ENTERPRISE ICT ARCHITECTURES PROJECT

Author(s): **Carbone Emanuele (10726300)**
Carli Federico (10713446)
Ferrazzano Andrea (10703279)
Gorini Marco (10710545)
Pauselli Tommaso (10797253)

Group Number: **6**

Academic Year: 2023-2024

Contents

1 | First Project Delivery

1.1. Introduction

“Consider the problem of specifying the data design for a music application. Users enrol into the platform by providing unique mail, username, and password. Their registration date is stored, too. Users can be premium users if they pay a monthly fee. Users listen to podcasts and songs. Songs are grouped into categories, described by a name and a description, and are played by various artists while various authors produce podcasts. Podcasts are described by a unique identifier, title, duration (in seconds), and publication date. Songs include all the attributes of podcasts, including the text of the song. The app tracks who listens to what, when, and whether they listened to the whole song or podcast. Users can create playlists that aggregate songs. These are described by a unique identifier, title, creation date, and whether they are public. The current date is also stored whenever a song is added to a playlist. Users can also tag a song or a playlist as a favourite. The current date is also stored whenever a song or a playlist is tagged as a favourite.”

1.2. ER Model

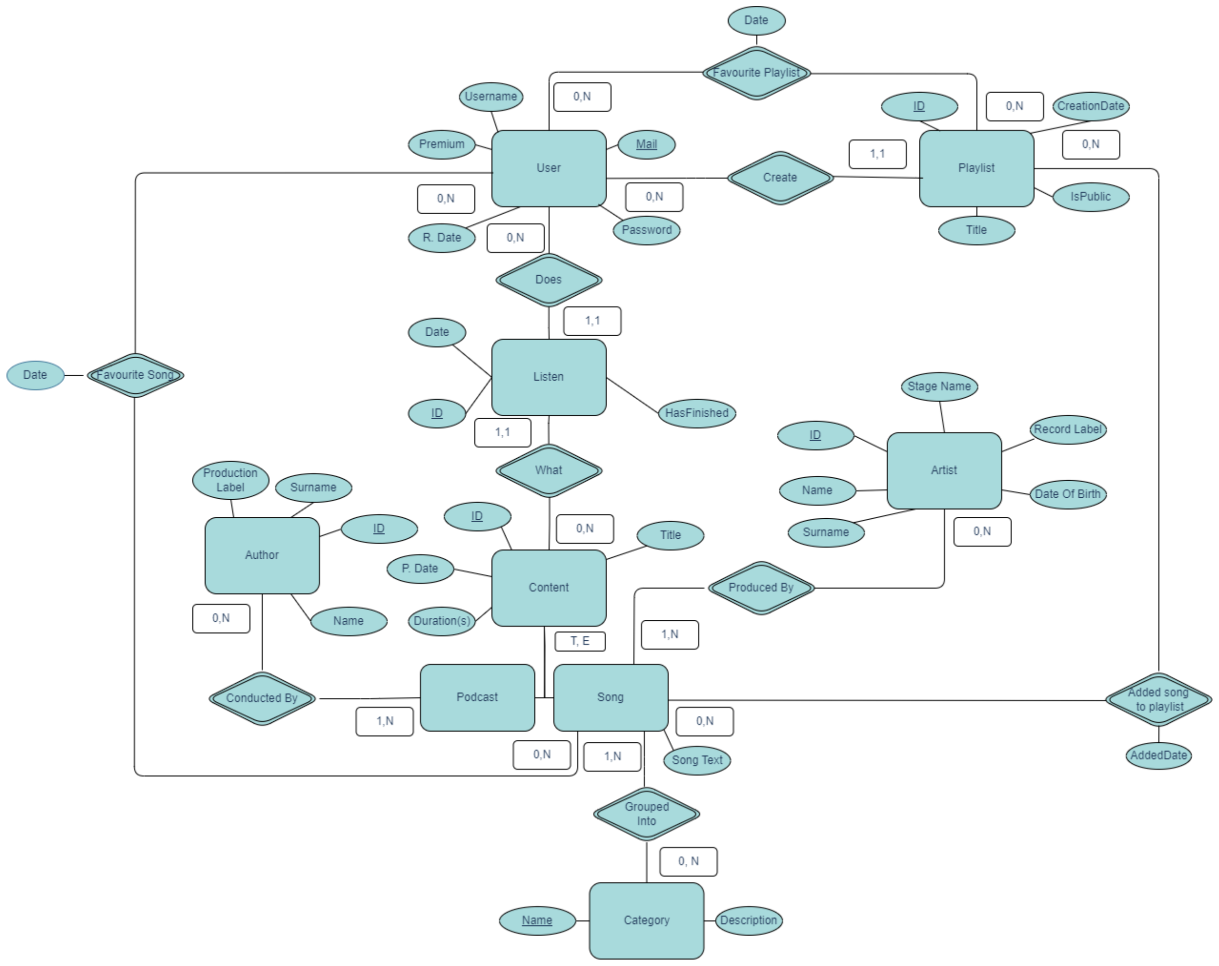


Figure 1.1: E/R Model of the Database

Note: to read the cardinality, you have to read it from the closest entity toward the opposite one, f.ex.: "User" can "create" from 0 to N "Playlist", and a "Playlist" can be "created" by at least 1 and not more than 1 "User".
In the Appendix (page 25) there is the link to the draw.io file.

The model has the following entities:

- **User**:we decided to create this entity in order to represent the user's profile and his information.It has 5 attributes and we choose the mail attribute as the primary key because it is a unique information.The "Username" attribute means the nickname with which the user is represented in the platform. Premium is a Boolean attribute that indicates whether the user has paid for the premium subscription and R.Date indicates the date the user registered. The password attribute contains the account's password;

- **Listen**: we decided to represent every single listen as an entity because in this way is possible to store multiple listening sessions of the same song or podcast by the same user using an incremental id. We could have only created a relation between the user and content by adding to the bridge table an attribute idListen and using it as a part of the key formed by (idUser,idSong/idPodcast,idListen). However, we preferred to give importance to listening as an instance because it has several attributes such as the streaming date and the Boolean attribute “HasFinished” that record whether a song has been listened to in its entirety or not. In addition to that if we had used the relation described before we would have had the same table with the only difference that the key would have been formed by the id plus the two foreign keys from User and Content(Podcast or Song), in this case, the id attribute would have indicated the number of the stream for each user;
- **Content**: we think that the best way to uniquely identify it is through an auto-incremented "id" (because there could be songs with the same titles), and it has the attributes "title", "duration"(as Duration(s) in the model) and "publicationDate"(as P.Date in the model);
  There is an ISA-hierarchy(Total: from the text we assume that on this information system the content creators are ONLY divided in artists and authors; Exclusive: from the text we assume that an artist can be only an artist, and the same for the author) on the following two sub-entities:
  - **Podcast**: it has only the same attributes of "Content";
  - **Song**: it also has the same attributes of "Content" and the attribute "text";
- **Playlist** : the “id” attribute is the primary key and identifies the playlist, the Boolean attribute “isPublic” indicates whether the playlist is public or not. ”Title” indicates the name given by the creator and can’t be a key because more playlists might have the same name;
- **Artist** : we think that the best way to uniquely identify it is through an auto-incremented "id", and it has the attributes "name", surname, "stageName" (in case it could have a different name to introduce to the audience), "dateOfBirth" and "recordLabel";
- **Author**: we think that the best way to uniquely identify it is through an auto-incremented "id", and it has the attribute "name", "surname", and "productionLabel";

- **Category**: this entity represents the genre of a song. We create an entity instead of considering it as an attribute in the song entity because it has the “description” attribute and,as seen before in the creator case, a song can belong to more categories at the same time so we would have needed a table anyway in order to represent the multivalued attribute;

It is not clear from the text that it is mandatory to create the Artist and Author entities, since they are not given importance as entities, but we wanted to enhance them as such for the completeness and logical cleanliness of the Database, furthermore if it is a song isartist will be equal to 1 and isauthor must be equal to 0. It could have been done in two other ways:

- insert a "Creators" attribute of type VARCHAR(255) into "Content", and concatenate the names of the artists into a string, all separated by commas;
- simply create a single "Creator" entity connected to "Content", put identifying attributes of our choice and distinguish between those who produce podcasts and songs with the booleans "isAuthor", "isArtist";

The model has the following relations:

- **Create**: it’s between "User" and "Playlist", the first is able to create from 0 to infinite possible playlists, meanwhile a playlist can be created by just one and only one user.
- **Does**: it’s between "User" and "Listen", user can have from 0 to N listens while a listen is uniquely associated to a user.The (1,1) cardinality expresses the fact that a listen can’t exist alone without the activity of a user.
- **What**: it’s between "Listen" and "Content", a listen is related to only one content while content can be streamed in more listens. The (1,1) cardinality expresses the fact that a listen can’t exist without content.
- **FavouritePlaylist**: it’s between "User" and "Playlist", the first is able to consider from 0 to infinite possible playlists as favourite, meanwhile a playlist can be considered as favourite by nobody until infinite users; the relation has an attribute “date” that indicates the date on which the user has added the song as a favourite. In this way, we don’t value the presence of multiple additions to the same playlist by the same user. However, this can be stored by adding a dateTime or an id attribute and using it as a part of the key of the bridge table between the user and the playlist.
- **FavouriteSong**: it’s between "User" and "Song", the first is able to consider from

0 to infinite possible playlists as favourite, meanwhile a song can be considered as favourite by nobody until infinite users; the relation has an attribute “date” that indicates the date on which the user has added the song as a favourite. Note that a user can add a song as a favourite even if he has never listened to it. In this way, we don’t value the presence of multiple selections of the same song as a favourite one by the same user. However, this can be stored by adding a dateTime or an id attribute and using it as a part of the key of the bridge table between the user and the song. We thought there could have been the possibility to represent the relation by only adding a “isFavourite” attribute to the listen entity with the constraint that the idSong attribute on “listen” isn’t NULL (due to the constraint expressed in the logical model paragraph we know that in this case, the idPodcast would be NULL). The date would be the same as the listen’s. However, in this way, we wouldn’t be able to save as favourite songs those that we never listen to.

