Maastricht University

DSAI - Data Science and Artificial Intelligence

DB Project - Report - KEN2110

Find Your Track

Part II

Author

Detry Clément - i6194609 (Group 32)

Paul-Henri Spaaklaan 1, 6229 EN Maastricht, Pays-Bas

October 7, 2020

Contents

1 Problem description

During this course, students were asked to realise a project about a case study for which they should build a database system. This report will tell you more about the development of it through several steps.

2 Find Your Track database

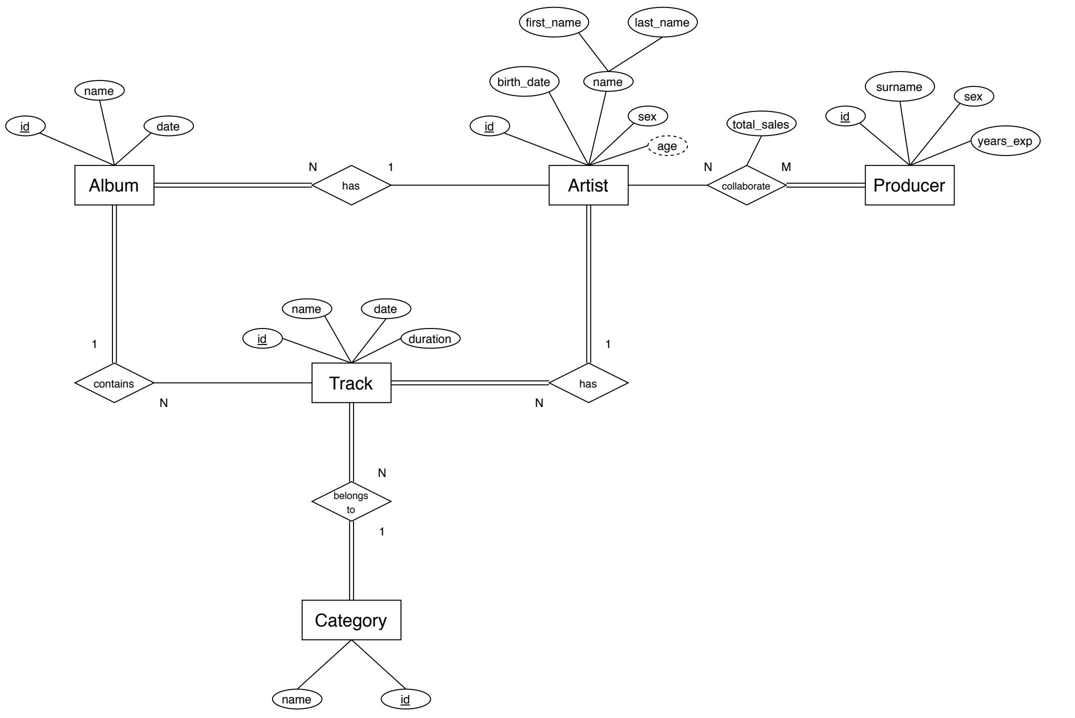
It has certainly already happened to many of you to want to create a playlist of music that suits your mood. Whether it's for a certain style of music, a particular artist or a specific period of time, you'll find what you're looking for. Because this database system is made for those who wish to escape through music or to become blind-test professionals.

Being passionate about music for several years now, it was important for me to be able to provide a database that could help many people to discover the music they want.

In order to solve this problem in the widest possible sense, many nice features had to be thought of to make the database as useful as possible. It must be able to take into account, for example, the possibility of being able to find a series of music titles from a pop artist. But also to be capable of identifying several artists who have collaborated with a producer you particularly like. These are just a few examples that can be done using the database.

3 Entity Relationship model

3.1 Model



3.2 Model description

To make the ER diagram easier for everyone to understand, a description of the different entities and relationships is given in the next two sections.

