Bases de Dados 2018/2019

Gestão de Streaming de Música

Grupo 710:
André Mamprin Mori - up201700493
Daniel Gazola Bradaschia - up201700494

Descrição

O tema escolhido para nosso projeto é uma base de dados para gerir um serviço de Streaming de música (ex. Spotify, Apple Music).

A classe principal do projeto é a *User.* Considerando suas relações, a princípio um utilizador pode seguir outros utilizadores, *playlists* e artistas, sendo essa última relação do tipo não-recíproca. Nota-se que diversas informações são guardadas referentes ao utilizador, nomeadamente: nome, idade, email, imagem de perfil.

Por fim, existe uma relação de agregação entre as classes *User* e *Subscription*, sendo que a última armazena informações sobre a escolha de inscrição do utilizador, tais como o tipo, a taxa, método de pagamento e data de validade.

Em seguida, temos a classe *Playlist* que, como dito anteriormente, pode ser seguida por um utilizador assim como pode ser de pertence. A classe possui como atributos o nome, imagem de capa e descrição. É importante apontar a existência de uma subclasse nomeada *Top*, com atributo data, que se trata de uma *playlist* gerada a partir das músicas mais ouvidas.

Por definição, *playlists* são constituídas por uma ou mais músicas, que são representadas pela classe *Song*, com atributos de nome, tempo de duração, e contagem de reproduções. Existe uma relação direta com a classe *Top,* com a presença de uma classe de atributo na qual se armazena o índice da posição da música em um *ranking* de mais ouvidas.

A partir da relação de composição da classe *Song,* temos a classe *Album,* com seus respectivos atributos. Nota-se que esta classe é necessária para a organização e armazenamento do trabalho dos artistas, representados pela classe *Artist* e que tem uma relação de concatenação com *Album.*

A classe *Artist,* com estrutura composta por nome, imagem de perfil, imagem de capa e biografia, possui duas relações intrínsecas com duas outras: *Genre* e *Concert.* Da primeira temos uma classe que definirá uma característica fulcral de todo artista, gravando seu nome e uma breve descrição sobre seu estilo musical. Já da classe *Concert* temos uma relação direta em que artistas podem ter, ou não, um concerto agendado para o futuro.

Devido a relação naturalmente associativa entre *Artist* e *Concert* foi escolhido implementar uma classe *Location,* responsável por armazenar o local do concerto. Por fim, percebeu-se possíveis relações entre *Location, User e Artist,* sendo relevante guardar a localização de cada uma dessas classes, assim foram estabelecidas tais relações.

Atributos

User: *Name, Email, Profile Picture, Age*

Subscription: *Subscription type, Subscription Fee, Payment Method, Expiration Date*

Playlist: *Name, Cover Image, Description*

Top: *Date*

Song: *Name, Length, Play Counts*

Album: *Name, Number of Songs, Year, Label, Cover Picture*

Artist: *Name, Profile Picture, Cover Picture, Bio*

Genre: *Name, Description*

Concert: *Date*

Location: *City, Country*

Esquema Relacional e Dependências Funcionais

User(idUser, idLocation->Location, name, email, profilePicture, age)
{idUser} -> {idLocation, name, email, profilePicture, age}

Subscription(idUser->User, subscriptionType, subscriptionFee, paymentMethod, expirationDate)
1.{idUser->User} -> {subscriptionType}
2.{subscriptionType } -> {subscriptionFee, paymentMethod,expirationDate}

Location(idLocation,city, country)
{idLocation} -> {city, country}

Genre(idGenre, name, description)
{idGenre} -> {name, description}

Artist(idArtist, idLocation->Location, idGenre->Genre, name, profilePicture, coverPicture, bio)
{idArtist} -> {idLocation, idGenre, name, profilePicture, coverPicture, bio}

Concert(idConcert, idArtist->Artist, idUser->User, idLocation->Location, date)
{idConcert} -> {idArtist, idUser, idLocation, date}

Album(idAlbum, idArtist->Artist, name, numberSongs, year, label, coverPicture)
{idAlbum} -> {idArtist, name, numberSongs, year, label, coverPicture}

Song(idSong->Album, name, length, playCounts)
{idSong} -> {name, length, playCounts}

Playlist(idPlaylist->User, name, coverImage, description)
{idPlaylist->User} -> {name, coverImage, description}

Top(idTop->Playlist, dateCreation)
{idTop->Playlist} -> {dateCreation}

Forma Normal

Para identificar a 3ª Forma Normal, será necessário assegurar o cumprimento da regra de não-transitividade. Caso esta regra seja quebrada, também será a Forma Normal de Boyce-Codd, visto esta se tratar de uma versão ligeiramente mais restrita da anterior.

A violação ocorre na classe *Subscription,* porque, através de seu identificador podemos chegar ao seu tipo e a partir do tipo conseguimos os atributos restantes.