- **AddedSongToPlaylist**: it’s between "Playlist" and "Song", the first is able to hold from 0 to infinite possible songs as favourite, meanwhile a song can be added to 0 until infinite playlists; the relation has the attribute “date” in order to store the addition date. From the model, we don’t know who added a song to a playlist but we assume that only the creator can add songs to the playlist he created. So to know who added the song we need to know who created it. In order to have a scenario in which every user is able to add a song to a public playlist we should add a relation between the user and the song and another one between the song and playlist.

- **ConductedBy**:it’s between "Author" and "Podcast", the first one can conduct 0 to infinite possible podcasts(we are assuming that an author can be registered on the platform as an author without publishing any content), meanwhile a podcast can be conducted by 1 to infinite possible authors.

- **ProducedBy**: it’s between "Artist" and "Song", a song can have from 1 to N artists while an artist can have produced from 0 to N songs. It explains the fact that an artist can be in the database even if he has never made a song.

- **GroupedInto**: it’s between "Song" and "Category", the first one can be inside from 1 to infinite possible categories, meanwhile a category can have from 0 to infinite songs (because on the platform there could be songs that are not associated with the categories already existing).

1.3. Logical Model

Legenda:
key: Primary Key
key: Foreign Key

USER (**mail**, username, password, registrationDate, isPremium)

LISTEN (**id**, mailUser, idSong, idPodcast, date, finished)

PODCAST (**id**, title, duration, publicationDate)

SONG (**id**, title, duration, publicationDate, text)

PRODUCED_BY(**idArtist**, **idSong**)

CONDUCTED_BY(**idAuthor**, **idPodcast**)

PLAYLIST (**id**, title, creationDate, isPublic, mailUSer)

FAVOURITE_SONG (**idSong**, **mailUser**, favouriteDate)

FAVOURITE_PLAYLIST(**idPlaylist**, **mailUser**, favouriteDate)

ARTIST (**id**, stageName, name, surname, dateOfBirth, recordLabel)

AUTHOR (**id**, name, surname, productionLabel)

ADDED_SONG_TO_PLAYLIST (**idSong**, **idPlaylist**, addedDate)

GROUPED_INTO (**idSong**, **nameCategory**)

CATEGORY (**name**, description)

- **User**: it is uniquely identified by the primary key mail, and it has the attributes username, password, registrationDate and isPremium;
- **Listen**: it has an auto-incremented id as primary key, and because this entity has relations with Content and Playlist as "1 to many", it needs to get as foreign keys the primary keys of the other entities: mailUser, (because Content has 2 sub-entities Song and Podcast, it's necessary to get their primary keys) idSong and idPodcast, and it has the attributes date and finished;
- **Podcast**: it has an auto-incremented id as primary key, and it has the attributes title, duration and publicationDate;

- **Song**: it has an auto-incremented id as primary key, and it has the attributes title, duration, publicationDate and text;
- **Produced_By**: because Produced_By is a "many to many" relation between Artist and Song, it has been necessary to assign both their primary keys idArtist and idSong to it as primary and foreign keys;
- **Conducted_By**: because Conducted_By is a "many to many" relation between Author and Podcast, it has been necessary to assign both their primary keys idAuthor and idPodcast to it as primary and foreign keys;
- **Playlist**: it has an auto-incremented id as primary key, and because this entity has a relation with user as "1 to many", it needs to get as foreign key the primary keys of the other entity id est mailUser, and it has the attributes title, creationDate and isPublic;
- **Favourite_Song**: because Favourite_Song is a "many to many" relation between User and Song, it has been necessary to assign both their primary keys mailUser and idSong to it as primary and foreign keys, furthermore it has the attribute favouriteDate to know when the song was added as favourite;
- **Favourite_Playlist**: because Favourite_Playlist is a "many to many" relation between User and Playlist, it has been necessary to assign both their primary keys mailUser and idPlaylist to it as primary and foreign keys, furthermore it has the attribute favouriteDate to know when the playlist was added as favourite;
- **Artist**: it has an auto-incremented id as primary key, and it has the attributes name, surname, stageName, dateOfBirth and recordLabel;
- **Author**: it has an auto-incremented id as primary key, and it has the attributes name, surname and productionLabel;
- **Added_Song_To_Playlist**: because Added_Song_To_Playlist is a "many to many" relation between Song and Playlist, it has been necessary to assign both their primary keys idSong and idPlaylist to it as primary and foreign keys, furthermore it has the attribute addedDate to know when the song was added to the playlist;
- **Grouped_Into**: because Grouped_Into is a "many to many" relation between Category and Song, it has been necessary to assign both their primary keys nameCategory and idSong to it as primary and foreign keys;
- **Category**: its name is the primary key, and it has the attribute description;

1.4. Relational Model

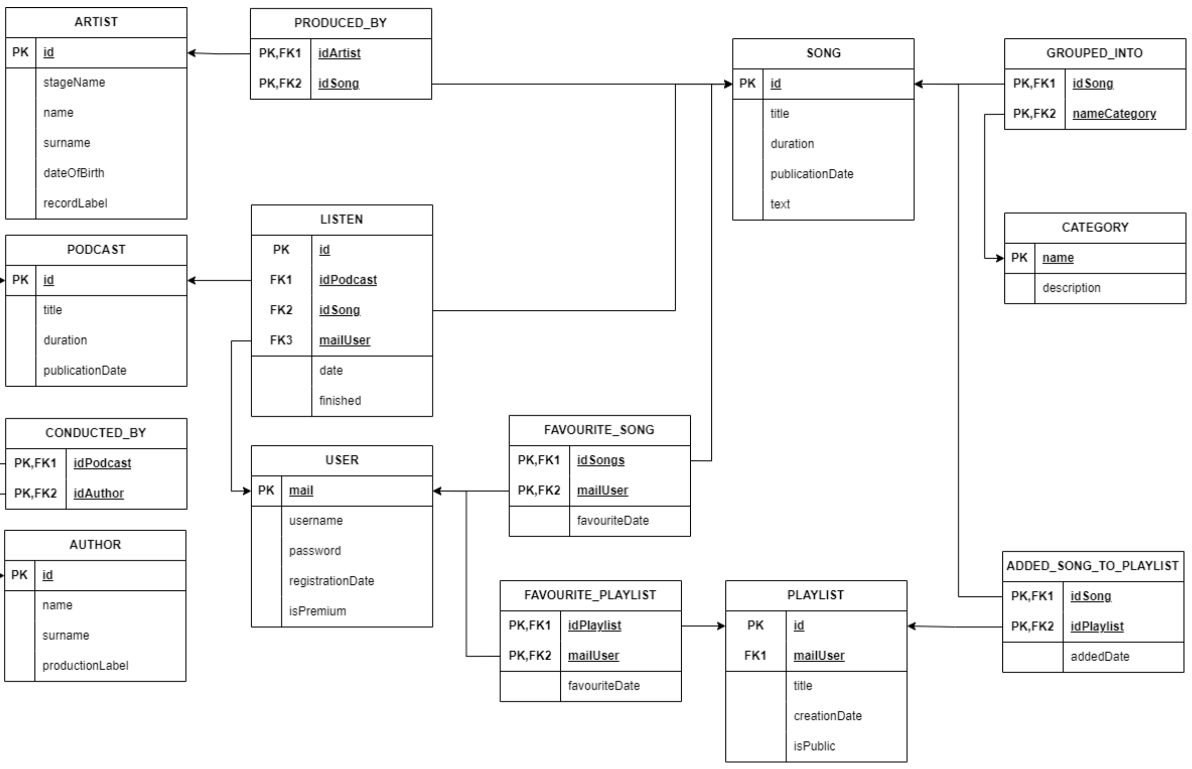


Figure 1.2: Relational Model of the Database

From the user entity, we create the user table with the following attributes: username, Premium, R. Date, Password and Mail(primary key).

From the content, as the hierarchy was total and exclusive, we created the song table and the podcast table. Both of them have the following attributes: ID(primary key), P.date, duration(s), and Title; the song has also the additional attribute Songtext.

From the listen we create the listen table with the following attributes: Id(primary key), Date, HasFinished, mailUser(foreign key), idSong(foreign key), and idPodcast(foreign key). The table has the constraint that mailUSer must be different from null, one among idSong and idPodcast must be different from null and the other equal to null. We don't need a bridge table since listen has both the cardinalities equal to (1,1) so that a listen can only have one content(a song or a podcast) and only one listener.

We create a table for the Author's entity with the following attributes: ID(primary key), name, surname, and ProductionLabel. Then we add a bridge table as a way to represent the “conducted by” relationship that has the couple idAuthor and idPodcast as primary key; both of them are foreign keys, the first refers to the author while the second refers to the podcast.

The produced by relation is represented by a bridge table that has the couple idArtist and idSong as primary key; both of them are foreign keys, the first refers to the artist while the second refers to the song.

From category, we create its table with the following attributes: name(primary key) and description.

The “grouped into” relationship is represented by the bridge table that has the couple idSong and nameCategory as primary key; both of them are foreign keys, the first refers to the song while the second refers to the category.

The “Favourite Song” relationship is depicted by the bridge table that has the couple idSong and idUser as primary key; both of them are foreign keys, the first refers to the song while the second refers to the user. It has also the date attribute. As we said before, by doing this, we don’t value and can’t record multiple additions of the same song by the same user. It would be still possible by adding a dateTime attribute to the primary key.

The “added song to playlist” relationship is represented by the homonymous bridge table that has the primary key formed by idPlaylist and idSong; both of them are foreign keys, the first refers to the playlist while the second refers to the song. The table also has the “AddedDate” attribute.

A playlist is created by only one user so the relationship is expressed in the Playlist entity as a foreign key (mailUser from id in user) that is related to the creator.

The “favourite playlist” relationship is represented by the homonymous bridge table that presents the primary key formed by idPlaylist and idUser; both of them are foreign keys, the first refers to the playlist while the second refers to the user. The table also has the date attribute. As we said before, by doing this, we don’t value and can’t record multiple selections as favourites of the same playlist by the same user. It would be still possible by adding a dateTime attribute to the primary key.

