**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

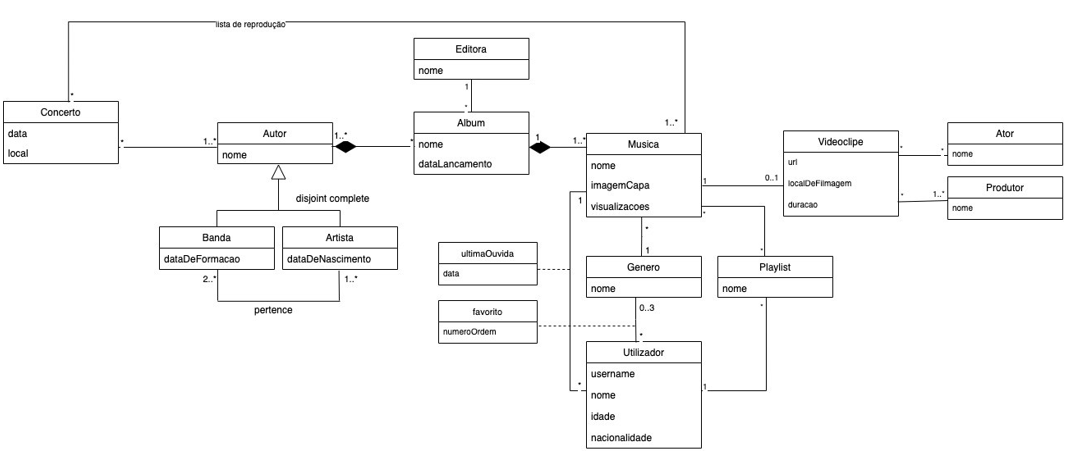
Uma empresa organizadora de eventos, especializada na gestão de festivais e concertos, pretende entender melhor as preferências audiovisuais da população jovem. Desta forma, a empresa, na organização do próximo festival, denominado “Paços de Dança”, que decorrerá durante o mês de agosto em Paços de Ferreira, 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.

# Diagrama UML Revisto