Relativamente às restantes relações enumeradas na página anterior, não existirá quebra da 3ª Forma Normal, nem da Forma Normal de Boyce-Codd na medida em que o lado esquerdo de cada dependência é uma super-key do esquema relacional – condição suficiente para cumprir ambas.

Restrições

Para assegurar uma boa manutenção da base de dados, assim como fornecer segurança adicional ao utilizador, recorreu-se ao uso de restrições na produção de várias classes, nomeadamente do tipo chave, de integridade referencial, CHECK e NOT NULL.

Nas ocasiões onde a restrição NOT NULL é associada a um atributo, manifesta-se a obrigatoriedade da existência deste mesmo atributo para a formação mínima da classe. Analisaremos tais ocasiões:

1. Nomeadamente, toda classe com atributo *Name*
2. *City e Country* devem obviamente serem preenchidos
3. A classe *User* deve obrigatoriamente ter um email

A restrição CHECK foi usada com o objetivo de restringir certos atributos, mantendo um certo nível de realidade nestes casos:

1. Em *Songs, playCounts* e *length* devem ser maiores que 0
2. Na classe *User,* sua idade deve ser positiva
3. *Album* deve possuir musicas e ter ano superior a 1900
4. *SubscriptionType* de ser obrigatoriamente “F” ou “P”, para *free ou premium* respectivamente

A restrição UNIQUE foi especialmente aplicada em *Genre,* nomeadamente em seu nome, o qual não é uma chave, mas deve possuir uma denominação única. Por último, relativamente a restrições de integridade referencial, aplicámos chaves estrangeiras a classes intrinsecamente relacionadas entre si.

Interrogações

Foram desenvolvidos 10 *queries* que testam o correto funcionamento dos diversos componentes da base de dados:

1) Listar todos os Álbuns lançados no ano 2000:

   Percorrendo os registros da classe ***Album,*** após verificar o parâmetro year, retorna uma lista de álbuns lançados no ano especificado.

2) Contar todos os utilizadores Premium:

   Com o uso da operação COUNT(), verifica-se a relação entre as classes ***User*** e ***Subscription***, contando quantas assinaturas dos utilizadores são Premium.

3) Listar os artistas de Metal que são de New York, USA:

A relação entre ***Artist***, ***Genre*** e ***Location*** é verificada após operações de NATURAL e INNER JOIN entre as tabelas, verificando se o artista é do local e do gênero especificado.

4) Concertos que ocorreram nos primeiros 6 meses de 2015:

   Verifica a relação entre ***Concert*** e ***Artist***, utilizando o atributo do tipo DATE da classe para obter os que encontram-se dentro da data especificada, com o uso de BETWEEN.

5) Número de álbuns produzidos por cada Label em ordem crescente:

   Com ***Album*** obtém-se uma lista de todas as Labels e quantos álbuns cada uma produziu, utilizando a operação COUNT() e operações de arranjo GROUP BY e ORDER BY, para organizar em ordem crescente.

6) Listar as 5 músicas mais ouvidas na plataforma:

   Em ***Song,*** utilizando a funcionalidade ORDER BY, obtém-se uma lista das músicas ordenadas em ordem decrescente por número de playCounts, e utiliza-se a função LIMIT para obter apenas 5.

7) Número de playlists consideradas TOP criadas por um utilizador:

   Em ***Top*** utilizando a operação COUNT(), recebe-se o número de playlists que são consideradas Top.

8) Lista datas em que playlists tornaram-se tops:

   Verifica a relação entre ***Top***, ***Playlist*** e ***User*** após aplicar operações de INNER JOIN entre as tabelas e com a funcionalidade GROUP BY para retornar uma tabela com o utilizador e a data em que a playlist tornou-se Top.

9) Listar Artistas e utilizadores com suas Localidades em ordem alfabética do país:

   Verifica a relação entre ***User*** e ***Location*** e entre ***Artist*** e ***Location***. Retorna uma lista com utilizadores, artistas e suas respectivas localidades, por meio da utilização da função UNION, que une duas tabelas em uma.

10) Total de play counts para cada gênero:

    Verifica a relação entre ***Artist***, ***Genre***, ***Song*** e ***Album*** utilizando a operação SUM() para somar as playCounts das músicas de cada gênero disponível. Ao final utiliza-se a função GROUP BY para organizar a tabela pelos gênero.