3.2.1 Entities

- Entity **Artist** has an id as its primary key, a composite name attribute made up of a first name and a last name, a sex gender and a birth date that creates a derived attribute age.
- Entity **Track** has an id as its primary key, a name, a date that corresponds to its release date and a duration attribute to give the information about the music track length.
- Entity **Album** has an id as its primary key, a name and a date that corresponds to its release date.
- Entity **Producer** has an id as its primary key, a surname, a sex gender and an attribute that defines the producer's years of experience in the domain.
- Entity **Category** has an id as its primary key and a name.

3.2.2 Relationships

- Artist $< - >$ Producer
  An artist can collaborate with several producers. A producer can collaborate with different artists as well and has a total participation in the artist entity since a producer need at least one artist to work with.
- Artist $< - >$ Album
  An artist can have more than one album but he does not have a total participation with the album entity since an artist can exist without having any official album but only music tracks. An album can only have one artist and has a total participation through the artist entity since it cannot exist without an artist that creates it.
- Artist $< - >$ Track
  An artist can have several music tracks and has a total participation since he needs to have the track entity to be considered. A track can only have one artist and also has a total participation since it cannot exist without an artist.
- Album $< - >$ Track
  An album must contain more than one track and has a total participation towards the track entity because without any of it, an album cannot exist. Several music tracks can be contained in the same album but the participation is not total since a track can be created without having necessarily an album.

- Track $< - >$ Category
  More than one track can belong to the same category but one song cannot have two categories. The participation is total because a track needs to have a category to be considered in the database. A category can belong to several music tracks and the participation is total as well since a category cannot be added to the database if no track is inside of it.

4 Mapping ER to Relational Model

4.1 Process

4.1.1 Strong entity types

- Artist
- Track
- Album
- Producer
- Category

4.1.2 Weak entity types

No weak entity type in the ER model.

4.1.3 1:1 relations

No 1:1 relation in the ER model.

4.1.4 1:N relations

- Artist (1) $- >$ (N) Album
  The foreign key is artist_id in Album pointing to primary key in Artist.
- Artist (1) $- >$ (N) Track
  The foreign key is artist_id in Track pointing to primary key in Artist.
- Album (1) $- >$ (N) Track
  The foreign key is album_id in Track pointing to primary key in Artist.
- Category (1) $- >$ (N) Track
  The foreign key is category_id in Track pointing to primary key in Category.

