**FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO**





Gestão e Estudo de Preferências Musicais



Base de Dados

Licenciatura em Engenharia Informática e Computação

**Diogo Miguel Ferreira da Costa** *up202007770* **José António Santos Costa** *up202004823* **Manuel João Gomes Alves Amorim** *up202007485*

20 novembro, 2021

Índice

[Descrição do Contexto 3](#_Toc89962683)

Diagrama UML ……………………………………………………………………………………………………………………………. 4

[Diagrama Relacional e Dependências Funcionais 5](#_Toc89962684)

[Formas Normais 7](#_Toc89962686)

[Restrições 8](#_Toc89962687)

[Diagrama UML Revisto 10](#_Toc89962689)

# Descrição do Contexto

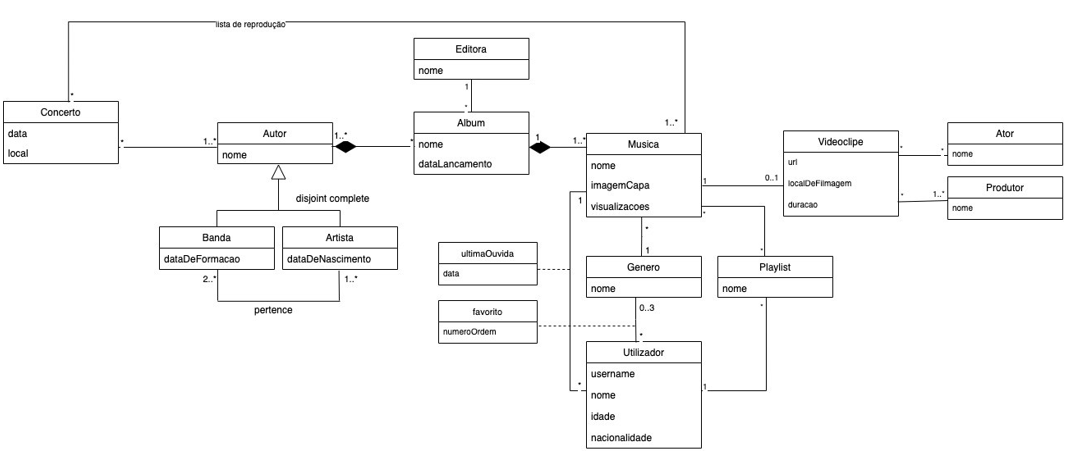
Uma empresa organizadora de eventos, especializada na gestão de festivais e concertos, pretende entender melhor as preferências audiovisuais da população jovem. Desta forma, a empresa, na organização do próximo festival, denominado “Paços de Dança”, que decorrerá durante o mês de agosto em Paços de Ferreira, pretende proporcionar aos seus consumidores uma experiência em conformidade com os seus gostos tanto musicais como visuais. Para isso, terá não só de estudar as músicas mais ouvidas pelos utilizadores de plataformas de *streaming* de música, como também deverá recolher dados acerca dos videoclips e dos concertos realizados pelos seus autores.

Os **utilizadores** destas plataformas são identificados por um nome, *username* único, idade e nacionalidade. Estes podem criar várias **playlists**, constituídas por um conjunto de **músicas**, podendo cada **música** pertencer a inúmeras **playlists**. Cada **música** é caracterizada por um nome e o número de visualizações. Adicionalmente, uma dada **música** pertence a um único **género** e poder-se-á aceder aos três **géneros** musicais favoritos de um dado **utilizador** (cada um associado a um número de ordem) e à sua última **música** ouvida, estando esta associada a uma data.

Cada **música** pertence a um e apenas um **álbum**, que tem um nome e uma data de lançamento associado. Cada **álbum** pertence a uma **editora**, da qual sabemos o respetivo nome. **Um álbum** é constituído por uma ou mais **músicas** e cada **editora** pode produzir vários **álbuns**. Por sua vez, cada **álbum** é criado por um ou mais **autores**, no caso de haver colaborações, e cada **autor** poder produzir inúmeros **álbuns**. Os **autores** subdividem-se em **artistas** a solo, dos quais sabemos a data de nascimento, e em **bandas**, constituídas por 2 ou mais **artistas**, das quais sabemos a data de formação. Sobre os **concertos**, que estão associados a 1 ou mais **autores**, registaremos a data e o local de sua realização e a lista das **músicas** reproduzidas durante o **concerto** (formada por pelo menos 1 música).

Para o estudo dos gostos visuais dos utilizadores, de forma a produzir um festival esteticamente apelativo, será associado a cada **música**, caso exista, um **videoclip**, caracterizado por um *URL*, um local de filmagem e uma duração. Este pode ser protagonizado por inúmeros **atores** e é produzido por um ou mais **produtores,** todos eles definidos pelo seu nome.

