

Bases de Dados

*Gestão de Stream de Música*

*2MIEIC01 – Grupo 102*

*(12 de março de 2017)*

Bárbara Silva  **up201505628**@fe.up.pt

Julieta Frade **up201506530**@fe.up.pt

Miguel Fernandes **up201503538**@fe.up.pt

Descrição

Este projeto baseia-se na gestão de um serviço de *stream* de música, semelhante ao *Spotify*.

Em primeiro lugar, consideremos a classe **Utilizador** e suas inter-relações.  
Um usuário da plataforma tem a possibilidade de seguir outros utilizadores e **Intérpretes**, sendo esta última relação não-recíproca. É-lhes também proporcionado um serviço de troca de **Mensagens** – instanciou-se uma classe para memorização da sua data de envio e conteúdo.

A base de dados armazena uma quantidade de dados considerável relativamente ao **Utilizador**, como o seu id, *username*, foto de perfil e idade – um atributo derivado, calculado através da sua data de nascimento.

Cada **Utilizador** pode guardar diferentes **Dispositivos**,como telemóvel, computador ou tablet para poder usufruir do serviço em diferentes locais, sendo, obviamente, obrigatório o registo de pelo menos um.

Caso o usuário procure mais funcionalidades, existe ainda a opção de elevar o estado da conta para **Utilizador Premium**, à custa de uma mensalidade fixa. Este *upgrade* inclui a remoção de anúncios, descarregamento de músicas e tempo de reprodução ilimitado. Para distinguir os dois modelos de conta, idealizamos duas relações de generalização. A classe **Utilizador Free** mantém registo do tempo de audição decorrido; a classe **Utilizador Premium** guarda o valor da mensalidade, dado que diferentes planos de pagamento poderão resultar em tarifas distintas. Ambas agem como extensão da classe-mãe **Utilizador**.

De seguida, examinemos o conceito de **Playlist**. Esta estrutura possui um nome, imagem e descrição, todos atribuídos por um utilizador do sistema.

Assim, de facto, uma lista de reprodução estabelece duas relações particulares com um usuário: a de propriedade – um cliente pode ser dono de várias playlists, porém cada uma destas possui um só dono – e a de acompanhamento – um cliente pode seguir várias playlists distintas e, do mesmo modo, a playlist poderá ser seguida por vários clientes diferentes.

Uma **Playlist** é, evidentemente, constituída por **Músicas**, uma estrutura com nome, duração, número de reproduções e um **Género** musical predominante associados. Considerou-se fazer sentido, do ponto de vista conceptual, a noção de playlist apenas ser válida quando esta não se encontra vazia, justificando a multiplicidade 1..\* utilizada.

Introduzido o conceito de **Música** nesta abstração, abordemos as suas ligações com as outras classes da nossa *database*. Uma música estabelece uma relação de composição com **Álbum**, gerando uma classe de associação no processo, que preserva a sua posição na lista de faixas. Evidencie-se o uso deste tipo de associação, que constringe uma música a unicamente um álbum. Não se considerou a existência de coletâneas por se tratar de um caso excessivamente particular e comprometer a interpretação lógica e imediata da estrutura.

Descrição (continuação)

Iremos agora aprofundar o conceito de **Intérprete**, cuja estrutura é composta por um nome, uma *flag* de verificado – que se trata de um booleano ativado para artistas que verificam o seu perfil – foto de perfil, de capa e biografia. Esta classe tem relações intrínsecas com duas outras: **Cidade** e **Concerto.** Por sua vez, **Cidade** está associada a **País**.

É relevante apontar que, do mesmo modo que **Música** e **Álbum** estabelecem, entre si, uma relação de composição, também **Álbum** e **Intérprete**. Foi assumido que um grupo apenas poderá ser considerado, efetivamente, um grupo musical, se tiver produzido algum álbum. Note-se que se considerou um **Single** como um tipo de álbum. De tal forma, o conceito de **Intérprete** seria infundado até a existência de uma discografia, sendo justificada a relação.