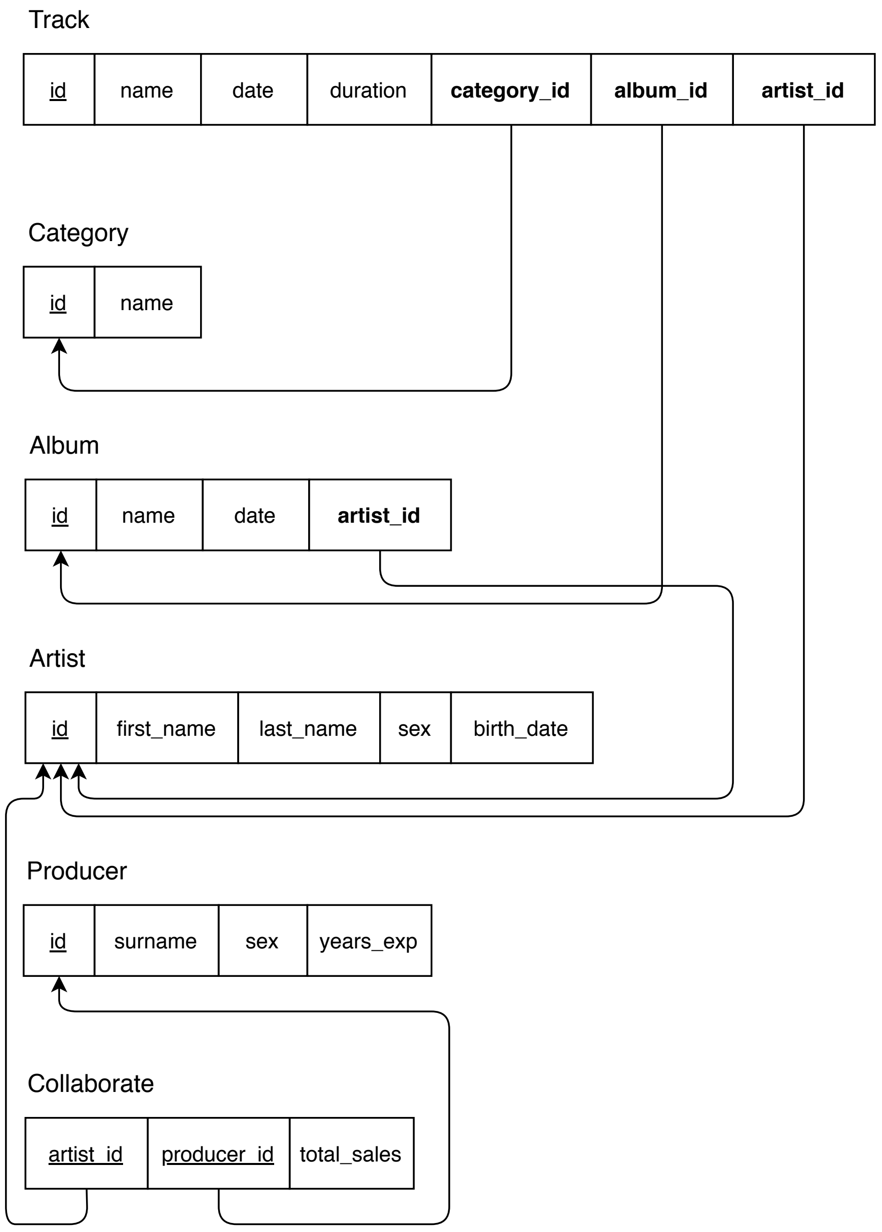
4.1.5 N:M relations

- Artist (N) − > (M) Producer
  Foreign keys are artist_id and producer_id in the new entity called Collaborate, respectively pointing to primary keys in Artist and Producer.

5 Relational Model

5.1 Model

The Relational Model with the latest updates that were made in the process part is now complete.



5.2 Database Normalization

Based on the decision concerning the database design, it seems that the database is normalize to a 3NF type.

1NF

- All the attributes are single-valued in the design. Because any column of any tables contains more than 1 value.
- Each column contain values of the same type.
- Each column have a unique name.

  => So, the database is 1NF.

2NF

- There is no partial dependency in the database design. There exists no attribute in any tables that only depends on one of the two composite primary keys. For example, the Collaborate entity is defined by a composite primary key and one attribute. This attribute total_sales depends on both primary keys and not only one.

  => So, the database is 2NF.

3NF

- There is no transitive dependency such as P $->$ Q and Q $->$ R, then P $->$ R. Because there exists no attribute that depends on another attribute that is no the primary one in the table.

  => So, the database is 3NF.

6 Implementation

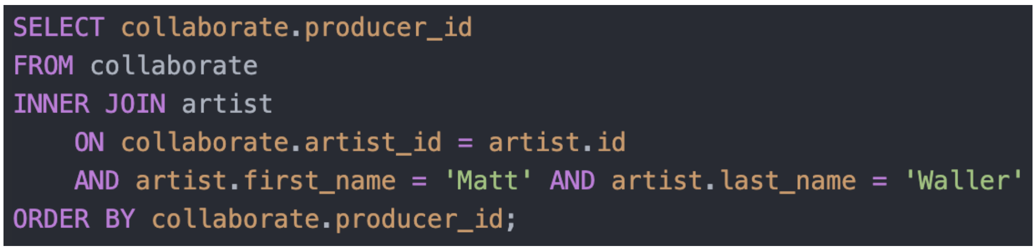
6.1 Database

All the tables with data, queries and views have been implemented using PostgreSQL. As requested in the instructions, all tables have at least 10 rows.

