Report

Assignment 3

Author: Elise ANJEL & Joakim ROSENQVIST
Semester: HT2019
Area: Computer Science
Course code: 2DV513

Contents

1 Idea

We want to create a resource where music lovers can find information about artists, labels and releases. To do this, we will have a database containing a set of releases, and through a web interface the user can search in different categories or filter by clicking on for example an artist's name to get a filtered list of that artist's releases.

Our goal is to have a web back-end connected to the database, and dynamically generate pages for the user. Since we can not host the site on a big compute cluster, we will limit the size of the database. Data will be gathered from Discogs in XML format and then parsed and inserted into the database, according to our schema. The web-site will be driven by Flask, a Python framework.

2 Logical model

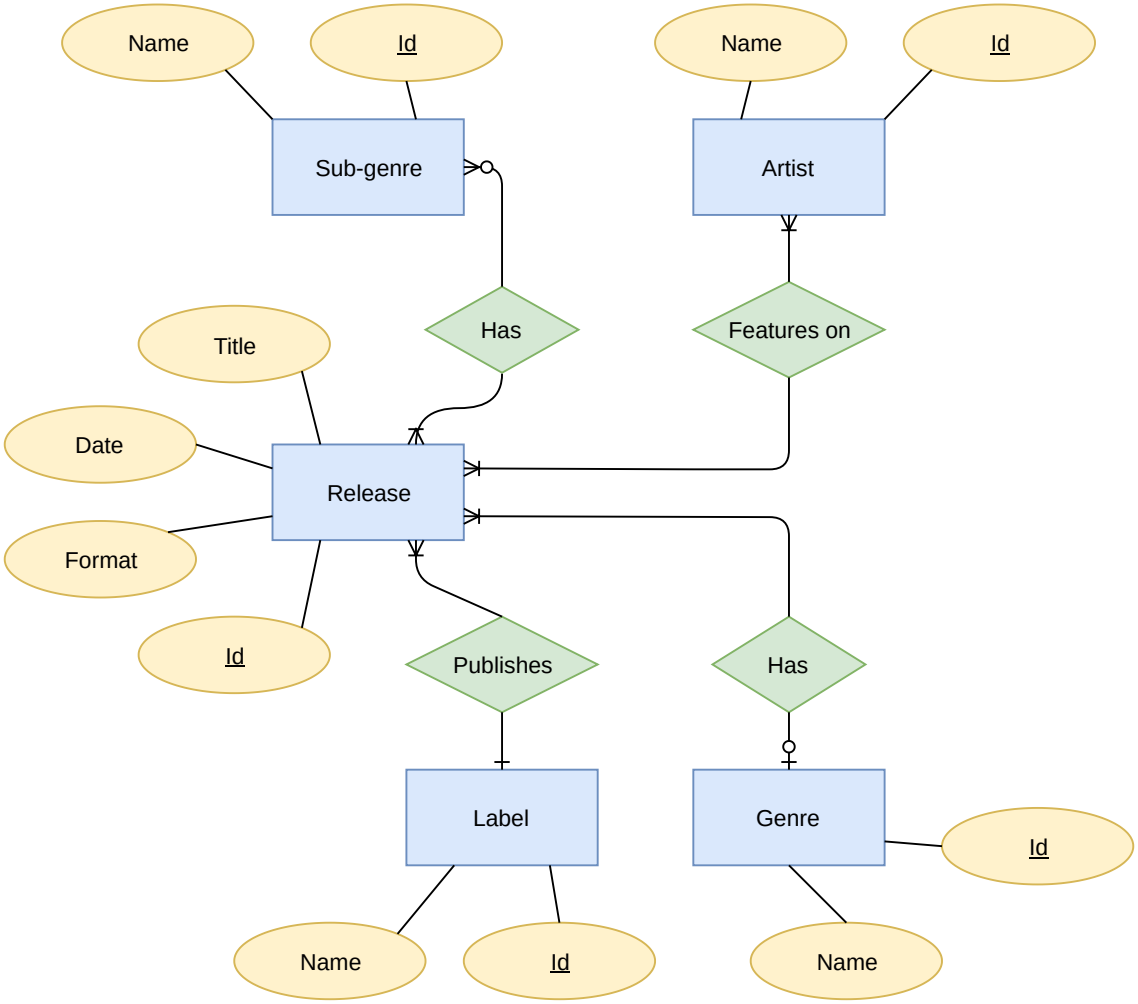

Figure 1: E/R Diagram

The model is based on an initial analysis of the Discogs data-set we used. To avoid duplication on artists, genres and sub-genres (styles), we decided to make those into separate entities. There was more data available that we could have modeled, but we decided these parts were most essential for our application. Simple information like release dates and formats were kept as attributes of the Release entity.

3 Design in SQL

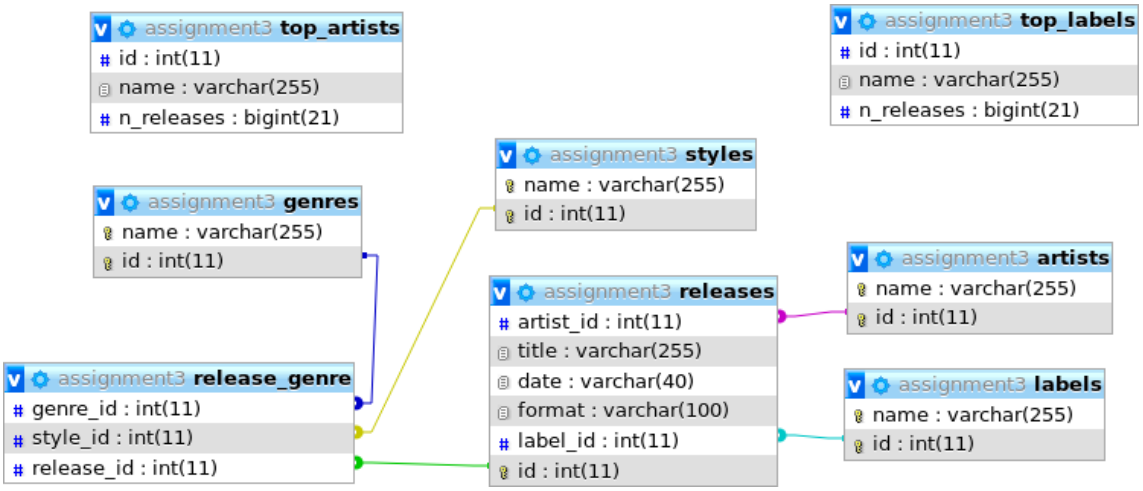

Figure 2: Diagram of SQL schema

