints. Then load the actual tables with the temporary table. The reason behind this is because we can load data into the database more easily if there are no constraints, and maintain those constraints when we insert the data from the temporary table into the actual table. Note that the temporary csv files are deleted and the temporary tables are dropped after they are used.

For example, the following is the script used to populate playlist:

```
COPY temp_playlist
        FROM '/Users/vinoddalavai/LocalDocuments/
          CSCI620_BigData/Project/temp/temp_playlist.csv'
        DELIMITER E','
        CSV HEADER;

SELECT DISTINCT temp_playlist.playlistid, temp_playlist.name,
                temp_playlist.collab, temp_playlist.duration
INTO playlist
FROM temp_playlist;

ALTER TABLE playlist
ALTER COLUMN playlistid TYPE INT USING playlistid::INTEGER,
ALTER COLUMN collab TYPE BOOLEAN USING collab::BOOLEAN,
ALTER COLUMN duration TYPE INT USING duration::INTEGER;

ALTER TABLE playlist
ADD PRIMARY KEY (playlistid);

DROP TABLE temp_playlist;
```

A temp_playlist table is created using the COPY command and the temp_playlist.csv file. When inserting data into the actual playlist table, a constraint is that we only want distinct rows, which is specified in the SELECT INTO statement. We then ALTER the table to make sure that it matches the schema and, finally, we drop the temporary table.

Do note that the path to the file used in the COPY command needs to be changed when running on different machines (since the path will be different).

A similar process is done for every table of the schema that does not represent a relationship.

For tables that represents relationships, the participating entities' URIs are the only things stored initially. Because in the original data set, the URIs are used to uniquely identify an artist, album, or track (note that the playlists already have integer ids). We cannot store the integer ids initially because we only created the serial integer ids for these entities as we created their tables.

By joining the temporary relationship tables with the entities participating in the relationship on the URIs, we can then access the corresponding integer ids of the entities and use it to create the final relationship tables.

4.5 Enum Classes

The enum classes are very simple, as their main purpose is to define variable names to "stand-in" for otherwise long variables, which makes code easier to understand. They are mentioned in the previous subsection (4.3), but the actual code is also included here to more clearly show what they are.

scripts.py

```
from enum import Enum

class Scripts(Enum):
    SCHEMA = 'sql_scripts/create_schema.sql'
    PLAYLIST = 'sql_scripts/populate_playlist.sql'
    ARTIST = 'sql_scripts/populate_artist.sql'
    TRACK = 'sql_scripts/populate_track.sql'
    TRACK_PLAYLIST = 'sql_scripts/populate_track_playlist.sql'
```

table_columns.py

```
from enum import Enum

class TableColumns(Enum):
    PLAYLIST = ['pid', 'name', 'collaborative', 'duration_ms']
    ARTIST = ['artist_uri', 'artist_name']
    TRACK = ['track_uri', 'track_name', 'album_uri']
    TRACK_PLAYLIST = ['track_uri', 'pid', 'pos']
```

temp_files.py

```
from enum import Enum

class TempFiles(Enum):
    PLAYLIST = 'temp/temp_playlist.csv'
    ARTIST = 'temp/temp_artist.csv'
    TRACK = 'temp/temp_track.csv'
    TRACK_PLAYLIST = 'temp/temp_track_playlist.csv'
```

For example, when running a script, instead of specifying to run the script 'sql_scripts/create_schema.sql', the Scripts enum class allows us to refer to it as 'SCHEMA'. Similarly, instead of specifying the columns in the playlist table as a list, we can just refer to the columns as 'PLAYLIST'. The same concept applies to TempFiles.

4.6 time_logger.py

A simple script used to time each part of the data loading process. The time returned changes the unit from seconds to minutes if the run time is greater than 60 seconds.