Diagrama UML



4

# Diagrama Relacional e Dependências Funcionais

**Utilizador** (id, nome, username, dataNascimento, nacionalidade, idUltimaOuvida->Música)

{username} -> {id, nome, dataNascimento, nacionalidade, username, idUltimaOuvida }

{id} -> {nome, dataNascimento, nacionalidade, username, idUltimaOuvida}

**Música** (id, nome, imagemCapa, reproduções, idVideoclip->Videoclip, idGénero->Género, idÁlbum->Álbum)

{id} -> {nome, imagemCapa, reproduções, idVideoclip, idGénero, idÁlbum)

**Género** (id,nome)

{id} -> {nome}

{nome} -> {id}

**Playlist** (id, nome, idUtilizador->Utilizador)

{id} -> {nome, idUtilizador }

**Álbum** (id, nome, dataLançamento, idEditora->Editora)

{id} -> {nome, dataLançamento, idEditora}

**Editora** (id, nome)

{id} -> {nome}

**Autor** (id, nome)

{id} -> {nome}

**Banda** (id->Autor, nome->Autor, dataDeFormação)

{id} -> {nome, dataDeFormação}

**Artista** (id->Autor, nome->Autor, dataDeNascimento)

{id} -> {nome, dataDeNascimento}

# Diagrama Relacional e Dependências Funcionais (continuação)

**Concerto** (id, data, local)

{id} -> {data, local}

**Videoclip** (id, url, localDeFilmagem, duração)

{id} -> {url, localDeFilmagem, duração}

**Ator** (id, nome)

{id} -> {nome}

**Produtor** (id, nome)

{id} -> {nome}

**VideoclipAtor** (idVideoclip->Videoclip, idAtor->Ator)

**VideoclipProdutor** (idVideoclip->Videoclip, idProdutor->Produtor)

**PlaylistMúsica** (idPlaylist->Playlist, idMúsica->Música)

**AutorÁlbum** (idAutor->Autor, idÁlbum->Álbum)

**ConcertoAutor** (idConcerto->Concerto, idAutor->Autor)

**ConcertoMúsica** (idConcerto->Concerto, idMúsica->Música)

**GénerosFavoritos** (idUtilizador->Utilizador, idGénero->Género, númeroOrdem)

{idUtilizador, númeroOrdem} -> {idGénero}

**BandaArtista** (idBanda->Banda, idArtista->Artista)

# Formas Normais

De modo a proteger relações na Segunda Forma Normal (2NF) de sofrerem anomalias numa eventual atualização, é necessário remover dependências transitivas das mesmas, progredindo assim para um modelo na Terceira Forma Normal (3NF). Desta forma, o diagrama proposto não viola as regras da Terceira Forma Normal, na medida em que não incorre em nenhuma dependência transitiva, ou, noutras palavras, para todo o A->B não trivial, A é uma *superkey* ou B é constituído por *prime attributes*.

Por outro lado, uma relação apenas está na Forma Normal Boyce-Codd (BCNF), quando todos os seus determinantes forem uma *candidate key*, ou seja, quando para todo o A->B, A é uma *superkey*. Deste modo, o diagrama proposto não cumpre com estas regras nas relações **Utilizador** e **Género**.

**Utilizador:**

* {username} -> {id} : *username* não é uma *key* da relação.

**Género:**

* {nome} -> {id} **:** nome não é uma *key*  da relação.

# Restrições

Tendo como prioridade o funcionamento eficiente da base de dados, implementaram-se diferentes restrições (*constraints*) na criação das várias classes.

**NOT NULL** - esta restrição é associada a atributos cuja existência é obrigatória para a correta definição da classe:

* Na classe **Utilizador**, os atributos nome, *username*, e data de nascimento estão sinalizados como **NOT NULL**.
* Na classe **Música**, os atributos nome, reproduções e idAlbum foram definidos como **NOT NULL**, no entanto, os atributos imagemCapa e idVideoclip, que não são indispensáveis para a definição da classe não foram definidos desta forma.
* Na classe **Genero**, o nome não pode ser nulo.
* Na classe **Playlist**, os atributos nome e idUtilizador estão assinalados como **NOT NULL**, isto porque a definição de **Playlist** exige um Utilizador que a crie, bem como a atribuição de um nome.
* O mesmo raciocínio foi aplicado para as outras classes, por essa razão omite-se a justificação para todas as outras restrições **NOT NULL**. No entanto, explicitam-se abaixo todos os outros atributos onde esta restrição se aplica:
  + Álbum: nome
  + Editora: nome
  + Autor: nome
  + Banda: nome
  + Concerto: data, local
  + Videoclip: *url*
  + Ator: nome
  + Produtor: nome
  + GenerosFavoritos: numero Ordem

**UNIQUE** – esta restrição serve para elementos distintos de uma classe que não operem como chaves/identificadores, fica o caso em que foi usada:

* O atributo username na classe **Utilizador**, visto os nomes de utilizadores serem restritos ao primeiro *user* que o escolha.