Gatilhos

Para garantir a monitorização e a manutenção da base de dados foram implementados os seguintes gatilhos:

1) *AddUser*

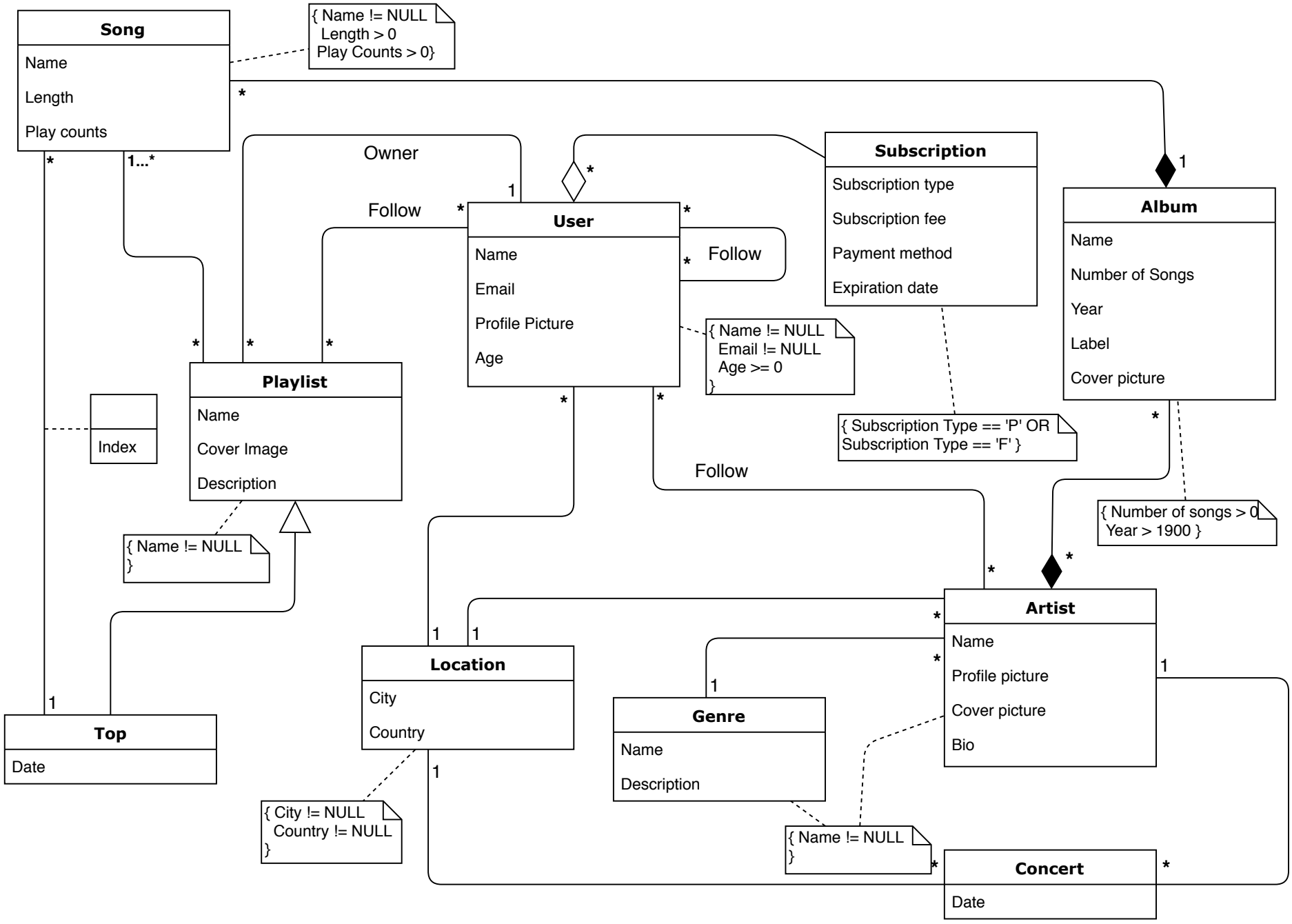
Após a criação de um *USER,* deve-se garantir a criação de sua *SUBSCRIPTION,* que por definição será do tipo *FREE.*

2) *UpdtConcert*

Quando ocorre a necessidade de se alterar *DATECONCERT* de um determinado concerto, devemos garantir que a atualização seja válida, não podendo ser uma data nula.

3) *AddSong*

Quando adiciona-se uma nova música, é necessário atualizar o álbum a qual ela pertence, nomeadamente o campo *NUMBERSONGS.*


Song
Name
Length
Play counts
{ Name != NULL
Length > 0
Play Counts > 0}
Owner
Follow
User
Name
Email
Profile Picture
Age
Follow
{ Name != NULL
Email != NULL
Age >= 0
}
Subscription
Subscription type
Subscription fee
Payment method
Expiration date
{ Subscription Type == 'P' OR
Subscription Type == 'F' }
Album
Name
Number of Songs
Year
Label
Cover picture
{ Number of songs > 0
Year > 1900 }
Playlist
Name
Cover Image
Description
{ Name != NULL
}
Index
Follow
Top
Date
Location
City
Country
{ City != NULL
Country != NULL
}
Genre
Name
Description
{ Name != NULL
}
Artist
Name
Profile picture
Cover picture
Bio
Concert
Date