Dada a natureza associativa destas três classes, deduzimos que seria lógico integrá-las num tuplo. De tal forma, um **Intérprete** poderá ter (ou não) **Concertos** agendados para o futuro, cada um destes armazenando uma data e realizando-se numa única **Cidade**. Para completar o tuplo, basta associar **Intérprete** à sua **Cidade** de origem.

Por último, o sistema gera uma lista ordenada das músicas mais ouvidas numa dada **Cidade**, o **Top** de músicas, no qual cada uma terá uma posição, ou seja, uma classificação entre todas as outras presentes no mesmo. Assim, associamos **Cidade** e **Música**, a partir de **Top**.

Atributos

Utilizador

* Nome
* *Username*
* *Password*
* Foto de Perfil
* Data de Nascimento
* *E-Mail*
* / Idade

Utilizador Premium

* Mensalidade

Utilizador Free

* Tempo Limite

Mensagem

* Conteúdo
* Data de Envio

Dispositivos

* Nome

Tipo de Dispositivo

* Nome do Tipo

Playlist

* Nome
* Imagem
* / Duração
* Descrição

Intérprete

* Nome
* Verificação
* Foto de Perfil
* Foto de Capa
* Biografia

Álbum

* Nome
* Capa
* Ano

Tipo de Álbum

* Nome do Tipo

Música

* Nome
* Duração
* Reproduções
* Género

Género

* Nome

Concerto

* Data

Local

* Cidade
* País

Top

* Posição

Diagrama Relacional e Dependências Funcionais

***Utilizador*** *(id, nome, username, password, fotoPerfil, dataNascimento, email)*

1. {id} -> {nome, username, password, fotoPerfil, dataNascimento, email, idade}

***UtilizadorFree****(id->Utilizador, tempoLimite)*

1. {id->Utilizador} -> {tempoLimite}

***UtilizadorPremium****(id->Utilizador, mensalidade)*

1. {id->Utilizador} -> {mensalidade}

***Mensagem*** *(id, conteúdo, dataEnvio)*

1. {id}->{conteúdo, dataEnvio}

***Envia****(id->Mensagem, idUser->Utilizador)*

***Recebe****(id->Mensagem, idUser->Utilizador)*

***SegueUtilizador*** *(idUser->Utilizador, idUserSeguido->Utilizador)*

***SegueIntérprete*** *(idUser->Utilizador, idIntérprete->Intérprete)*

***SeguePlaylist*** *(idUser->Utilizador, idPlaylist->Playlist)*

***Intérprete****(id, nome, verificado, fotoPerfil, fotoCapa, biografia, idCidade->Cidade)*

1. {id}->{nome}
2. {nome}->{verificado, fotoPerfil, fotoCapa, biografia, idCidade}

***Álbum*** *(id, nome, capa, ano, idTipoAlbum->TipoÁlbum, idIntérprete->Intérprete)*

1. {id}->{nome, capa, ano, idTipoAlbum->TipoÁlbum, idIntérprete}

***TipoÁlbum*** *(idTipoAlbum, tipoNome)*

1. {idTipoAlbum} ->{tipoNome}
2. { tipoNome } ->{idTipoAlbum }

***Música*** *(id, nome, duração, reproduções)*

1. {id} -> {nome, duração, reproduções}

***Género*** *(idGénero, nome)*

1. {idGénero} -> {nome}
2. {nome}->{idGénero}

Diagrama Relacional e Dependências Funcionais (continuação)

***MusicaAlbum*** *(idMúsica->Música, idÁlbum->Álbum, índice)*

1. {idMúsica} -> {idÁlbum, índice}
2. {idÁlbum, índice}->{idMúsica}

***Playlist*** *(id, nome, imagem, duração, descrição, idDono->Utilizador)*

1. {id} -> {nome, idDono}
2. {nome, idDono}->{imagem, duração, descrição}

***MusicaPlaylist*** *(idMusica->Música, idPlaylist->Playlist)*

***Dispositivo*** *(idDispositivo, nome, idTipoDispositivo->TipoDispositivo, idUser->Utilizador)*

1. {idDispositivo} -> {nome, idTipoDispositivo->TipoDispositivo, idUser->Utilizador}

