Universidade do Minho
Mestrado Integrado em Engenharia Informática
Licenciatura em Ciências da Computação

Unidade Curricular de Bases de Dados

Ano Letivo de 2017/2018

Arpeggio Music – Serviço de Música

Inês Sampaio, João Mourão, Pedro Almeida, Rui Vieira

Novembro, 2017

BD

Data de Recepção	
Responsável	
Avaliação	
Observações	

Arpeggio Music – Serviço de Música Digital

Inês Sampaio, João Mourão, Pedro Almeida, Rui Vieira

Novembro, 2017

<</opcional Dedicatória>>

Resumo

Este relatório demonstra o desenvolvimento de uma base de dados para uma aplicação de disponibilização de música em formato digital (Arpeggio Music). Aqui serão apresentados todos os passos necessários para a realização deste trabalho, desde a criação do Modelo Conceptual até ao Modelo Físico.

Inicialmente é feita uma pequena introdução do projeto, sendo esta apresentada em conjunto com a contextualização, o caso de estudo, a motivação e principais objetivos, bem como a análise de viabilidade do processo.

Posteriormente, realiza-se o levantamento e a respetiva análise de requisitos, apresenta-se o método de levantamento e de análise de requisitos que foi adotado, seguido dos requisitos levantados propriamente ditos, separados pelo seu tipo (requisitos de descrição, de exploração ou de controlo). Após a enumeração dos requisitos é feita uma análise geral dos mesmos.

Na apresentação do Modelo Conceptual são descritas as entidades, os relacionamentos envolvidos e os respetivos atributos. Para cada entidade expõe-se informação sobre a sua descrição e ocorrência, além de serem identificadas as chaves primária e alternativas. Para cada atributo especifica-se o domínio de valores possíveis, e terminamos com a validação do modelo de dados pelo utilizador.

Posto isto, é feita a transição do Modelo Conceptual para Lógico. Este último, depois de estruturado, é validado com base nas regras de normalização, até à terceira forma normal, e das interrogações do utilizador. Faz-se então uma nova validação do modelo, sendo apresentadas as transações mais importantes do sistema.

Surge depois o Modelo Físico, e expõe-se a tradução do Modelo Lógico para um Sistema de Gestão de Base de Dados (SGBD), totalmente em SQL. Algumas interrogações do utilizador são explicitadas nesta fase, bem como as transações estabelecidas que anteriormente haviam sido traduzidas para SQL. Após calculada uma estimativa do espaço em disco que poderá ser usado pela base de dados, terminamos com a definição das vistas de utilização e caracterização de alguns mecanismos de segurança.

As conclusões do projeto são expostas por fim, e descrevem-se algumas formas para estender o trabalho realizado no futuro.

Área de Aplicação: Desenho e Arquitetura de Sistemas de Base de Dados relativamente ao uso de uma aplicação responsável pela disponibilização de música em formato digital por todo o mundo.

Palavras-Chave: Base de Dados, Base de Dados Relacionais, Análise de Requisitos, Modelo Conceptual, Modelo Lógico, Modelo Físico, MySQL WorkBench, SQL.

Índice

Índice de Tabelas

1. Definição do sistema

1.1. Contexto de aplicação do sistema

Streaming é uma forma de distribuição digital frequentemente utilizada para distribuir conteúdo multimédia através da Internet, que retira a necessidade de o utilizador armazenar todo este conteúdo no seu computador. O conteúdo que pode ser distribuído por este método varia imenso, sendo um dos mais populares a música.

O início do streaming de música deu-se em Janeiro de 1993, o primeiro serviço lançado tinha o nome de *Internet Underground Music Archive* (IUMA), e permitia a músicos que não estivessem ligados a empresas discográficas, partilhar a sua música com os seus fãs.

Com o passar dos anos estes serviços foram aumentando a sua complexidade, permitindo que os seus utilizadores criassem playlists personalizadas, ouvissem os seus artistas favoritos e conhecessem novos tipos de música, tudo isto sem realizar nenhum pagamento (mas ouvindo publicidade), ou pagando por uma conta Premium (sem publicidade e com mais funcionalidades).

Em 2017 nasceu um novo serviço de streaming, Arpeggio Music, criada em Portugal com a finalidade de permitir que todo o mundo consiga ouvir música de forma legal por um preço mais acessível que o mercado oferecia. Para tornar o seu serviço mais atrativo, a Arpeggio Music criou uma subscrição com pagamento mensal de 5€ para até 5 utilizadores da mesma família, enquanto que para um utilizador individual o preço mensal rondaria os 2,5€.

O objetivo desta nova empresa é chegar a todos os pontos do globo, melhorando a forma como ouvimos música.

A base de dados que pretendemos implementar focar-se em todo o sistema de um serviço de streaming de música. De maneira a facilitar o entendimento deste processo, damos como exemplo o Spotify, um dos maiores serviços de streaming de música a nível global, com todas as funcionalidades que uma boa plataforma de distribuição de música deve ter, como a criação de playlists pessoais, a possibilidade de seguir os artistas favoritos e uma vasta biblioteca de músicas.