4.7 Program Output

The following is the output from running the main.py script on an m2 MacBook Air, note that this result corresponds to loading only the first 250K playlists:

```
python3 main.py data/ localhost project
Converting json files to csv file ...
Execution Time = 2.79 minutes

Connecting to database ...
Creating schema ...
Preparing in-memory dataframe ...
Execution Time = 1.19 minutes

Populating playlists ...
Execution Time = 42.86 seconds

Populating artists ...
Execution Time = 2.24 minutes

Populating albums ...
Execution Time = 2.66 minutes

Populating tracks ...
Execution Time = 7.76 minutes

Populating track_playlists ...
Execution Time = 4.64 minutes

Populating track_artists ...
Execution Time = 7.53 minutes

Populating album_artists ...
Execution Time = 2.52 minutes
```

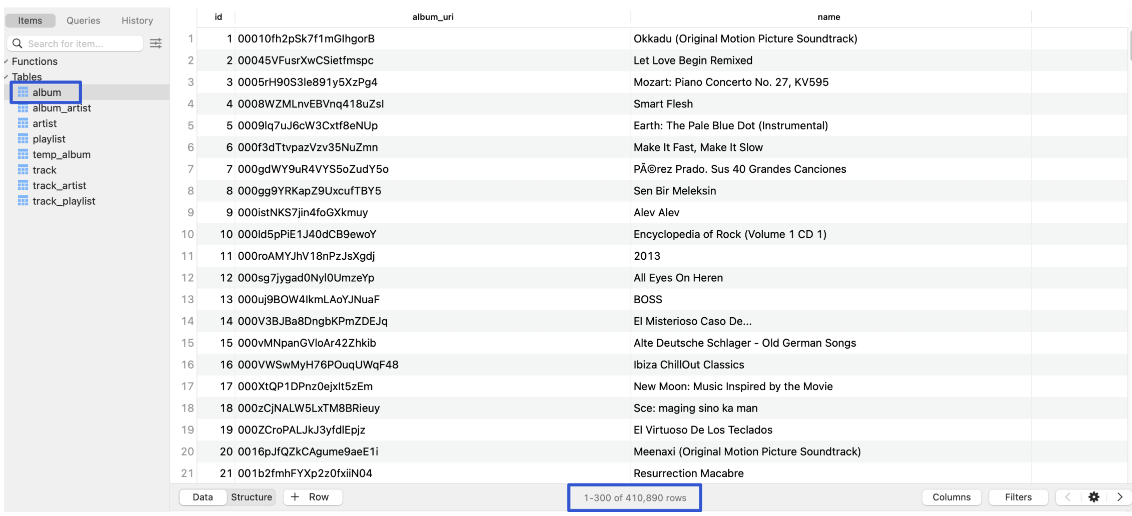
This sums up to about 32 minutes of run time to load files from "mpd.slice.0-999.json" to "mpd.slice.240000-249999.json" (the resulting data set corresponds to tracks, artists, album, etc. in the first 250,000 playlists).

By multiplying by 4 (250K is a quarter of a million), we can roughly approximate the run time of loading all files to be 128 minutes, or a bit over two hours.

5 Sample Data

Here are some sample data from each table:

album:

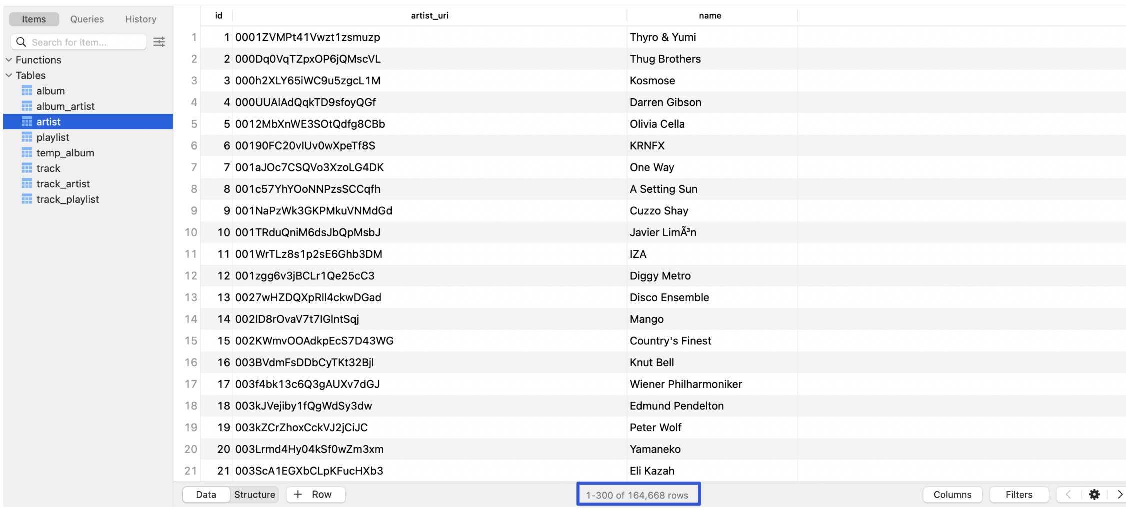


	id	album_uri	name
1	1	00010fh2pSk7f1mGlhgorB	Okkadu (Original Motion Picture Soundtrack)
2	2	00045VFusrXwCSietfmspc	Let Love Begin Remixed
3	3	0005rH90S3le891y5XzPg4	Mozart: Piano Concerto No. 27, KV595
4	4	0008WZMLnvEBVnq418uZsl	Smart Flesh
5	5	0009lq7uJ6cW3Cxtf8eNUp	Earth: The Pale Blue Dot (Instrumental)
6	6	000f3dTtvpazVzv35NuZmn	Make It Fast, Make It Slow
7	7	000gdWY9uR4VYS5oZudY5o	PÃ©rez Prado. Sus 40 Grandes Canciones
8	8	000gg9YRKapZ9UxcufTBY5	Sen Bir Meleksin
9	9	000istNKS7jin4foGXkmuy	Alev Alev
10	10	000ld5pPiE1J40dCB9ewoY	Encyclopedia of Rock (Volume 1 CD 1)
11	11	000roAMYJhV18nPzJsXgdj	2013
12	12	000sg7jygad0Nyl0UmzeYp	All Eyes On Heren
13	13	000uj9BOW4lkmLAoYJNuaF	BOSS
14	14	000V3BJBa8DngbKPmZDEJq	El Misterioso Caso De...
15	15	000vMNpanGVloAr42Zhkib	Alte Deutsche Schlager - Old German Songs
16	16	000VWSwMyH76POuqUWqF48	Ibiza ChillOut Classics
17	17	000XtQP1DPnz0ejxlt5zEm	New Moon: Music Inspired by the Movie
18	18	000zCjNALW5LxTM8BRieuy	Sce: maging sino ka man
19	19	000ZCroPALJkJ3yfdlEpjz	El Virtuoso De Los Teclados
20	20	0016pJfQZkCAgume9aeE1i	Meenaxi (Original Motion Picture Soundtrack)
21	21	001b2fmhFYXp2z0fxiiN04	Resurrection Macabre

track:

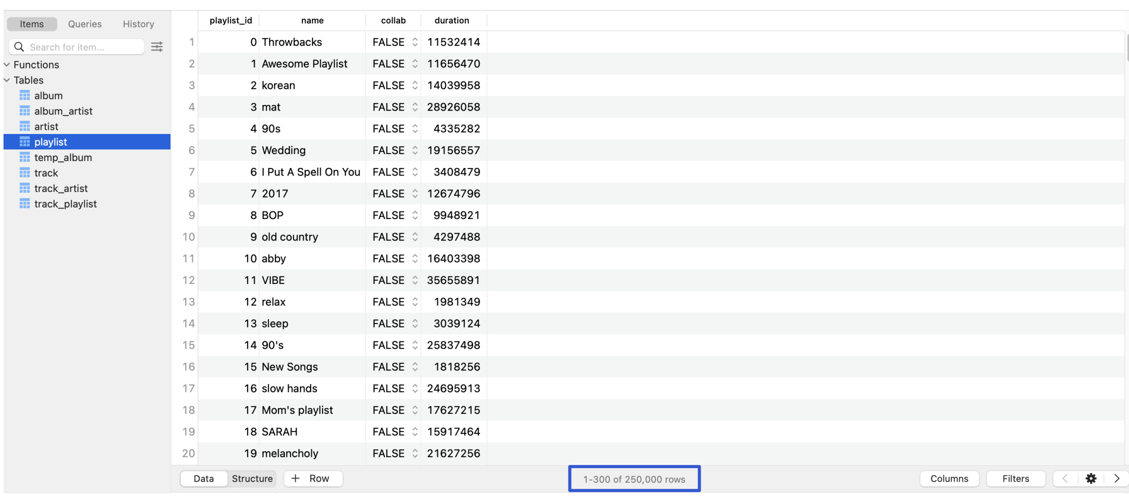
	id	track_uri	name	album_uri	duration	album_id
1	1	0000uJA4xCdxThagdLkkLR	Heart As Cold As Stone	3SPMBGMEvPw21lmT5b1ApW	161186	198954 →
2	2	00039MgrmLoIzSpuYKurn9	Thas What I Do	0UHfgx3ITlxePDXLaN5Y6x	222727	43543 →
3	3	0005w1bMJ7QAMl6DY98oxa	Sonata in G Major, BuxWV 271: Allegro -	6oRWciCAwKegipcCc5FlWe	111573	350015 →
4	4	0006Rv1e2Xfh6QooyKJqKS	Nightwood	1BD29pKydSXe1EsHFj0GrQ	189638	64005 →
5	5	0007AYhg2UQbEm88mxu7js	Mandarin Oranges Part 2	32RJzqlapfiU0fr2l4SSW9	198000	161268 →
6	6	000CTwOSsvRs0bgXlwB64e	Shady World	6LOmO9x8uPwlc3gSXZqqi8	155238	344813 →
7	7	000GjfnQc7ggBayDiy1sLW	Abeja Miope	18XgaglyP4lDDZQFS3yQBW	140360	60904 →
8	8	000JBgYWfJQqdFaRqu2n3f	Li'l Darlin'	2zm8AkHjOhwdvnj8NSVg6g	227626	158156 →
9	9	000JCyEkMFumqCZQJAORiQ	California Water	4pfWrpnLFM1jHiw3foRpKK	207124	245858 →
10	10	000mA0etY38nKdvf1N04af	If I Gave Myself To Someone Else	662PiU3dRsilN0Gp87liSF	214506	321638 →
11	11	000pmJ2wC4EvNSbXa3aHwT	Kinky	7Lr9MKDWbncltiuCUwK1ET	115693	397832 →
12	12	000QFYFej3TzZq8HCTK36j	The Cold Swedish Winter	2fPFPP6sXqd105nk6oym5T	229880	124310 →
13	13	000U7VJzLWjyk6J8lFqnMZ	Interlude: Toilet Bowl Shawty - feat. Mike Epps	0VeraxKSlJGusnOHYGOtWq	74040	45146 →
14	14	000ULyVqUhqnAyA0Um3MEH	Simmer	23s2PFr0MCiFAf8FlpHElU	251000	109495 →
15	15	000uWezkHfg6DbUPf2eDFO	Me I Disconnect From You	3MBXzJXHFBslpPUcxNB3jn	321679	188185 →
16	16	000vBQK1sQ1ppKAEUBmYuG	All Men Are Liars	2F8e1ORfjtshM2M7nY0ckJ	204253	123377 →
17	17	000VZqvXwT0YNqKk7iG2GS	Mercy	75kQjmcR1YnlrBwVW1s4G1	256478	373921 →
18	18	000WkK6rRXx9fRbrw3L7W8	Hey Alli	2wltdutA8RJd32Wfr1O55B	202746	152840 →
19	19	000x2qE0Zl3hodeVrnJK8A	(Love Me Like Music) I'll Be Your Song	2N0AgtWbCmVoNUl2GN1opH	200626	136803 →
20	20	000xj8EilAkpZnEyOZ4jZi	Por un Gramo de Amor	2Il4CycesL00QEQYsNE5sh	305373	129249 →

artist:



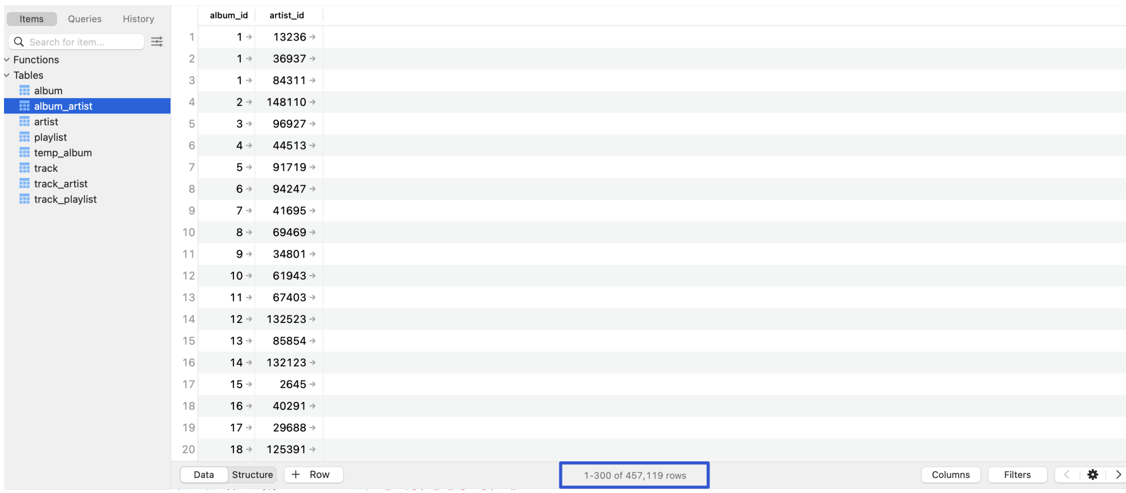
	id	artist_uri	name
1	1	0001ZVMPt41Vwzt1zsmuzp	Thyro & Yumi
2	2	000Dq0VqTZpxOP6jQMscVL	Thug Brothers
3	3	000h2XLY65iWC9u5zgcL1M	Kosmose
4	4	000UUAlAdQqkTD9sfoyQGf	Darren Gibson
5	5	0012MbXnWE3SOtQdfg8CBb	Olivia Cella
6	6	00190FC20vlUv0wXpeTf8S	KRNFX
7	7	001aJOc7CSQVo3XzoLG4DK	One Way
8	8	001c57YhYOoNNPzsSCCqfh	A Setting Sun
9	9	001NaPzWk3GKPMkuVNMdGd	Cuzzo Shay
10	10	001TRduQniM6dsJbQpMsbJ	Javier LimÃ³n
11	11	001WrTLz8s1p2sE6Ghb3DM	IZA
12	12	001zgg6v3jBCLr1Qe25cC3	Diggy Metro
13	13	0027wHZDQXpRll4ckwDGad	Disco Ensemble
14	14	002lD8rOvaV7t7IGlntSqj	Mango
15	15	002KWmvOOAdkpEcS7D43WG	Country's Finest
16	16	003BVdmFsDDbCyTKt32Bjl	Knut Bell
17	17	003f4bk13c6Q3gAUXv7dGJ	Wiener Philharmoniker
18	18	003kJVejiby1fQgWdSy3dw	Edmund Pendelton
19	19	003kZCrZhoxCckVJ2jCiJC	Peter Wolf
20	20	003Lrmd4Hy04kSf0wZm3xm	Yamaneko
21	21	003ScA1EGXbCLpKFucHXb3	Eli Kazah

playlist:



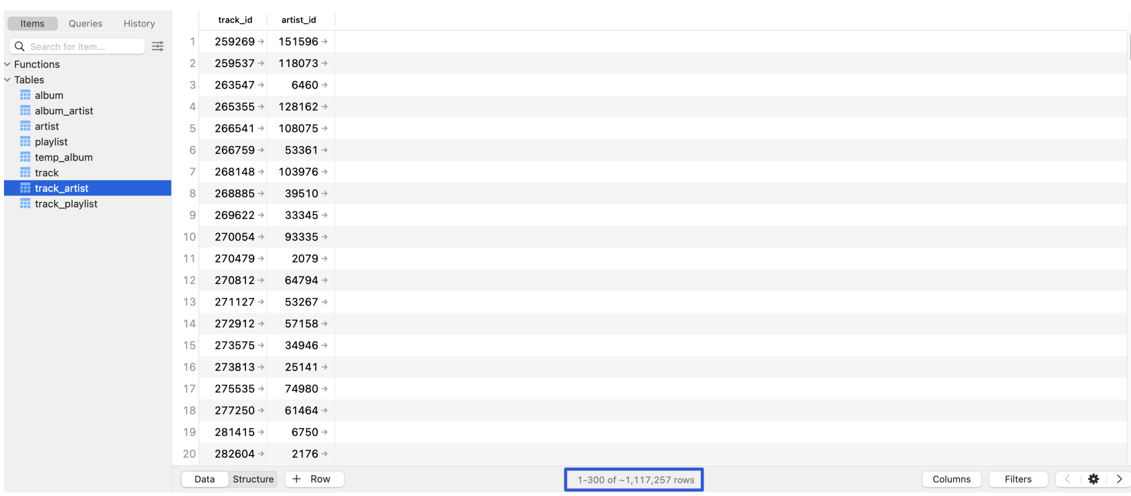
	playlist_id	name	collab	duration
1	0	Throwbacks	FALSE	11532414
2	1	Awesome Playlist	FALSE	11656470
3	2	korean	FALSE	14039958
4	3	mat	FALSE	28926058
5	4	90s	FALSE	4335282
6	5	Wedding	FALSE	19156557
7	6	I Put A Spell On You	FALSE	3408479
8	7	2017	FALSE	12674796
9	8	BOP	FALSE	9948921
10	9	old country	FALSE	4297488
11	10	abby	FALSE	16403398
12	11	VIBE	FALSE	35655891
13	12	relax	FALSE	1981349
14	13	sleep	FALSE	3039124
15	14	90's	FALSE	25837498
16	15	New Songs	FALSE	1818256
17	16	slow hands	FALSE	24695913
18	17	Mom's playlist	FALSE	17627215
19	18	SARAH	FALSE	15917464
20	19	melancholy	FALSE	21627256