The majority of the attributes that needs to be are NOT NULL values. It allows the database to contain as much relevant information as possible. Foreign keys attributes have all be configured using a delete parameter which ensures that all rows related to the element in question are cascaded out of the database. For example, if an artist is deleted from the table, all the rows related to this artist will be deleted at the same time (all his music tracks, albums, collaborations, etc). This allows the database to remain clean, without having irrelevant information in there.

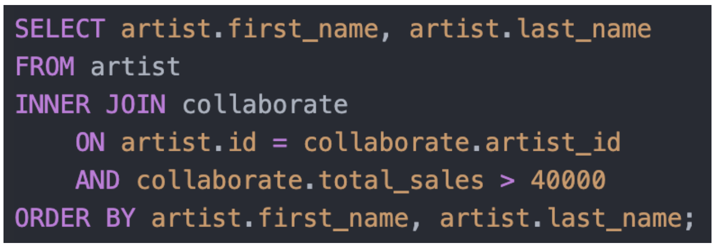
Implementations about the database, queries and views can be found in the files called 'database.sql' and 'queries_views.sql'.

6.2 Queries



```
SELECT collaborate.producer_id
FROM collaborate
INNER JOIN artist
    ON collaborate.artist_id = artist.id
    AND artist.first_name = 'Matt' AND artist.last_name = 'Waller'
ORDER BY collaborate.producer_id;
```

The first query example retrieves all the *producer_id* who have already collaborate with an *artist.first_name* and *artist.last_name* (in this special case, Matt Waller). To find this information, tables collaborate and artist need to be joined. When the condition that both *artist_id* from both tables are equal and the *artist.first_name* and *artist.last_name* correspond to the right artist, result can be selected.



```
SELECT artist.first_name, artist.last_name
FROM artist
INNER JOIN collaborate
    ON artist.id = collaborate.artist_id
    AND collaborate.total_sales > 40000
ORDER BY artist.first_name, artist.last_name;
```

The second example is finding the *artist.first_name* and the *artist.last_name* of artists that makes over 40'000 *total_sales* with a single collaboration. This query is joining two tables, the artist and the collaboration one. In this case, two conditions must match together, both *artist_id* need to be equal and the *total_sales* should be higher than the given number.

```sql
SELECT album.name, album.date
FROM album
INNER JOIN artist
    ON album.artist_id = artist.id
    AND (artist.birth_date LIKE '%1962%' OR artist.birth_date LIKE '%1989%')
ORDER BY album.date;
```

The third query example is about finding the *album.name* and the *album.date* of artist that was born in 1962 and 1989. This time, both album and artist tables must be joined. When the *artist_id* that match the condition can be get from the artist table, the data concerning album can be returned.

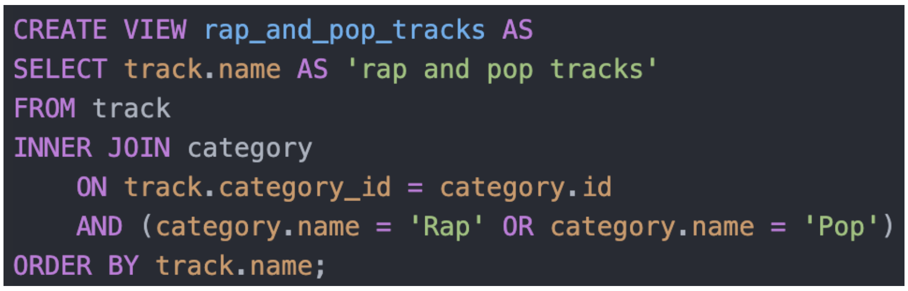
```sql
SELECT track.name, track.category_id
FROM track
INNER JOIN collaborate
    ON track.artist_id = collaborate.artist_id
INNER JOIN producer
    ON collaborate.producer_id = producer.id
    AND producer.surname = 'Stanislaw'
ORDER BY track.name;
```

The final query example allows to find all the *track.name* and corresponding *track.category_id* which are product by a given producer (in this particular case, Stanislaw). This one joins three different tables to make the request possible: track, collaborate and producer. First, the condition that both *artist_id* from track and collaborate tables must be respected. Then, when both *producer_id* match with the condition that the *producer.surname* needs to equal a certain value, *track.name* can easily be obtained.