***TipoDispositivo*** *(idTipoDispositivo, tipoNome)*

1. {idTipoDispositivo}->{tipoNome}
2. {tipoNome}->{idTipoDispositivo}

***Concerto*** *(idConcerto, data, id->Intérprete, idCidade->Cidade)*

1. {idConcerto}->{data, id, idCidade }

***Cidade(****id, nome, idPaís->País)*

1. {id}->{nome, idPaís}
2. {nome, idPaís}->{id}

***País*** *(id, nome)*

1. {id}->{nome}
2. {nome}->{id}

***Top*** *(idMúsica->Música, idCidade->Cidade, posição)*

1. {idMúsica, idCidade} -> {posição}
2. {idCidade, posição}->{id}

Formas Normais

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.

O modelo proposto viola estas normas nas relações **Playlist** e **Intérprete**. Em **Playlist** verifica-se na medida em que, através do identificador da playlist é possível chegar ao nome desta e identificador do dono. Por sua vez, através destes dois atributos, existe uma ligação aos atributos restantes – imagem, duração e descrição. Deste modo, haverá conexão indireta entre estes dois conjuntos de atributos, sustentando-se uma relação de transitividade. Em **Intérprete** é porque através do identificador do intérprete podemos chegar ao seu nome, e do nome ao resto dos atributos.

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. Abordemos talvez os casos menos triviais da restrição no modelo:

* Os atributos *nome*, *capa* e *ano* da classe Álbum são declarados como NOT NULL, pois o conceito de álbum não poderia subsistir sem a existência destas propriedades.
* Os atributos *nome* e *duração* da classe Playlist são imperiais para uma apresentação *user-friendly* da lista de reprodução (ID age como identificador, porém não é intuitivo para o usuário visualizar as suas playlists através de números identificativos) e para assegurar que a lista de reprodução não se encontra vazia. A *imagem* e *descrição* acabam por ser opcionais, dado que a sua não-inicialização não inibem de nenhum modo o correto e intuitivo funcionamento da playlist.

A restrição UNIQUE foi especialmente aplicada a atributos identificadores de uma certa classe, embora estes não operem como chaves. Fica como exemplo:

* Atributo *nome* da classe Género – não existem dois estilos musicais com a mesma denominação.
* Atributo *nome* da classe Intérprete – a base de dados não permite o registo de artistas com nomes repetidos.
* Atributo *tipoNome* da classe TipoÁlbum – não existem dois tipos de álbum com a mesma denominação.
* Atributo *tipoDispositivo* da classe TipoDispositivo – não existem dois tipos de dispositivo com a mesma denominação.

Por outro lado, a restrição CHECK haverá sido praticada com a mentalidadede de restringir certos aspetos de atributos, aliando uma maior segurança a um melhor controlo dos dados. A listagem de usos inclui:

* O atributo *ano* na classe Álbum tem um limite mínimo de 1920 para minimizar problemas de memória de armazenamento, assim como qualidade.
* O atributo *biografia* na classe Intéprete tem um limite de 300 carateres, para evitar blocos de texto demasiado extensos.
* O atributo *conteúdo* na classe Mensagem está limitado a 140 carateres.
* O atributo *duração* na classe Música não poderá exceder 3600 unidades (segundos) e não deverá ser inferior ou igual a 0 unidades.
* O atributo *posição* na classe Top deverá ser inferior ou igual a 100, visto apenas existirem exatamente cem elementos a qualquer altura na lista de faixas.
* Por motivos de segurança, tanto o atributo *username* como *password* da classe Utilizador deverão ter comprimentos situados entre 6 e 12.

Diagrama de Classes – UML