# Restrições (continuação)

**CHECK –** esta restrição é implementada com a intenção de restringir os valores e aspetos de alguns de atributos, assim como é possível observar nos seguintes momentos:

* No atributo reproduções da classe **Música**, é verificado se o valor é maior ou igual a zero (≥0).
* No atributo url da classe **Videoclip**, confirma-se se o URL correspondente pertence à plataforma *Youtube*, analisando a *string* para a presença de “https://www.youtube.com/watch?v=”. Também, no atributo duração, verifica-se se duração ≥ 0.
* Na relação **GénerosUtilizador** verifica-se se o atributo numeroOrdem é positivo e menor ou igual a 3 (0 ≤ numeroOrdem ≤ 3), visto, por razões de simplificação da base de dados e de processamento apenas serão considerados os 3 **Géneros** favoritos do **Utilizador**.

Para além das restrições mencionadas acima, de modo a manter a integridade referencial exigida pelo nosso modelo, aplicámos chaves estrangeiras a relações/classes que relacionavam atributos de diferentes classes. Esta implementação foi feita através do método **REFERENCES**.

Um exemplo desta ideia está presente na classe **Playlist**, visto a cada **Playlist** estar associado um **Utilizador**, neste caso por meio do atributo idUtilizador.

# População

Para popular a base de dados utilizaram-se, na sua maioria, dados reais de forma à base de dados ser o mais realista possível. Assim, foi empregue uma grande quantidade de trabalho na criação de um programa em python, que conta com mais de 550 linhas de código. Foi feita a conexão com duas API’s distintas: a API do “*deezer*”, que permitiu obter dados relacionados com as musicas, artistas, álbuns, géneros e editoras, e a API do “youtube”, através da qual se obteve informação acerca dos videoclips e das suas visualizações. Uma vez que esta base de dados tem como intenção ilustrar de forma reduzida o essencial de um sistema de gerenciamento de músicas, videoclips e concertos, foi “populada” com um número reduzido de artistas. No entanto, o código python que desenvolvemos é capaz de lidar com grandes massas de dados obtidos de fontes fidedignas e reais e, caso fosse necessário, a nossa base de dados estaria apta para abarcar com toda essa informação.

Deste modo, foi despendido, apesar de não ser obrigatório, um tempo extra na função de popular a base de dados, uma vez que todos os elementos do grupo consideraram que tornaria o processo de gestão e criação de base de dados menos teórico e muito mais realista.

# Interrogações

Foram realizadas 10 interrogações de níveis de complexidade variados e que tentaram abranger o máximo de funcionalidades do SQlite possível. Para além disso, utilizou-se grande parte das classes em pelo menos uma interrogação, de forma a demonstrar de forma completa o funcionamento da base de dados.

1. Numero de álbuns que cada editora produziu

Nível de dificuldade: Fácil

Descrição: A cada editora diferente é associado o respetivo número total de álbuns que foram produzidos por ela

Funcionalidades: Utilização da associação entre **Editora** e **Álbum** e aplicação da função **COUNT** através de um **GROUP BY** dos álbuns por ID de Editora.

1. Autores que têm trabalhos (álbuns) a solo e em banda

Nível de dificuldade: Difícil

Descrição: São apresentados todos os autores que, na base de dados, são intérpretes de álbuns a solo e enquanto membros de uma banda

Funcionalidades: Primeiramente, são obtidos os ID e nomes dos membros de cada banda através da interação entre as classes **Banda**, **BandaArtista** e **Autor**. De seguida, são obtidos os álbuns desses membros individualmente através do interação das classes supracitadas com a classe **AutorAlbum**. Depois, verifica-se se esses mesmos artistas individuais têm álbuns enquanto membros de Banda. Para isso, realiza-se uma **SUB-QUERY** onde se verifica se o ID da banda pertence à tabela que resulta da junção de **Banda** com a classe **AutorAlbum**, tabela esta que contém os álbuns de cada banda.