1.5. SQL Requests

1.5.1. Code's Description

CREATION TABLES:

```
CREATE TABLE USER (
    mail VARCHAR(255) PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    password VARCHAR(255),
    registrationDate DATE,
    isPremium BOOLEAN NOT NULL
    );
```

The code allows to create the User table with all its attributes. We consider the username as not null and unique value even if it isn't the primary key; also premium can't be null because in our opinion it is important information from the streaming platform's perspective in a future upgrade of the model.

```
CREATE TABLE SONG (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    duration INT,
    publicationDate DATE,
    text VARCHAR(255)
    );
```

The code allows to create the song table and all its attributes. The id is auto-incremental so that we have the songs ordered following the addition order to the db. The title attribute can't be null. The duration has an INT format and represents the duration in seconds.

```
CREATE TABLE PODCAST (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    duration INT,
    publicationDate DATE
    );
```

The code allows to create the podcast table and all its attributes. The id is auto-incremental so that we have the podcasts ordered following the addition order to the db. The title attribute can't be null. The duration, like the song table, has an INT format and represents the duration in seconds.

```
CREATE TABLE ARTIST (
    id INT AUTO_INCREMENT PRIMARY KEY,
    stageName VARCHAR(255) NOT NULL,
    name VARCHAR(255),
    surname VARCHAR(255),
    dateOfBirth DATE,
    recordLabel VARCHAR(255)
    );
```

The code allows to create the artist table and all its attributes. The id is auto-incremental so that we have the artists ordered following the addition order to the db. The stageName attribute can't be null.

```
CREATE TABLE AUTHOR (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    surname VARCHAR(255) NOT NULL,
    productionLabel VARCHAR(255)
    );
```

The code allows to create the author table and all its attributes. The id is auto-incremental so that we have the authors ordered following the addition order to the db. The name and surname attributes can't be null.

```
CREATE TABLE CATEGORY (
    name VARCHAR(255) PRIMARY KEY,
    description VARCHAR(255)
    );
```

The code allows to create the category table and all its attributes.

```
CREATE TABLE PLAYLIST (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    creationDate DATE,
    isPublic BOOLEAN,
    mailUser VARCHAR(255) NOT NULL,
    FOREIGN KEY (mailUser) REFERENCES USER(mail)
    );
```

The code allows to create the playlist table and all its attributes. The id is auto-incremental so that we have the playlists ordered following the addition order to the db. The mailUser is the foreign key referred to the user.The title can't be null.A playlist always has a creator so mailUser can't be null.

```
CREATE TABLE LISTEN (
    id INT AUTO_INCREMENT PRIMARY KEY,
    date DATE,
    finished BOOLEAN,
    mailUser VARCHAR(255) NOT NULL,
    idSong INT,
    idPodcast INT,
    FOREIGN KEY (mailUser) REFERENCES USER(mail),
    FOREIGN KEY (idSong) REFERENCES SONG(id),
    FOREIGN KEY (idPodcast) REFERENCES PODCAST(id),
    CONSTRAINT EitherForeignKeyIsNull CHECK (
    (idSong IS NULL AND idPodcast IS NOT NULL) OR
    (idSong IS NOT NULL AND idPodcast IS NULL))
```

```
        );
```

The code allows to create the Listen table and all its attributes. The id is auto-incremental so that we have the listens ordered following the addition order to the db. The mailUser(id) is the foreign key referring to the user, idSong(id) is the one referring to Song and the idPocast is the one referring to Podcast(id). We create the constraint by which a listen refers in mutual exclusion to a song or to a podcast. A listen has only 1 user so mailUser can't be null.

```
    CREATE TABLE PRODUCED_BY (
        idArtist INT,
        idSong INT,
        PRIMARY KEY (idArtist, idSong),
        FOREIGN KEY (idArtist) REFERENCES ARTIST(id),
        FOREIGN KEY (idSong) REFERENCES SONG(id)
        );
```

The code allows to create the "Produced By" bridge table and all its attributes. IdSong and IdArtist are the attributes that form the primary key and they also are foreign keys respectively from Song's and Artist's primary keys.

```
    CREATE TABLE CONDUCTED_BY(
        idAuthor INT,
        idPodcast INT,
        PRIMARY KEY (idAuthor, idPodcast),
        FOREIGN KEY (idAuthor) REFERENCES AUTHOR(id),
        FOREIGN KEY (idPodcast) REFERENCES PODCAST(id)
        );
```

The code allows to create the "Conducted By" bridge table and all its attributes. IdPodcast and IdArtist are the attributes that form the primary key and they also are foreign keys respectively from Podcast's and Artist's primary keys.

```
CREATE TABLE FAVOURITE_SONG (
    idSong INT,
    mailUser VARCHAR(255),
    favouriteDate DATE,
    PRIMARY KEY (idSong, mailUser),
    FOREIGN KEY (idSong) REFERENCES SONG(id),
    FOREIGN KEY (mailUser) REFERENCES USER(mail)
    );
```

The code allows to create the "Favourite Song" bridge table and all its attributes. IdSong and mailUser are the attributes that form the primary key and they also are foreign keys respectively from Song's and User's primary keys.

```
CREATE TABLE FAVOURITE_PLAYLIST (
    idPlaylist INT,
    mailUser VARCHAR(255),
    favouriteDate DATE,
    PRIMARY KEY (idPlaylist, mailUser),
    FOREIGN KEY (idPlaylist) REFERENCES PLAYLIST(id),
    FOREIGN KEY (mailUser) REFERENCES USER(mail)
    );
```

The code allows to create the "Favourite Playlist" bridge table and all its attributes. IdPlaylist and mailUser are the attributes that form the primary key and they also are foreign keys respectively from Playlist's and User's primary keys.

```
CREATE TABLE ADDED_SONG_TO_PLAYLIST (
    idSong INT,
    idPlaylist INT,
    addedDate DATE,
    PRIMARY KEY (idSong, idPlaylist),
    FOREIGN KEY (idSong) REFERENCES SONG(id),
    FOREIGN KEY (idPlaylist) REFERENCES PLAYLIST(id)
    );
```

The code allows to create the “Added song to a playlist” bridge table and all its attributes. IdPlaylist and idSong are the attributes that form the primary key and they also are foreign keys respectively from Playlist’s and Song’s primary keys.

```
CREATE TABLE GROUPED_INTO (
    idSong INT,
    nameCategory VARCHAR(255),
    PRIMARY KEY (idSong, nameCategory),
    FOREIGN KEY (idSong) REFERENCES SONG(id),
    FOREIGN KEY (nameCategory) REFERENCES CATEGORY(name)
    );
```

The code allows to create the “Grouped into” bridge table and all its attributes. nameCategory and idSong are the attributes that form the primary key and they also are foreign keys respectively from Category’s and Song’s primary keys.

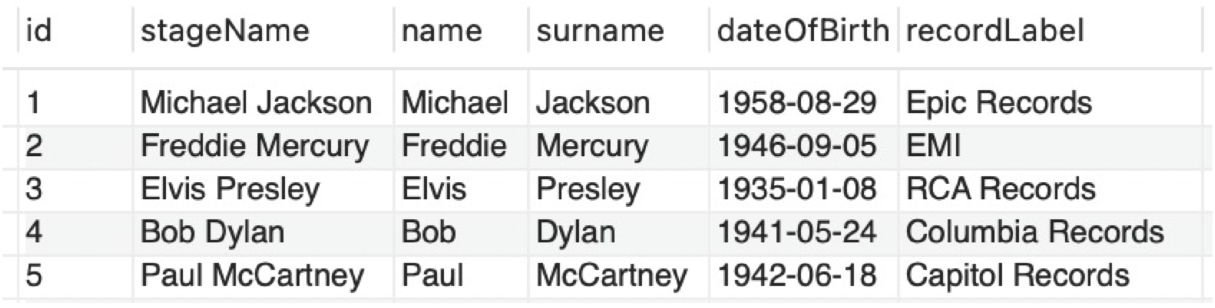
1.5.2. Database Population

We populated the database using the function "INSERT INTO TABLE_NAME(...) VALUES (...), (...), ..., (...)".

So for the first table "ARTIST" we built it in this way:

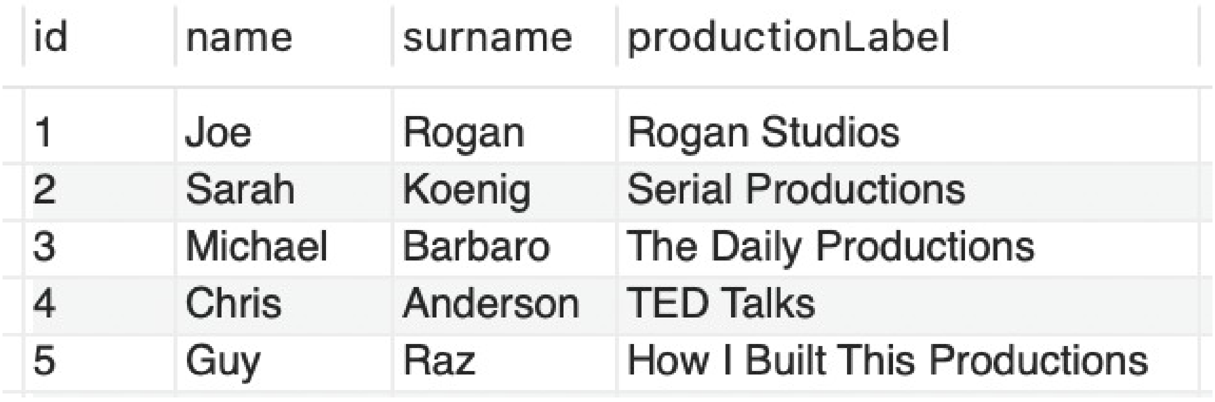
```
INSERT INTO ARTIST (stageName, name, surname, dateOfBirth, recordLabel)

VALUES

('Michael Jackson', 'Michael', 'Jackson', '1958-08-29', 'Epic Records'),

('Freddie Mercury', 'Freddie', 'Mercury', '1946-09-05', 'EMI'),

...
```