As seen in Figure 2, we translated our model by putting the entities in separate tables and relating them through foreign keys. Every release has an artist and a label, and references the id column of the corresponding tables. With genre and style (sub-genre) kept as separate entities we needed a way to i.e. search for releases of a certain genre or what styles appear on a release of a certain genre. To do this, we used a separate table only containing foreign keys to link the information together.

Date attribute couldn't be of DATE type, since the data-set contains a variety of formats, like combination of a string and year.

We also added to views to represent top 5 artists with most releases and top 5 labels with most releases.

4 SQL queries

Where selecting from 'table', it's a variable used in the Python code and could be any table in the database based on choices in the user interface.

Get specific tuple using ID from each table. This query is done whenever we want to display information about i.e. a certain artist or label:

```
SELECT * FROM table WHERE id = 'id';
```

Get all content from each table, limit to 100. This query is used for listing the contents of a table. The limit is there to make sure we don't keep more data in memory than necessary on the web server. For anything more specific, the user has to use the search:

```
SELECT * FROM table LIMIT 100;
```

Get all releases for a specific artist, join with label to get labels' names:

```
SELECT artist_id, title, date, format, label_id,
releases.id AS release_id, name AS label_name
FROM releases JOIN labels ON releases.label_id = labels.id
WHERE releases.artist_id = 'artist_id';
```