1. Numero musicas tocadas no primeiro concerto do autor 1

Nível de dificuldade: Média

Descrição: Em relação ao artista #1, é calculado o número de músicas que são tocadas no concerto mais antigo (com menor data)

Funcionalidades: Primeiro, é explicitada a relação entre **Concerto** e **ConcertoAutor**, que permite obter os múltiplos autores de cada concerto e respetiva data de realização. De seguida, é limitada a seleção aos concertos com o autor #1 e é calculada a data do concerto mais antigo através da função **MIN**(data). Obtém-se assim o ID do concerto com menor data, que será utilizado para descobrir o numero de músicas nele tocadas, através da tabela **ConcertoMusica** e da função **COUNT**.

1. Playlists com musicas de autores todos diferentes

Nível de dificuldade: Difícil

Funcionalidades: Primeiramente, é realizada a junção entre as classes **PlaylistMusica**, **Musica** e **AutorAlbum**, uma vez que a partir da 1º classe é possível obter as músicas pertencentes a cada playlist, de seguida obtém-se através da 2º classe os álbuns a que cada música pertence e, por fim, obtém através da 3º classe os autores de cada álbum, ficando assim com a relação Playlist-Autores explicitada. De seguida, através de um **GROUP BY** por ID de playlist, são **contados** os autores diferentes em cada playlist (através do argumento **DISTINCT**) e também o número de autores total incluindo repetições. Quando estas duas contagens são iguais, significa que a Playlist contém autores todos diferentes. Por fim, realiza-se uma junção com a classe **Playlist** para obter os nomes das playlists.

1. Géneros favoritos do utilizador 1 que também são favoritos do utilizador 2

Nível de dificuldade: Fácil

Funcionalidades: É realizada a **intersecção** entre os nomes dos géneros favoritos do utilizador 1 e utilizador 2. Para obter os géneros de cada utilizador explicita-se a relação entre **GenerosFavoritos** e **Genero**.

1. Numero médio de pessoas (produtores e atores) que participam num videoclip em Itália

Nível de dificuldade: Difícil

Funcionalidades: Primeiramente é realizada a junção entre as classes **Videoclip** e **VideoclipAtor** para obter os atores que participam em cada ator em Itália e através de um **GROUP BY** por ID de Videoclip realiza-se uma **contagem** do número de participantes atores por Videoclip.

Aplica-se um raciocínio análogo para a classe **VideoclipProdutor**.

De seguida, é aplicada uma **UNION ALL** para adicionar na mesma tabela os valores referentes ao nº de atores e produtores de cada videoclip. Procede-se à soma destas duas contagens através de um **SUM** aplicado aos participantes por ID de videoclip. Por fim, é aplicada a função **AVG** ao nº de participantes.

1. Videoclips das músicas com a palavra "Love"

Nível de dificuldade: Fácil

Descrição: Esta query foi proposta com o propósito de demonstrar a qualidade dos dados com que foi populada a nossa base de dados. Para cada videoclip, os url obtidos podem ser testados para ver como correspondem aos videoclipes corretos associados a cada música.

Funcionalidades: É explicitada a relação entre as classes **Musica** e **Videoclip**, de forma a obter o videoclip associado a cada música. É utilizado o operador **LIKE** juntamente com o caractere especial de string **%** para encontrar as músicas que contêm a palavra Love no seu título.

1. Utilizadores do tipo "Indie Teenager" (última musica ouvida tem menos de 150 reproduções e nascido depois do ano 2000)

Nível de dificuldade: Fácil

Funcionalidades: Dá-se uso à associação entre a classe **Utilizador** e **Música**, uma vez que a cada utilizador está associado um ID de última música ouvida que remete para uma determinada música. Aplica-se uma restrição de seleção em que o número de reproduções dessa música tem de ser menor que 150 e outra que verifica se a data de nascimento é maior que 2000. Esta última restrição permite verificar a correta implementação de datas.

1. Top 3 artistas mais populares em playlists

Nível de dificuldade: Média

Descrição: Top 3 artistas com um maior número de suas músicas nas playlists

Funcionalidades: Inicialmente é realizada a junção entre 3 tabelas: a **PlaylistMusica**, **AutorAlbum**, **Autor**, para poder obter os autores de cada música de cada playlist. De seguida é realizado um **GROUP BY** por ID de playlist, utilizado um **COUNT** do número de músicas por playlist, um **ORDER BY** por numero de músicas e por ordem descendente e um **LIMIT** de 3.

1. Idade dos elementos cuja última música ouvida é a última musica mais ouvida global

Nível de dificuldade: Média

Funcionalidades: Inicialmente é feita uma **contagem** de utilizadores por ID de última música ouvida. De seguida, é calculado o valor máximo do nº de utilizadores ouvintes de uma determinada música. De seguida, o valor obtido nessa interrogação é utilizado como condição para encontrar os utilizadores que têm essa música como última ouvida e a idade desses utilizadores é calculada através da função **STRFTIME**.

# Diagrama UML Revisto