Vamos então realizar uma pequena base de dados de uma plataforma de distribuição de música, de dimensão considerável, que nos permitirá ter acesso a informação relativa ao serviço através de interrogações ao sistema, descrita numa lista de requisitos que apresentamos mais afrente.

1.2. Fundamentação da implementação da base de dados

A criação deste sistema tem como principal motivação facilitar o acesso, a inserção e a modificação de informação para uma gestão de recursos mais eficiente. Considerando a contextualização apresentada, sabemos que o sistema deste serviço é extremamente complexo sendo necessário manter toda a informação completa, atualizada e sempre disponível.

Devido à complexidade do sistema, é fundamental, para o correto funcionamento da plataforma de distribuição de músicas, uma base de dados bem construída.

O maior objetivo desta base de dados é simplificar e garantir a organização correta de toda a informação relativa ao serviço. Para tal, é necessário analisar todas as possibilidades à disposição do utilizador. Portanto, a base de dados foi construída mantendo um foco nos problemas que surgem regularmente aos utilizadores.

1.3. Análise da viabilidade do processo

Ao refletir sobre a implementação, ou não, de uma Base de Dados (BD), é fundamental que se analise a viabilidade do processo. Quando elaborada corretamente, uma BD traz inúmeras vantagens, das quais destacamos as seguintes:

- **Integridade dos dados**

  A BD não permite redundância, evitam-se assim possíveis conflitos entre diferentes versões da mesma informação.

- **Resposta rápida a interrogações**

  As interrogações feitas pelo utilizador serão respondidas de uma forma rápida e correta, mesmo que estas sejam extremamente complexas.

- **Organização da informação**

  Devido à forma como os dados estão organizados, é possível descobrir facilmente onde está localizada uma específica porção de informação.

Por consequência, é possível deduzir que a implementação da BD para este projeto tem como objetivo facilitar e melhorar o processamento e armazenamento da informação que por ela é guardada, e assim permitir uma melhor organização e um acesso simplificado a mesma informação.

2. Levantamento e análise de requisitos

2.1. Método de levantamento e de análise de requisitos adotado

Para ser possível perceber que funcionalidades e que tipo de informação a base de dados deveria conter, começamos por realizar uma reunião com o nosso cliente, de forma a levantar os requisitos que ele queria que tivéssemos em conta. Além deste levantamento de dados, analisamos também um outro grande serviço de música e realizamos entrevistas com os seus diversos utilizadores.

Após uma análise dos requisitos levantados do cliente, conseguimos perceber que seria necessário cumprir um conjunto de requisitos que apresentaremos nas seções seguintes.

2.2. Requisitos levantados

Os requisitos levantados neste projeto podem ser divididos em três grupos: requisitos de descrição, exploração e controlo, que serão explicitados seguidamente.

2.2.1. Requisitos de descrição

Utilizador

- Para se identificar, para além do seu username e um e-mail, necessita também de indicar o seu nome, data de nascimento, país e número de telemóvel.
- Possui ainda um tipo, que depende da natureza da sua subscrição, podendo ser Free ou Premium.

Playlist

- Deve ser identificada através do nome.
- Deve conter várias informações, tais como a sua descrição e duração total.

Faixa

- Deve ser identificada pelo nome.
- Contém diversas informações, como o seu género, data, duração e número de reproduções.

Artista

- Deve ser identificado através do nome.

- Deve conter várias informações, tais biografia e ranking (de artistas mais ouvidos).

Álbum

- Deve ser identificado através do nome.
- Contém informações, como data e duração.

2.2.2. Requisitos de exploração

- Deve ser possível procurar um utilizador pelo seu username, email, telemóvel ou id.
- Deve ser possível procurar um artista pelo seu nome ou id.
- Deve ser possível procurar uma faixa pelo seu nome ou id.
- Deve ser possível procurar um álbum pelo seu nome ou id.
- Deve ser possível procurar uma playlist pelo seu nome ou id.
- Deve ser possível procurar todas as músicas e álbuns de um artista.
- Deve ser possível procurar músicas por género.

2.2.3. Requisitos de controlo

Utilizador

- Tem como atributo um número de identificação único.

Playlist

- Tem como atributo um número de identificação único.

Artista

- Tem como atributo um número de identificação único.

Álbum

- Tem como atributo um número de identificação único.

Faixa

- Tem como atributo um número de identificação único.

2.3. Análise geral de requisitos

Foi realizada uma reunião com o responsável da Arpeggio Music onde apresentamos e discutidos os diversos requisitos. Ao ser dada a aprovação por parte do cliente, partiu-se para a execução do Modelo Conceptual da BD.

3. Modelação conceptual

3.1. Apresentação da abordagem de modelação realizada

Existem duas possíveis abordagens de modelação, a abordagem centralizada e a abordagem de integração de vistas. A primeira recolhe os requisitos de cada vista e junta tudo antes de passar para a fase de modelação, criando a modelação em apenas uma vista. A segunda mantém as vistas separadas e faz a modelação de acordo com isso. Apenas se justifica usar esta última abordagem quando existem diferenças significativas entre cada vista e quando a complexidade da informação é grande.