In the following pages we will show an example of the first elements of each table we built:



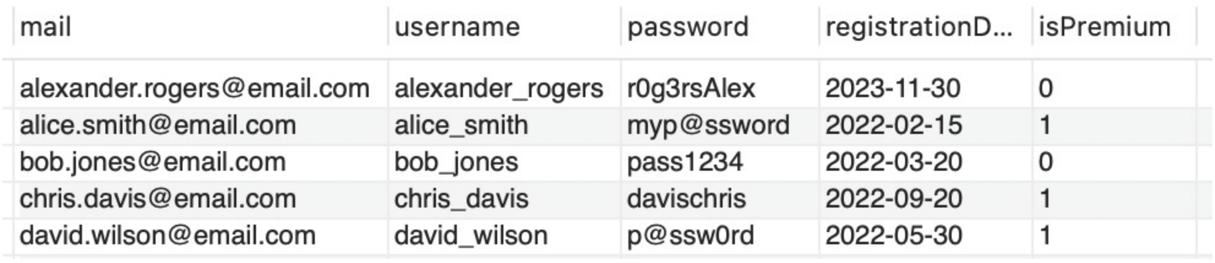
id	stageName	name	surname	dateOfBirth	recordLabel
1	Michael Jackson	Michael	Jackson	1958-08-29	Epic Records
2	Freddie Mercury	Freddie	Mercury	1946-09-05	EMI
3	Elvis Presley	Elvis	Presley	1935-01-08	RCA Records
4	Bob Dylan	Bob	Dylan	1941-05-24	Columbia Records
5	Paul McCartney	Paul	McCartney	1942-06-18	Capitol Records

Table 1.1: ARTIST



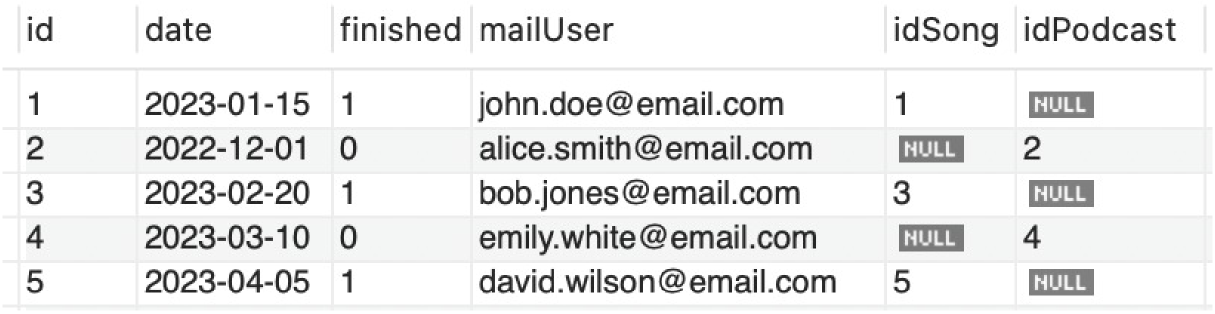
id	name	surname	productionLabel
1	Joe	Rogan	Rogan Studios
2	Sarah	Koenig	Serial Productions
3	Michael	Barbaro	The Daily Productions
4	Chris	Anderson	TED Talks
5	Guy	Raz	How I Built This Productions

Table 1.2: AUTHOR



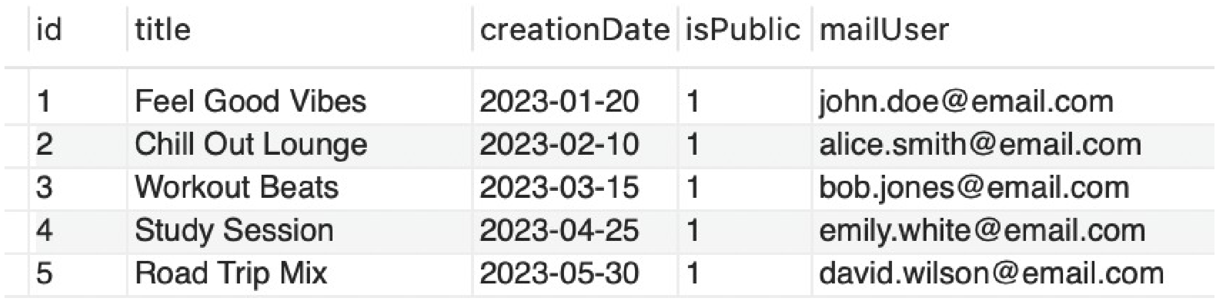
mail	username	password	registrationD...	isPremium
alexander.rogers@email.com	alexander_rogers	r0g3rsAlex	2023-11-30	0
alice.smith@email.com	alice_smith	myp@ssword	2022-02-15	1
bob.jones@email.com	bob_jones	pass1234	2022-03-20	0
chris.davis@email.com	chris_davis	davischris	2022-09-20	1
david.wilson@email.com	david_wilson	p@ssw0rd	2022-05-30	1

Table 1.3: USER



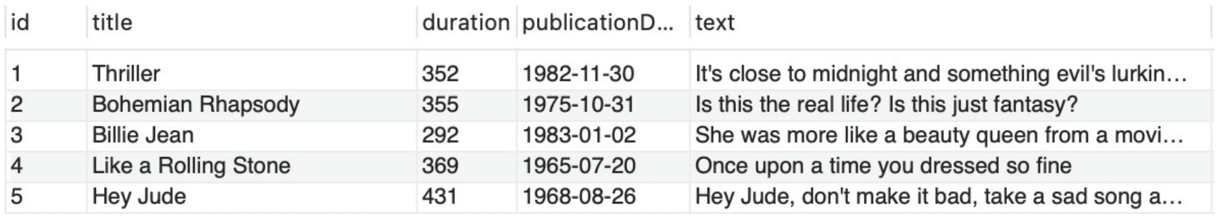
id	date	finished	mailUser	idSong	idPodcast
1	2023-01-15	1	john.doe@email.com	1	NULL
2	2022-12-01	0	alice.smith@email.com	NULL	2
3	2023-02-20	1	bob.jones@email.com	3	NULL
4	2023-03-10	0	emily.white@email.com	NULL	4
5	2023-04-05	1	david.wilson@email.com	5	NULL

Table 1.4: LISTEN



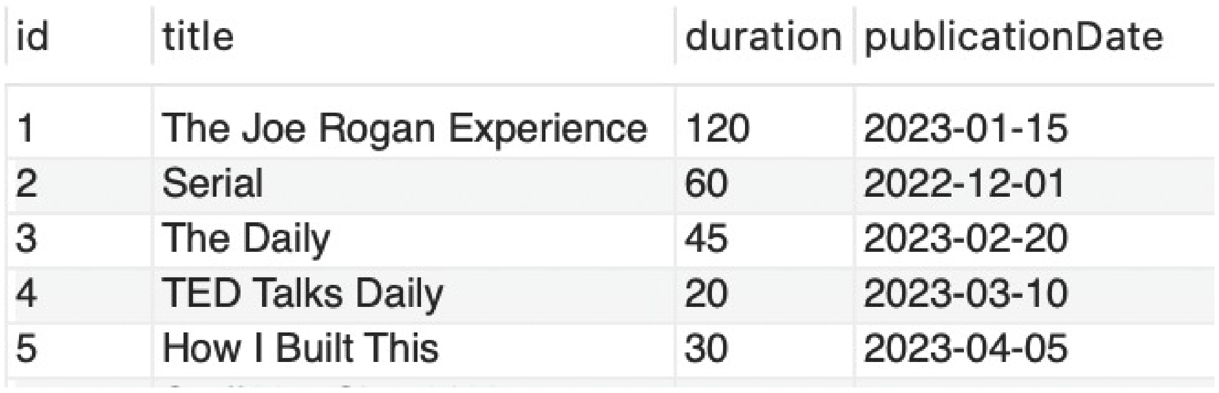
id	title	creationDate	isPublic	mailUser
1	Feel Good Vibes	2023-01-20	1	john.doe@email.com
2	Chill Out Lounge	2023-02-10	1	alice.smith@email.com
3	Workout Beats	2023-03-15	1	bob.jones@email.com
4	Study Session	2023-04-25	1	emily.white@email.com
5	Road Trip Mix	2023-05-30	1	david.wilson@email.com

Table 1.5: PLAYLIST



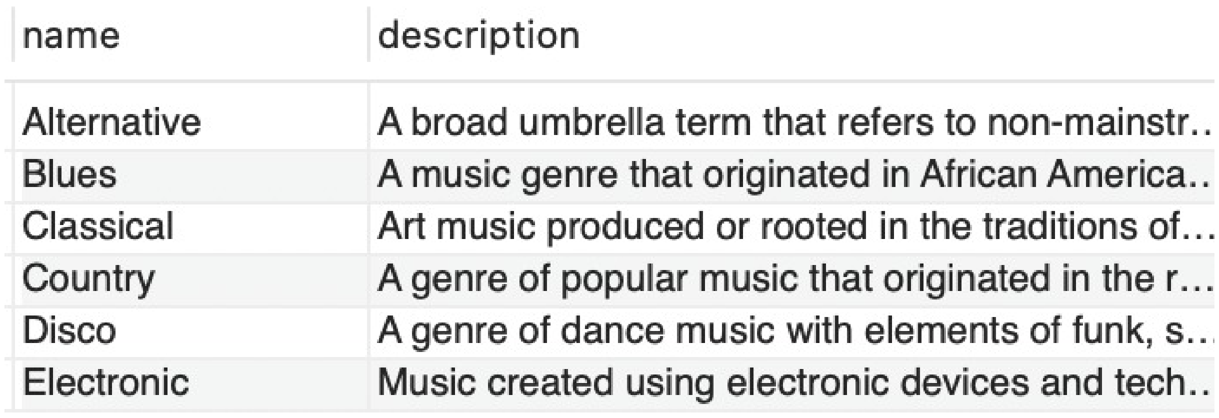
id	title	duration	publicationD...	text
1	Thriller	352	1982-11-30	It's close to midnight and something evil's lurkin…
2	Bohemian Rhapsody	355	1975-10-31	Is this the real life? Is this just fantasy?
3	Billie Jean	292	1983-01-02	She was more like a beauty queen from a movi…
4	Like a Rolling Stone	369	1965-07-20	Once upon a time you dressed so fine
5	Hey Jude	431	1968-08-26	Hey Jude, don't make it bad, take a sad song a…