Get all releases for a specific label, join with artists to get artists' names: label_id is the variable:

```
SELECT artist_id, title, date, format, label_id,
releases.id AS release_id, artists.name AS artist_name
FROM releases JOIN artists ON releases.artist_id =
artists.id WHERE releases.label_id = 'label_id';
```

Get sub-genres for a specific genre. Genres and styles are related through the release_genre table, therefore it has to be joined.

```
SELECT DISTINCT styles.name AS style_name, styles.id
AS style_id FROM styles JOIN release_genre ON
release_genre.style_id = styles.id WHERE
release_genre.genre_id = 'genre_id';
```

Get all releases for a specific genre, join with artists and labels to get the names and IDs. genre_id

```
SELECT DISTINCT artist_id, title, date, format, label_id, releases.id
AS release_id, artists.name AS artist_name, labels.name as label_name
FROM releases
JOIN artists ON releases.artist_id = artists.id
JOIN release_genre ON release_genre.release_id = releases.id
JOIN labels ON releases.label_id = labels.id
WHERE release_genre.genre_id = 'genre_id';
```

Get all releases for a specific subgenre, join with artists and labels to get the names and IDs. style_id is the ID to query for:

```
SELECT DISTINCT artist_id, title, date, format, label_id,
releases.id AS release_id, artists.name AS artist_name,
labels.name as label_name
FROM releases
JOIN artists ON releases.artist_id = artists.id
JOIN release_genre ON release_genre.release_id = releases.id
JOIN labels ON releases.label_id = labels.id
WHERE release_genre.style_id = 'style_id';
```

Search in artists, labels or genre tables. Since we have a limit on our listings, a search query is available to the user, to find anything more specific:

```
SELECT *
FROM db_table
WHERE name LIKE 'search_criteria';
```

Search in releases table. It's joined with artists to get artist name and id:

```
SELECT DISTINCT artist_id, title, date, format,
releases.id AS release_id, artists.name AS artist_name
FROM releases
JOIN artists ON releases.artist_id = artists.id
WHERE title LIKE 'search_criteria';
```

Create a view that displays the top 5 artists with most releases. This is used on the front page:

```
CREATE VIEW top_artists AS
SELECT artists.id, artists.name, COUNT(releases.id) AS n_releases
FROM releases
JOIN artists ON releases.artist_id = artists.id
GROUP BY artists.id
ORDER BY n_releases
DESC LIMIT 5;
```

Create a view that displays the top 5 labels with most releases. This is used on the front page.

```
CREATE VIEW top_labels AS
SELECT labels.id, labels.name, COUNT(releases.id) AS n_releases
FROM releases
JOIN labels ON releases.label_id = labels.id
GROUP BY releases.label_id
ORDER BY n_releases
DESC LIMIT 5;
```

The views are used by simply selecting all (*) from the views.

5 Implementation

We made an implementation with MariaDB as our DBMS and the user interface is a web application made with Python3/Flask. Here are some notes on how to get it running.

The data we used is `https://discogs-data.s3-us-west-2.amazonaws.com/data/2008/discogs_20080309_releases.xml.gz`

Instructions database:
A dump of the SQL database is attached in zipped form in the db folder. Possibly it is necessary to have a database named *assignment3* before trying to import the data.

Installation/instructions Python3/Flask:
Create pipenv, install plugins specified in Pipfile.
Run pipenv shell.
Export FLASK_APP environment variable:
export FLASK_APP=main_application.py (use set instead of export on Windows)
Run using 'flask run' (run in /src/ folder).

Note: The default user and password used to connect to the database are 'user' and 'CorrectHorseBatteryStaple', they connect to a MariaDB or MySQL server running on localhost. This can be changed in the source-code file '/src/app/routes.py'.

6 Supplemental video

The video shows the implementation in the works. Certain actions are performed then the queries run in the background are displayed from the report. For more information about the quires, see section Section 4: SQL queries and the source-code file '/src/app/routes.py' for the actual implementation.
Link: `https://youtu.be/bSjhgRSzdjk`