# A database for a music streaming service

We want to design a database for a new music streaming platform. The platform will offer a variety of audio contents on a subscription-based business model.

Customers are expected to register to the service by providing their full name, address, and e-mail. After the registration, users can listen to the free content offered by the service, but they will need to pay a subscription fee for any other content. To let the users pay for the subscriptions, the service also needs to keep track of payment methods.

The service will offer music albums and singles from music production companies, podcasts, audiobooks, radio, and live-streaming, as well as user-created audio tracks. Contents can be owned by companies or users of the platform. The owners of contents must set the price that is required to access their individual contents. Contents can be accessed for free, have a one-time individual fee, or be part of a subscription package. When users buy a subscription package for a period of time, they can access every content in that package until their subscription is expired.

Inside the platform, users could rate how much they like a content from 1 to 5, only if they already listened to it. Users can also create playlists and share them publicly or keep them private. The service should keep track of the content users listen to, when, and from which device. In this way, the service can recommend new content based on the genre one likes, current time of day, or any other information that may seem relevant to derive users' preferences.

On this system, we also define the following use cases, which set usage scenarios that the database design must accommodate:

- Users can list the audio contents of a given type they like the most, by setting a minimum rating. For example, users can list all the music singles for which they provided a rating of at least 4.
- User can visualize new content on the platform, given some preferences like type and/or genre. For example, the new classical music albums added to the platform during the last month.
- Administrators of the service can list the email addresses of users who didn't renew their subscriptions, along with which types of subscriptions were previously owned by them.
- The service can recommend content that users may like based on their profile. For example, the service could propose content that a user never listened to, but at least some other number of users rated as at least 4, and it is of the most listened genre for that user.

## Exam requirements

The project is intended for individual students, not for team work. Therefore, each student is expected to work on the project independently.

The project is articulated on a number of tasks, all to be completed at your best. At minimum, to pass the exam, one has to complete all tasks, and the final grade will depend on the quality of the submitted solutions. Submitting perfect solutions to a subset of the tasks, while not providing any solution for the remaining tasks, will not suffice to pass the exam.

The quality of solutions can range from simplistic to more sophisticated (using all the statements and more advanced functionalities illustrated during the lectures). Obviously, more sophisticated solutions will lead to a better grade.

Notice that the project requirements do not describe explicitly all the statements and the functionalities that you might use.

The solutions must not involve any external programming language or tool.

## Tasks

1. Design an Entity-Relationship Diagram using Chen's notation (Chapter 3). Additionally, provide a list of the entity types and relationship types you identified, along with the requirement(s) or use case(s) that led you to those design decisions.
2. Map the ER Diagram into a relational model by following the procedure presented in Chapter 9. For each step, write how you applied it and to which elements. Identify primary keys, foreign keys, and attributes' data domains. Provide the final relational model.
3. Create a SQL schema from the relational model, as it is shown in Chapter 6. The SQL schema must be compliant with MySQL.
4. Populate your database as much as it is reasonable. Also provide the relative CSV files and the commands to load the data in the DB.
5. Implement the use cases, by providing, for each use case, the SQL statement(s) that implements it. The SQL statements must be compliant with MySQL.

## Required deliverables

(also summarized in the report template)

1. The ER Diagram; [up to 7pts]
2. Mapping procedure from the ER Diagram to the relational model; [up to 6pts]
3. The relational model; [up to 6pts]
4. SQL schema from task 3; [up to 2pts]
5. CSV file and relative statements to populate the DB; [up to 2pts]
6. SQL statements from task 5; [up to 5pts]
7. A report that documents the design process, problems encountered, and the rationale behind the provided solutions. [up to 2pts]

**To be admitted to the exam, the deliverables must be presented by e-mail at giovanni.stilo@univaq.it and michele.tucci@univaq.it 3 days before the exam date.**