album_artist:



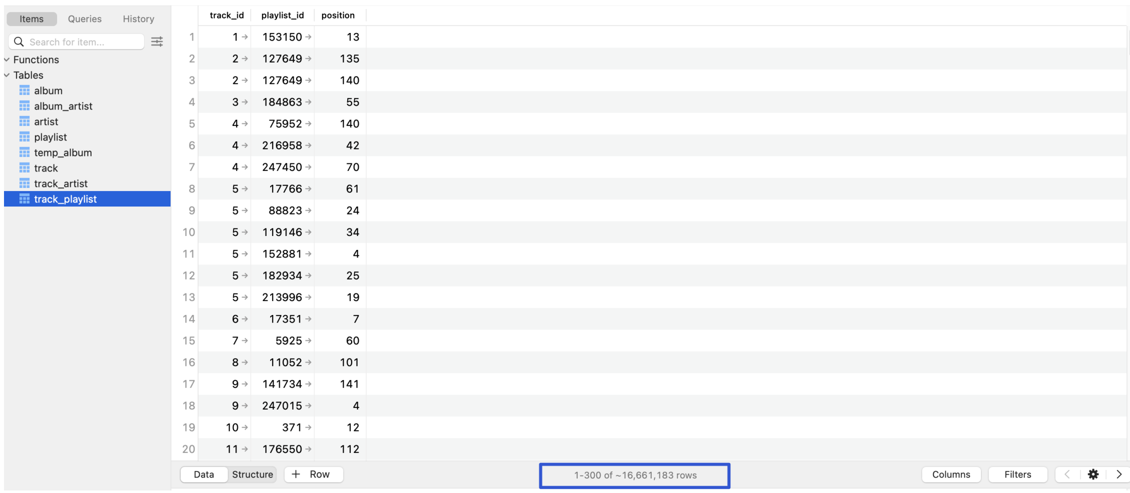
	album_id	artist_id
1	1	13236
2	1	36937
3	1	84311
4	2	148110
5	3	96927
6	4	44513
7	5	91719
8	6	94247
9	7	41695
10	8	69469
11	9	34801
12	10	61943
13	11	67403
14	12	132523
15	13	85854
16	14	132123
17	15	2645
18	16	40291
19	17	29688
20	18	125391

track_artist:



	track_id	artist_id
1	259269	151596
2	259537	118073
3	263547	6460
4	265355	128162
5	266541	108075
6	266759	53361
7	268148	103976
8	268885	39510
9	269622	33345
10	270054	93335
11	270479	2079
12	270812	64794
13	271127	53267
14	272912	57158
15	273575	34946
16	273813	25141
17	275535	74980
18	277250	61464
19	281415	6750
20	282604	2176

track_playlist:



	track_id	playlist_id	position
1	1	153150	13
2	2	127649	135
3	2	127649	140
4	3	184863	55
5	4	75952	140
6	4	216958	42
7	4	247450	70
8	5	17766	61
9	5	88823	24
10	5	119146	34
11	5	152881	4
12	5	182934	25
13	5	213996	19
14	6	17351	7
15	7	5925	60
16	8	11052	101
17	9	141734	141
18	9	247015	4
19	10	371	12
20	11	176550	112

Note that all of these images are also included in the submission for easier viewing (stored in an image folder).