U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Bases de Dados

Gestão de Stream de Música

2MIEIC01 – Grupo 102

(2 de Abril 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

Top

- Posição

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

Cidade

- Nome

País

- Nome

Diagrama Relacional e Dependências Funcionais

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

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

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

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

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

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

***Mensagem** (id, conteúdo, dataEnvio, idEmissor->Utilizador, idRecetor->Utilizador)*

1. {id, idEmissor, idRecetor}->{conteúdo, dataEnvio}

***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, idUser }

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 mentalidade 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.

Restrições (continuação)

Por último, relativamente a restrições de integridade referencial, aplicámos chaves estrangeiras a classes intrinsecamente relacionadas com outras.

Tome-se como exemplo a classe **Álbum**. Dado o facto de a cada álbum estar associado um tipo único, justifica-se o emprego de uma chave estrangeira apontada para **TipoÁlbum**. Da mesma forma, um **Álbum** pertence a um **Intérprete**, pelo que haverá que existir uma ligação através de integridade referencial entre ambas as classes.

Este mesmo raciocínio foi aplicado nas restantes classes que dispõem destas restrições, resultando num modelo bastante trivial de interpretar.

Interrogações

De seguida, apresenta-se uma lista de queries que se achou pertinente para a comprovação do correto funcionamento de vários componentes e suas conexões na base de dados.

1. Duração de álbuns com intérprete de ID #4: verificação das interligações entre **Intérpretes**, **Álbuns**, **MúsicaÁlbum** e **Música**, utilizando a diretiva SUM que adiciona sucessivamente a duração de cada música presente em cada álbum do artista requisitado. Por motivos de teste, utilizou-se o ID #4, que fornece os dados mais variados.
2. Número de concertos do intérprete com nome Twenty One Pilots no Reino Unido: verificação da associação da classe **Cidade** com **Concerto** e, por sua vez, com **Intérprete**. Dá uso ao método COUNT para manter registo do número de concertos do artista.
3. IDs dos utilizadores premium que usam telemóvel: verificação da associação da classe **Dispositivo** com **Utilizador**.
4. Média das posições das músicas da playlist com ID #4 no top de todas as cidades: verificação da associação da classe **Top** com **MúsicaPlaylist** e, por sua vez, **Música**. Utilizou-se a função matemática AVG, que realiza o cálculo da média da posição das músicas da lista de reprodução nos tops de cada cidade.
5. Top 5 dos intérpretes mais seguidos: verificação da associação **SegueIntérprete**, que liga **Utilizador** à classe **Intérprete**. Ordenação descendente, de acordo com a popularidade, fazendo uso da diretiva COUNT. É limitado por 5 posições, através do parâmetro LIMIT 5.
6. Playlists que contêm músicas dos Radiohead: verificação das interligações entre **MúsicaPlaylist** e **MúsicaÁlbum**, assim como a tentativa de correspondência do **Intérprete** pesquisado. Neste caso, 'Radiohead'.
7. A música em 6ª posição em cada cidade: verificação da associação **Cidade** com **Top**. A lista é agrupada pelo atributo nome de cada cidade.
8. Número de álbuns de estúdio de cada artista: verificação da associação das classes **Intérprete** e **TipoÁlbum** e, por sua vez, **Álbum**. Ordenado de forma descendente, estabelece a contagem de um tipo de álbum específico. Neste caso, utiliza-se o ID #1, relativo a álbuns de estúdio.
9. Artistas com concertos marcados no verão do ano 2017: verificação da relação **Intérprete** e **Concerto**, constatando o bom funcionamento do atributo data da classe Concerto, do tipo date, encapsulando a data pesquisada por entre duas outras, dando uso a parâmetros BETWEEN.
10. Número de músicas de cada playlist: verificação da relação **Playlist** e **MúsicaPlaylist**, estabelecendo uma contagem de faixas presentes na lista de reprodução, agrupados pelo nome da lista de reprodução.

Gatilhos

Por fim, foram definidos 4 gatilhos que são úteis para a monitorização e manutenção da base de dados, dois dos quais estão no mesmo ficheiro.

1. AdicionaUtilizador

Após ser criado um novo **Utilizador** (*AFTER INSERT*), é automáticamente criado um **Dispositivo** e um **UtilizadorFree**, isto é, sempre que um utilizador é adicionado à base de dados terá que estar associado a um dispositivo e a um tipo de conta, que por definição é UtilizadorFree.

2. ValidaDataNascimento

No intuito de alterar o argumento *dataNascimento* de um dado **Utilizador** (*BEFORE UPDATE*), a data inserida no formato 'AAAA-MM-DD' é validada ou não, isto é, uma data de nascimento só é válida se o utilizador tiver pelo menos 10 anos de idade, assim, foi implementado um mecanismo de verificação.

Este mecanismo consiste em aplicar a função **julianday** tanto à data atual como à de nascimento que foi inserida. Esta função recebe uma data e retorna o número de dias que passaram desde 24 de novembro de 4714 AC. Assim, sempre que o retorno de *julianday*(data de nascimento) for maior que *julianday*(data atual) menos 3650 (como estamos a trabalhar com dias, um ano tem 365 e como queremos que o utilizador tenha pelo menos 10 anos, multiplicamos os dias do ano por 10), a data de nascimento inserida será inválida.

Em suma, caso a data de nascimento seja inválida, este gatilho aborta a atualização da data e mostra uma mensagem de erro.

3. AdicionaMusicaPlaylist e RemoveMusicaPlaylist

Quanto ao primeiro gatilho, após ser adicionado um elemento a **MusicaPlaylist** (*AFTER INSERT*), ou seja, uma **Musica** foi adicionada a uma **Playlist**, sendo **MusicaPlaylist** a classe que as liga, o argumento *duracao* de **Playlist** é incrementado com a duração da música que foi inserida.

De forma a organizar ainda mais a manutenção da nossa base de dados, adicionamos neste ficheiro ainda outro gatilho, ***RemoveMusicaPlaylist***, que consiste em fazer exatamento o oposto do primeiro, isto é, quando um elemento é apagado de **MusicaPlaylist** (*AFTER DELETE*), o argumento *duracao* de Playlist é decrementado com a duração da música removida.

Instruções de execução

Estes passos deverão ser seguidos escrupulosamente, para evitar comportamento inesperado por parte da base de dados.

- Executar o ficheiro ***create.sql***.
- Executar o ficheiro ***populate.sql***.
- Correr as interrogações e gatilhos como pretendido, que se encontram nas pastas ***Queries*** e ***Triggers***, respetivamente, no diretório principal da entrega.

Esta hierarquia de passos advém do facto das interrogações e gatilhos dependerem da correta população das tabelas declaradas no ficheiro ***create.sql***.

Diagrama de Classes – UML