6.3 Views

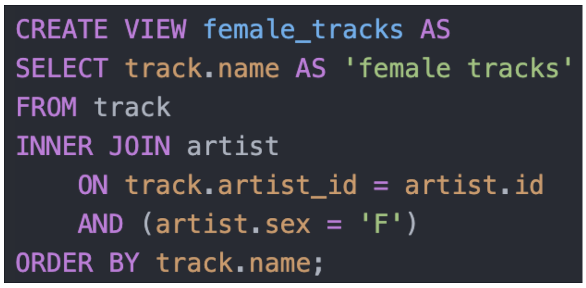
```sql
CREATE VIEW tracks_shorter_3_minutes AS
SELECT track.name AS 'tracks < 3 min', track.duration
FROM track
WHERE track.duration < 3.00
ORDER BY track.duration;
```

The first constructed view can be useful to use if the user only wants to have a quick look at all the database music tracks that are shorter than 3 minutes. It is possible that a listener prefers to have access to sounds that are not too long for him. All the tracks are then ordered by their duration.



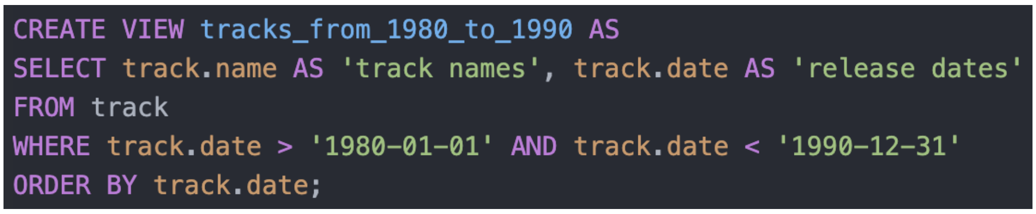
```sql
CREATE VIEW rap_and_pop_tracks AS
SELECT track.name AS 'rap and pop tracks'
FROM track
INNER JOIN category
    ON track.category_id = category.id
    AND (category.name = 'Rap' OR category.name = 'Pop')
ORDER BY track.name;
```

The second view example consists in only viewing the music names that comes from one or more categories the user wishes to listen to. In this predefined example, rap and pop are the desired types of categories. After returning the list, the tracks are sorted by name order which simplifies the classification.



```sql
CREATE VIEW female_tracks AS
SELECT track.name AS 'female tracks'
FROM track
INNER JOIN artist
    ON track.artist_id = artist.id
    AND (artist.sex = 'F')
ORDER BY track.name;
```

The third view is made up of a music list only composed by women. It may be useful for a user of the database to only have access to this kind of information if, for example, he prefers the voices of women to those of men. To make it easier, music names are then ordered by their respective names.



```sql
CREATE VIEW tracks_from_1980_to_1990 AS
SELECT track.name AS 'track names', track.date AS 'release dates'
FROM track
WHERE track.date > '1980-01-01' AND track.date < '1990-12-31'
ORDER BY track.date;
```

The fourth and final view has been designed to help potential users obtain access to the names of tracks that have been released between two predefined years. This view can be useful if we take the example of a user who wants to organize an 80's event and wishes to compose a playlist in the theme quickly. This time, tracks are arranged from oldest to newest in the given period of time.

7 Conclusion

The database is now finished and functional. It does indeed answer the problem stated at the beginning of the report and what it should look like throughout its creation.
Of course it can be improved with other parameters and contain much more data but for experimental use it is still sufficient. Hopefully it will prove its worth and will be of interest to many of you.