Table 1.6: SONG



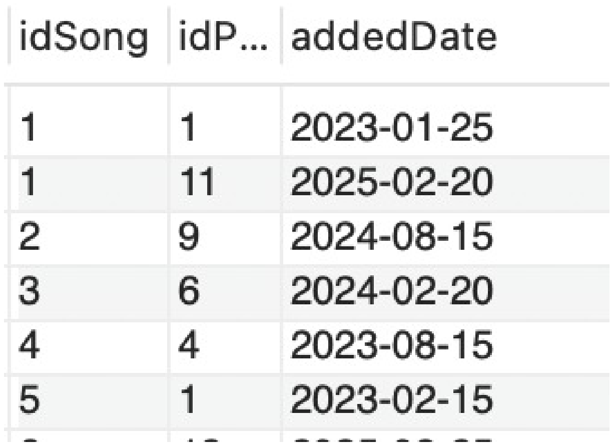
id	title	duration	publicationDate
1	The Joe Rogan Experience	120	2023-01-15
2	Serial	60	2022-12-01
3	The Daily	45	2023-02-20
4	TED Talks Daily	20	2023-03-10
5	How I Built This	30	2023-04-05

Table 1.7: PODCAST



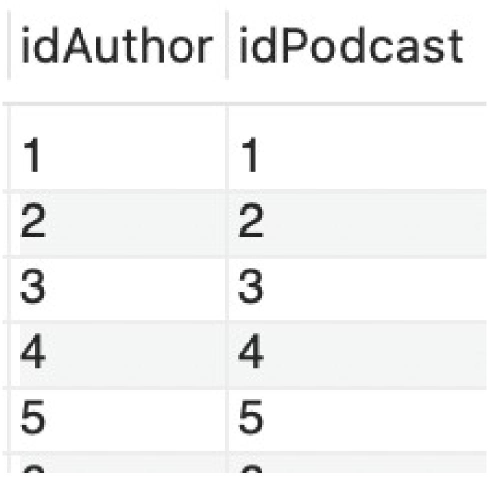
name	description
Alternative	A broad umbrella term that refers to non-mainstr..
Blues	A music genre that originated in African America..
Classical	Art music produced or rooted in the traditions of…
Country	A genre of popular music that originated in the r…
Disco	A genre of dance music with elements of funk, s..
Electronic	Music created using electronic devices and tech..

Table 1.8: CATEGORY



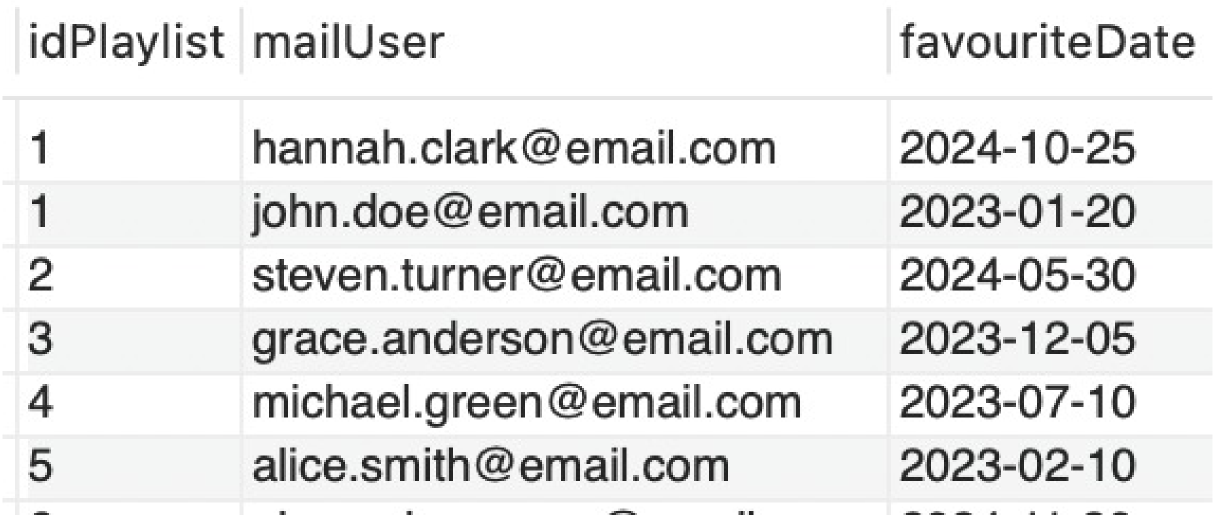
idSong	idP...	addedDate
1	1	2023-01-25
1	11	2025-02-20
2	9	2024-08-15
3	6	2024-02-20
4	4	2023-08-15
5	1	2023-02-15

Table 1.9: ADDED_SONG_TO_PLAYLIST



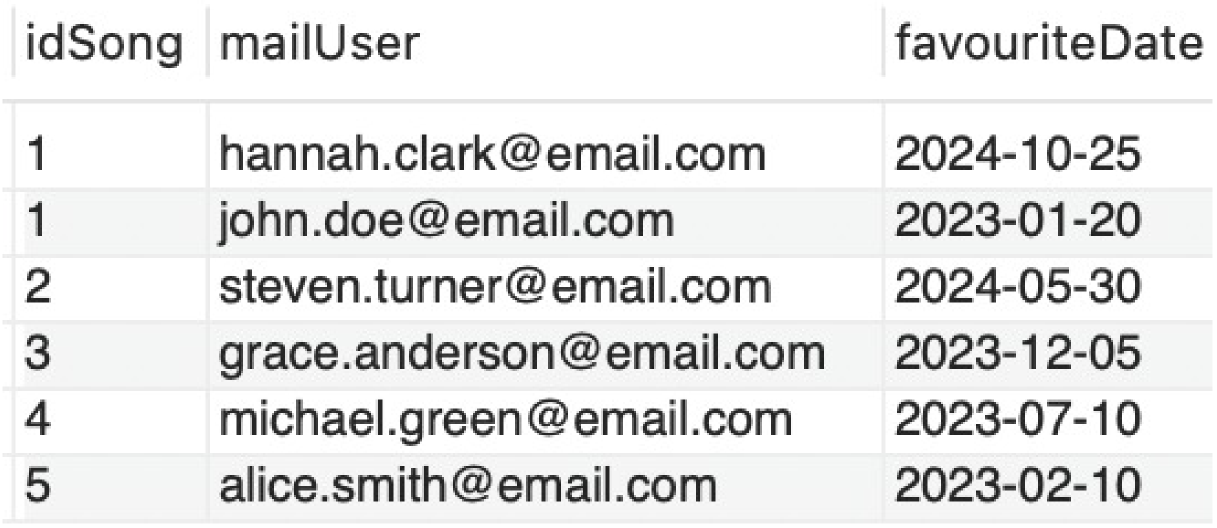
idAuthor	idPodcast
1	1
2	2
3	3
4	4
5	5

Table 1.10: CONDUCTED_BY



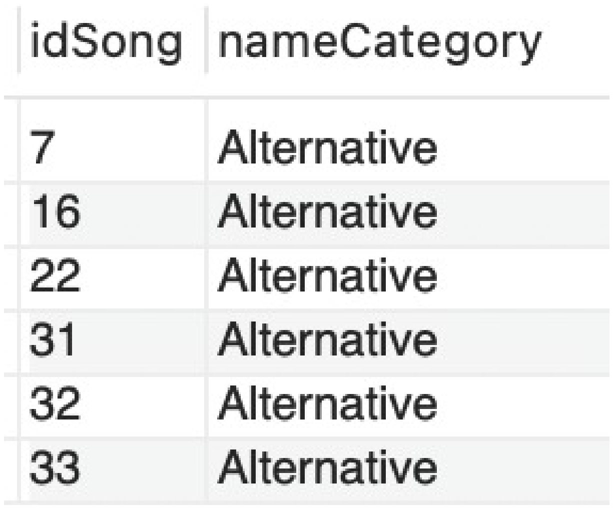
idPlaylist	mailUser	favouriteDate
1	hannah.clark@email.com	2024-10-25
1	john.doe@email.com	2023-01-20
2	steven.turner@email.com	2024-05-30
3	grace.anderson@email.com	2023-12-05
4	michael.green@email.com	2023-07-10
5	alice.smith@email.com	2023-02-10

Table 1.11: FAVOURITE_PLAYLIST



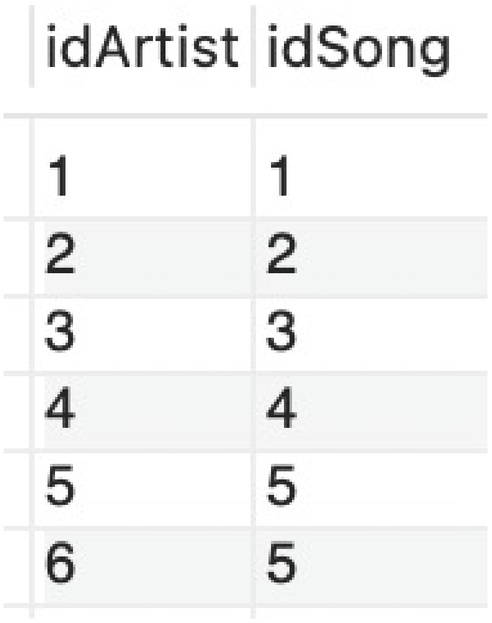
idSong	mailUser	favouriteDate
1	hannah.clark@email.com	2024-10-25
1	john.doe@email.com	2023-01-20
2	steven.turner@email.com	2024-05-30
3	grace.anderson@email.com	2023-12-05
4	michael.green@email.com	2023-07-10
5	alice.smith@email.com	2023-02-10

Table 1.12: FAVOURITE_SONG



idSong	nameCategory
7	Alternative
16	Alternative
22	Alternative
31	Alternative
32	Alternative
33	Alternative

Table 1.13: GROUPED_INTO



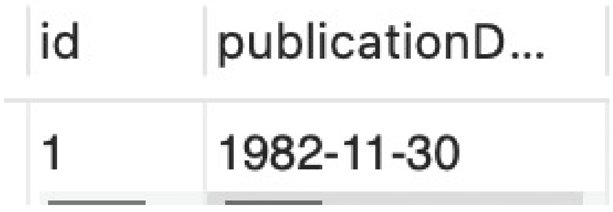
idArtist	idSong
1	1
2	2
3	3
4	4
5	5
6	5

Table 1.14: PRODUCED_BY

1.5.3. Queries

1) Find the Song

```
SELECT id, publicationDate
FROM SONG
WHERE title = "Thriller"
```

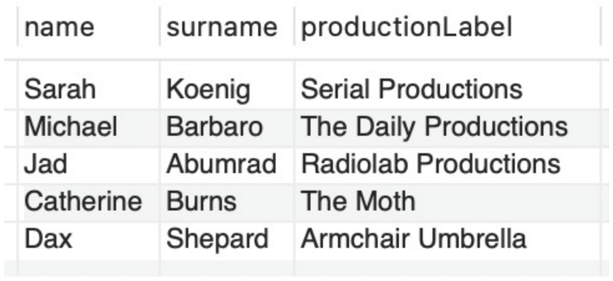


id	publicationD...
1	1982-11-30

This query returns the id and the publication date of the song called "Thriller".

2) The A Authors

```
SELECT name, surname, productionLabel
FROM AUTHOR
WHERE name LIKE "%a%"
LIMIT 5
```

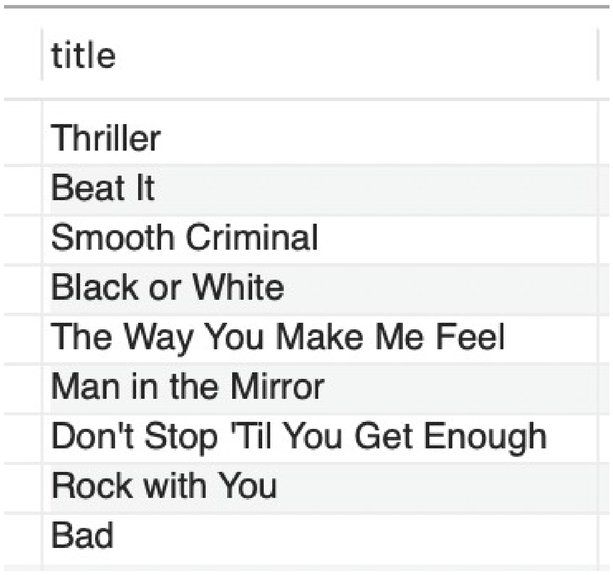


name	surname	productionLabel
Sarah	Koenig	Serial Productions
Michael	Barbaro	The Daily Productions
Jad	Abumrad	Radiolab Productions
Catherine	Burns	The Moth
Dax	Shepard	Armchair Umbrella

This query returns the name,surname and production label for the first 5 authors with a letter "a" in the name.

3) Michael Jackson's songs

```
SELECT title
FROM SONG
WHERE id IN ( SELECT x.idSong
FROM PRODUCED_BY as x join ARTIST as y on (x.idArtist = y.id)
WHERE y.stageName = "Michael Jackson" )
```

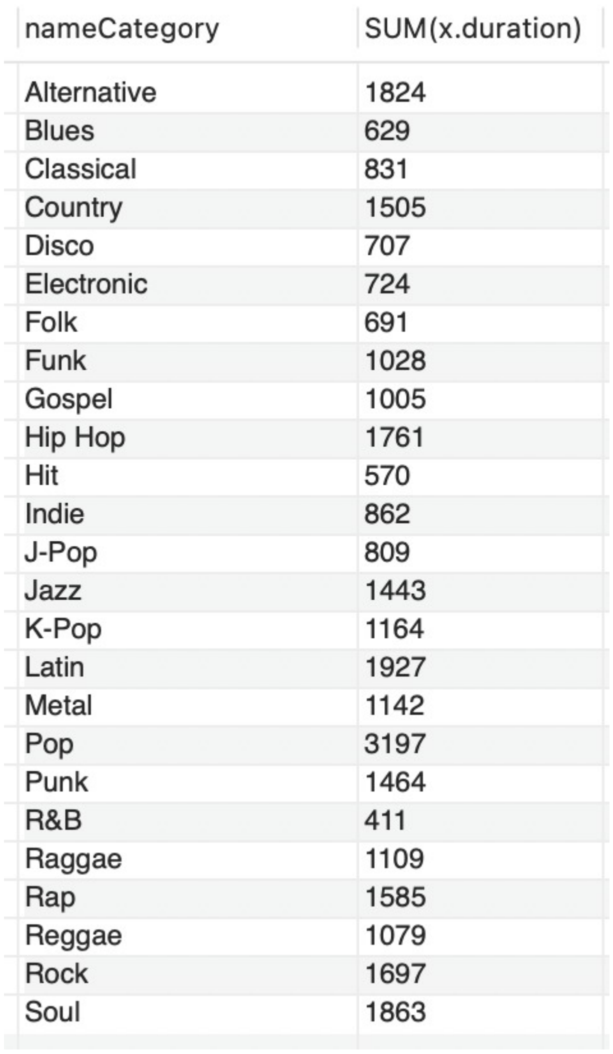


title
Thriller
Beat It
Smooth Criminal
Black or White
The Way You Make Me Feel
Man in the Mirror
Don't Stop 'Til You Get Enough
Rock with You
Bad

This query returns all the songs produced by Michael Jackson.

4) Duration of the songs

```
SELECT y.nameCategory, SUM(x.duration)
FROM SONG AS x JOIN GROUPED_INTO AS y on (x.id = y.idSong)
GROUP BY y.nameCategory
```

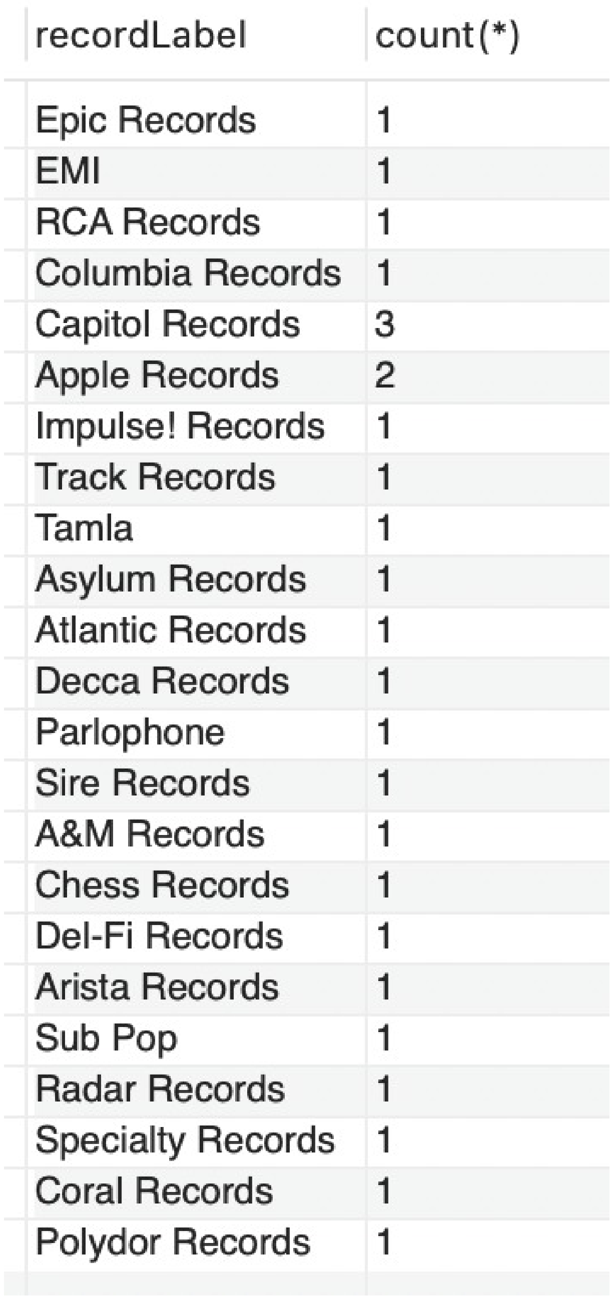


nameCategory	SUM(x.duration)
Alternative	1824
Blues	629
Classical	831
Country	1505
Disco	707
Electronic	724
Folk	691
Funk	1028
Gospel	1005
Hip Hop	1761
Hit	570
Indie	862
J-Pop	809
Jazz	1443
K-Pop	1164
Latin	1927
Metal	1142
Pop	3197
Punk	1464
R&B	411
Raggae	1109
Rap	1585
Reggae	1079
Rock	1697
Soul	1863

For each category, this query returns the sum of the duration of all the songs in the category.

5) How many artist for these labels?

```
SELECT recordLabel, count(*)
FROM ARTIST
GROUP BY recordLabel
```

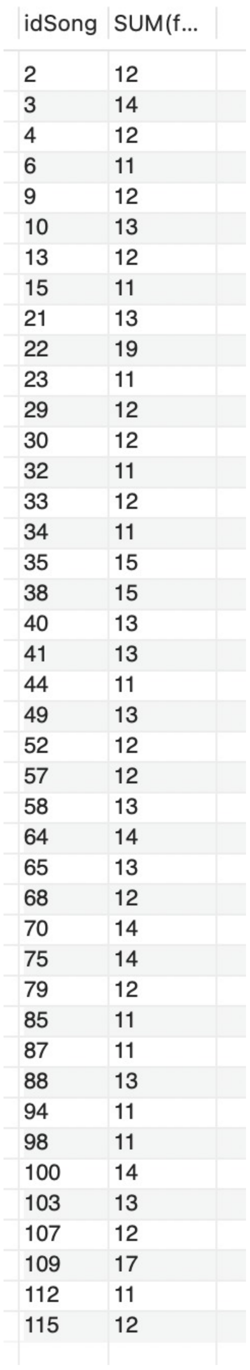


recordLabel	count(*)
Epic Records	1
EMI	1
RCA Records	1
Columbia Records	1
Capitol Records	3
Apple Records	2
Impulse! Records	1
Track Records	1
Tamla	1
Asylum Records	1
Atlantic Records	1
Decca Records	1
Parlophone	1
Sire Records	1
A&M Records	1
Chess Records	1
Del-Fi Records	1
Arista Records	1
Sub Pop	1
Radar Records	1
Specialty Records	1
Coral Records	1
Polydor Records	1

This query returns the number of artists under every record label.

6) Listen to it at least 11 times

```
SELECT idSong,SUM(finished)
FROM LISTEN
GROUP BY idSong
HAVING SUM(finished) > 10
```

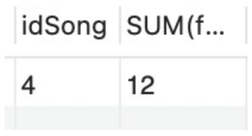


idSong	SUM(f...
2	12
3	14
4	12
6	11
9	12
10	13
13	12
15	11
21	13
22	19
23	11
29	12
30	12
32	11
33	12
34	11
35	15
38	15
40	13
41	13
44	11
49	13
52	12
57	12
58	13
64	14
65	13
68	12
70	14
75	14
79	12
85	11
87	11
88	13
94	11
98	11
100	14
103	13
107	12
109	17
112	11
115	12

This query returns the ID of the songs along with the number of times each song has been listened to in its entirety at least 11 times.

7) Even this Blues must be listened to at least 11 times

```
SELECT idSong,SUM(finished)
FROM LISTEN
WHERE idSong IN (SELECT idSong
                 FROM GROUPED_INTO
                 WHERE nameCategory = "Blues")
GROUP BY idSong
HAVING SUM(finished) > 10
```

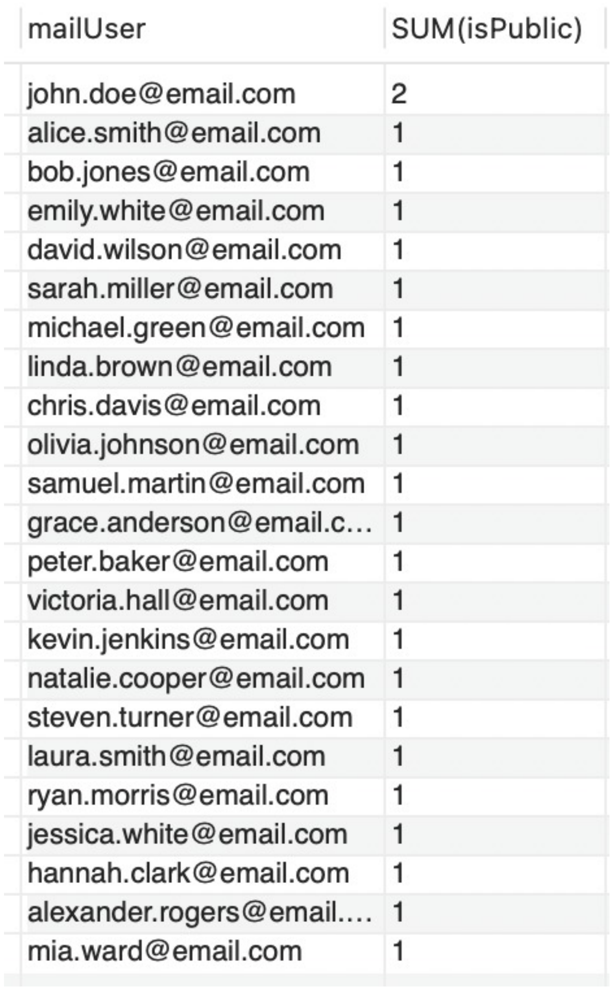


idSong	SUM(f...
4	12

This query returns the ID of the songs from the Blues category along with the number of times each song has been listened to in its entirety at least 11 times

8) How many playlists do you like?

```
SELECT mailUser, SUM(isPublic)
FROM PLAYLIST
WHERE id in (SELECT idPlaylist
             FROM FAVOURITE_PLAYLIST)
GROUP BY mailUser
```

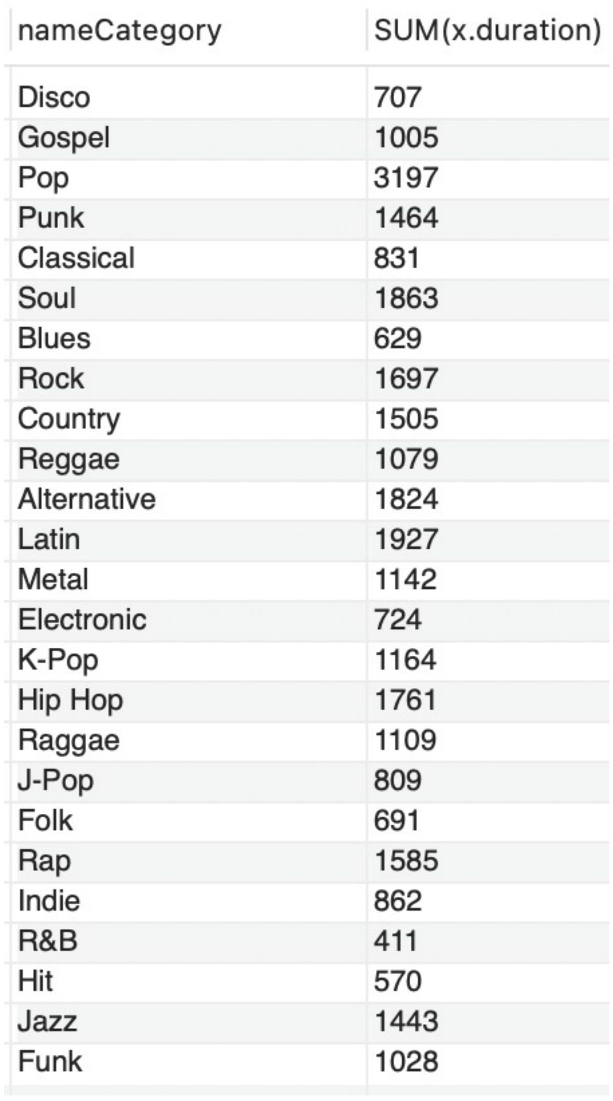


mailUser	SUM(isPublic)
john.doe@email.com	2
alice.smith@email.com	1
bob.jones@email.com	1
emily.white@email.com	1
david.wilson@email.com	1
sarah.miller@email.com	1
michael.green@email.com	1
linda.brown@email.com	1
chris.davis@email.com	1
olivia.johnson@email.com	1
samuel.martin@email.com	1
grace.anderson@email.c...	1
peter.baker@email.com	1
victoria.hall@email.com	1
kevin.jenkins@email.com	1
natalie.cooper@email.com	1
steven.turner@email.com	1
laura.smith@email.com	1
ryan.morris@email.com	1
jessica.white@email.com	1
hannah.clark@email.com	1
alexander.rogers@email....	1
mia.ward@email.com	1

This query returns the sum of the number of public playlists that are marked as favorite by at least one user for each user.

9) Songs for Millennials

```
SELECT y.nameCategory, SUM(x.duration)
FROM SONG AS x join GROUPED_INTO AS y on (x.id = y.idSong)
WHERE x.publicationDate < ’2000-01-01’
GROUP BY y.nameCategory
HAVING SUM(x.duration) > 300
```

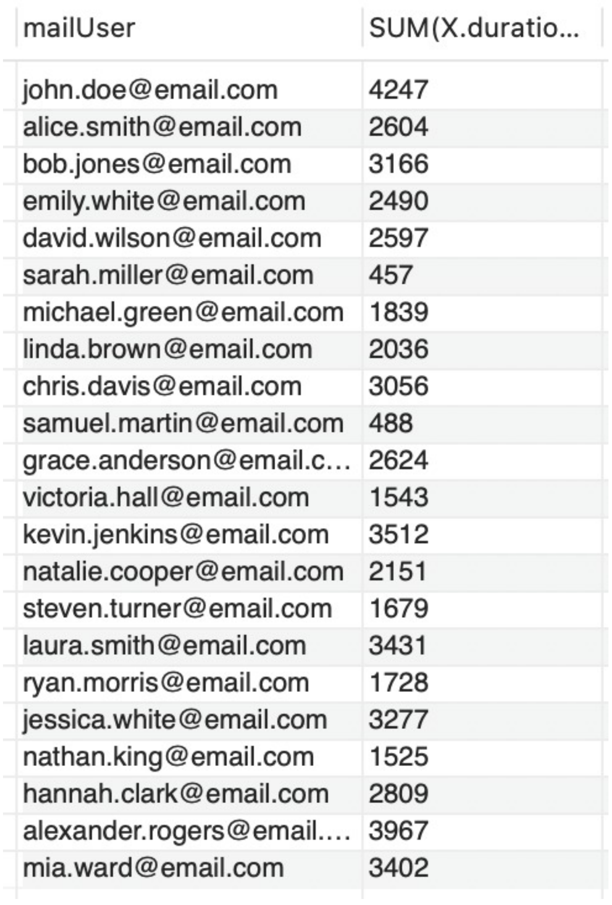


nameCategory	SUM(x.duration)
Disco	707
Gospel	1005
Pop	3197
Punk	1464
Classical	831
Soul	1863
Blues	629
Rock	1697
Country	1505
Reggae	1079
Alternative	1824
Latin	1927
Metal	1142
Electronic	724
K-Pop	1164
Hip Hop	1761
Raggae	1109
J-Pop	809
Folk	691
Rap	1585
Indie	862
R&B	411
Hit	570
Jazz	1443
Funk	1028

This query returns the names of the categories and the sum of the duration of the songs released before 2001 with a total duration of at least 300 seconds.

10) Emails and duration of real listeners

```
SELECT Z.mailUser, SUM(X.duration)
FROM (SONG as X join ADDED_SONG_TO_PLAYLIST as Y
        on (x.id = y.idSong)) join PLAYLIST as Z on (Y.idPlaylist = Z.id)
WHERE Z.isPublic = True
GROUP BY Z.mailUser
HAVING SUM(X.duration) > 400
```

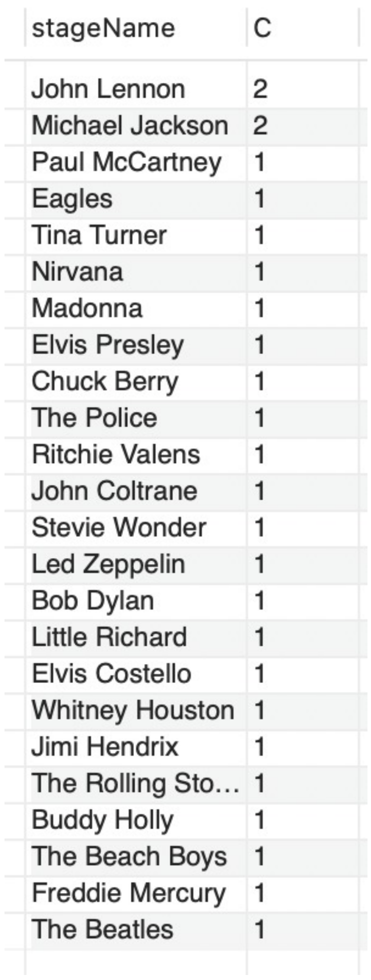


mailUser	SUM(X.duratio...
john.doe@email.com	4247
alice.smith@email.com	2604
bob.jones@email.com	3166
emily.white@email.com	2490
david.wilson@email.com	2597
sarah.miller@email.com	457
michael.green@email.com	1839
linda.brown@email.com	2036
chris.davis@email.com	3056
samuel.martin@email.com	488
grace.anderson@email.c...	2624
victoria.hall@email.com	1543
kevin.jenkins@email.com	3512
natalie.cooper@email.com	2151
steven.turner@email.com	1679
laura.smith@email.com	3431
ryan.morris@email.com	1728
jessica.white@email.com	3277
nathan.king@email.com	1525
hannah.clark@email.com	2809
alexander.rogers@email....	3967
mia.ward@email.com	3402

This query returns for each user, the user's email, and the sum of the duration of the songs that are in his public playlists, only if the sum is at least 400 seconds.

11) Top favourite songs

```
SELECT stageName, count(*) as C
FROM (ARTIST as a join PRODUCED_BY as p
        on (a.id=p.idArtist))join favourite_song as F on (f.idSong=p.idSong)
GROUP by stageName
ORDER by C DESC
```

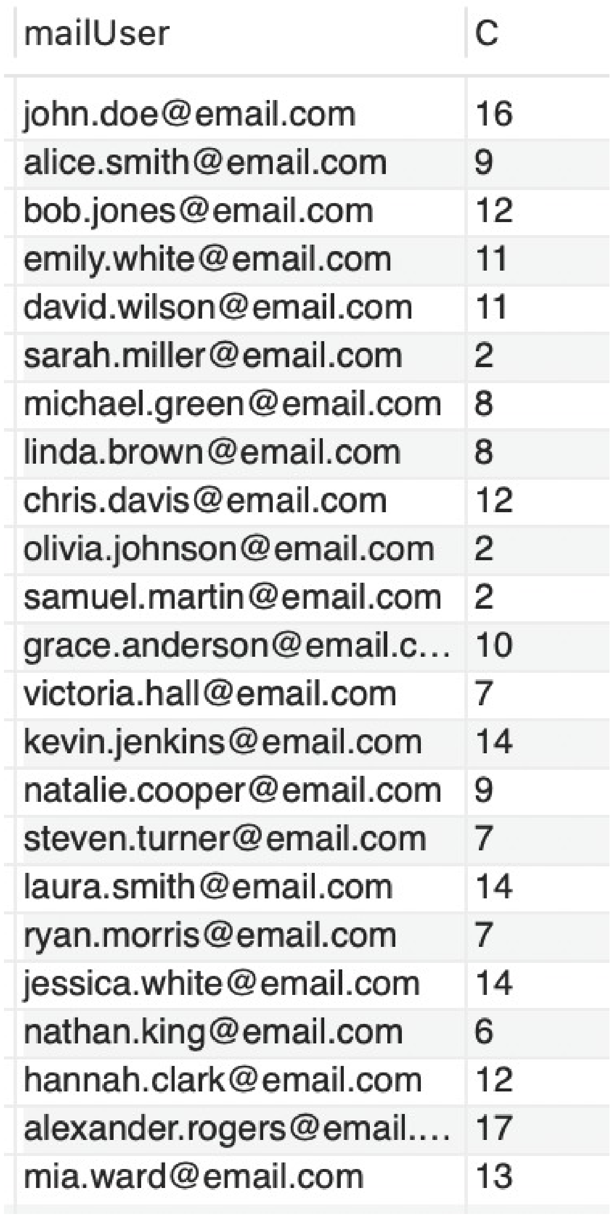


stageName	C
John Lennon	2
Michael Jackson	2
Paul McCartney	1
Eagles	1
Tina Turner	1
Nirvana	1
Madonna	1
Elvis Presley	1
Chuck Berry	1
The Police	1
Ritchie Valens	1
John Coltrane	1
Stevie Wonder	1
Led Zeppelin	1
Bob Dylan	1
Little Richard	1
Elvis Costello	1
Whitney Houston	1
Jimi Hendrix	1
The Rolling Sto…	1
Buddy Holly	1
The Beach Boys	1
Freddie Mercury	1
The Beatles	1

This query returns in descending order the number of songs that are marked as favorites by at least one user for each artist that has songs marked as favorites.

12) Show all the songs in the playlists

```
SELECT Z.mailUser, count(*) as C
FROM (SONG as X join ADDED_SONG_TO_PLAYLIST as Y
        on (x.id = y.idSong)) join PLAYLIST as Z on (Y.idPlaylist = Z.id)
WHERE Z.isPublic = True
GROUP BY Z.mailUser
HAVING C > 1
```

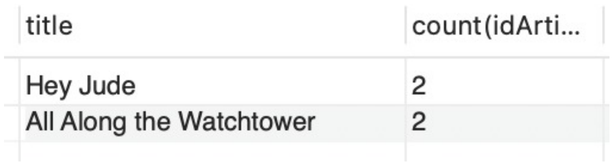


mailUser	C
john.doe@email.com	16
alice.smith@email.com	9
bob.jones@email.com	12
emily.white@email.com	11
david.wilson@email.com	11
sarah.miller@email.com	2
michael.green@email.com	8
linda.brown@email.com	8
chris.davis@email.com	12
olivia.johnson@email.com	2
samuel.martin@email.com	2
grace.anderson@email.c...	10
victoria.hall@email.com	7
kevin.jenkins@email.com	14
natalie.cooper@email.com	9
steven.turner@email.com	7
laura.smith@email.com	14
ryan.morris@email.com	7
jessica.white@email.com	14
nathan.king@email.com	6
hannah.clark@email.com	12
alexander.rogers@email....	17
mia.ward@email.com	13

This query returns the number of songs that are in a user's public playlists for each user.

13) Songs with more producers

```
SELECT title, count(idArtist)
FROM SONG JOIN PRODUCED_BY ON (id = idSong)
GROUP BY title
HAVING count(idArtist) > 1
```

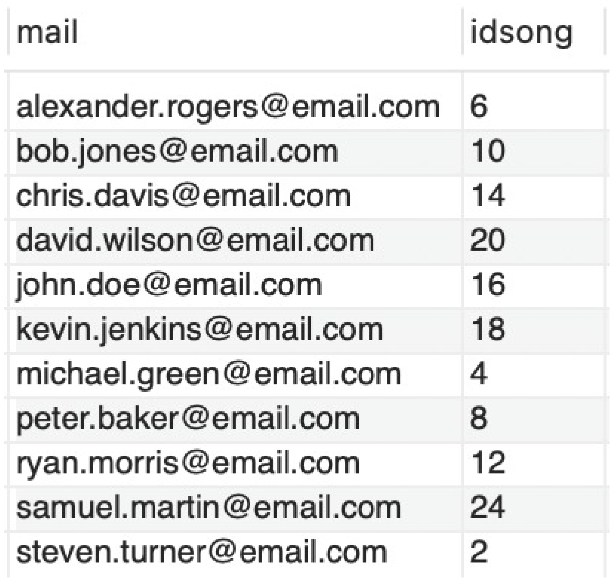


title	count(idArti...
Hey Jude	2
All Along the Watchtower	2

This query selects the songs that have been produced by at least two different producers

14) It's my favourite but I've never listend to it

```
SELECT mail,idsong
FROM USER as u join FAVOURITE_SONG as f on (u.mail=f.mailUser)
WHERE idSong not in(
                SELECT idSong
                FROM LISTEN
                WHERE (mailUser=u.mail)
                )
```



mail	idsong
alexander.rogers@email.com	6
bob.jones@email.com	10
chris.davis@email.com	14
david.wilson@email.com	20
john.doe@email.com	16
kevin.jenkins@email.com	18
michael.green@email.com	4
peter.baker@email.com	8
ryan.morris@email.com	12
samuel.martin@email.com	24
steven.turner@email.com	2

This query returns the songs for each user that he has checked as favorites even if he has never listened to them

A | Appendix

Here you can find the link for the ER and Relational Models files. Remember that to see both of them you have to open the file with diagrams.net (in the top section of the web page).

https://drive.google.com/file/d/1T8G-chJPXUrcEDJAtj-SuLVV58GpoU2E/view?usp=sharing

List of Figures

List of Tables