Perante estas características, definimos a utilização da abordagem centralizada, tendo como base de decisão o facto de a complexidade da BD não ser suficiente para justificar um outro tipo de abordagem.

3.2. Identificação e caracterização das entidades

Após a análise de requisitos, foi possível deduzir as entidades essenciais para a construção da BD.

Tabela 1: Descrição e caracterização de entidades

Entidades	***Descrição***	***Aliases***	***Ocorrência***
Utilizador	Termo que descreve todos os utilizadores do sistema	Cliente	Cada utilizador tem playlists associadas e podem seguir artistas e playlists
Artista	Termo que descreve todos os artistas do sistema	Compositor	Cada artista tem faixas e álbuns associados e podem ser seguidos por utilizadores do sistema
Playlist	Termo que descreve todas as playlists do sistema	Lista	Cada playlist tem faixas associadas e podem ser criadas pelos utilizadores
Faixa	Termo que descreve todas as faixas do sistema	Música	Cada faixa pertence a uma/várias playlist(s), a um álbum e a um artista
Album	Termo que descreve todos os álbuns do sistema	Coletânea	Cada álbum tem faixas e artistas associados

3.3. Identificação e caracterização dos relacionamentos

Tendo identificado as entidades, passamos agora à próxima fase, identificar os relacionamentos existentes entre elas.

Destacamos a entidade Faixa como sendo a entidade mais importante neste modelo, devido aos vários relacionamentos que tem com outras entidades.

Entre a entidade **Faixa** e a entidade **Playlist** existe um relacionamento. A cardinalidade deste relacionamento é **N:M** pois em uma Playlist existem **N Faixas** e uma Faixa está em **M Playlists**.

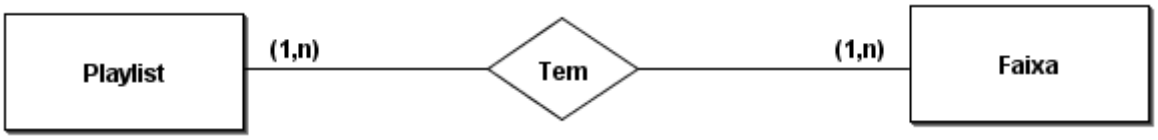


Ilustração 1: Relacionamento Playlist - Faixa

A entidade **Faixa** está também relacionada com o **Álbum**, isto porque a **um Álbum** pertencem **N Faixas**, sendo assim a cardinalidade é de **1:N**.

Ilustração 2: Relacionamento Faixa - Album

O último relacionamento da entidade **Faixa** é com a entidade **Artista**. **Um Artista** tem **N Faixas**, logo a cardinalidade é de **1:N**.

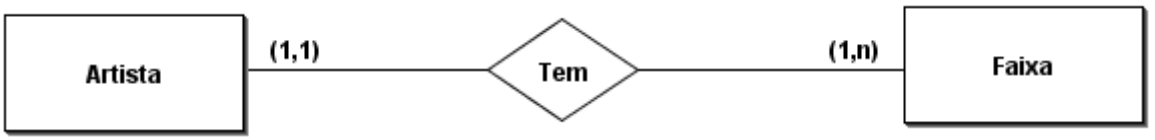


Ilustração 3: Relacionamento Faixa - Artista

A entidade **Artista** tem um relacionamento com a entidade **Álbum**. A cardinalidade deste relacionamento é de **1:N**, devido a **um Artista** ter **um ou mais Álbuns**.

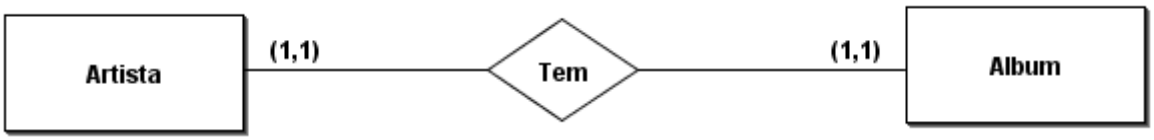


Ilustração 4: Relacionamento Artista - Album

O último relacionamento da entidade Artista é com a entidade Utilizador. Tem cardinalidade de **N:M**, devido a possibilidade de um Utilizador seguir **um ou vários Artistas** e de **um Artista** ser seguido por **um ou mais Utilizadores**.

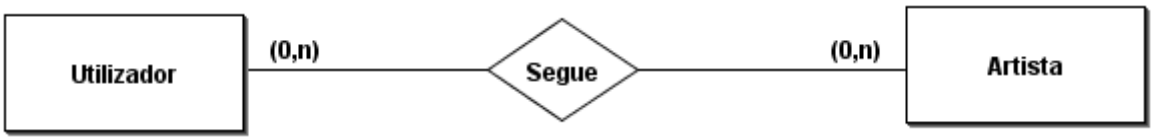


Ilustração 5: Relacionamento Artista - Utilizador

Entre a entidade **Utilizador** e a entidade **Playlist** existem dois relacionamentos. **Um Utilizador** cria **N Playlists**, este relacionamento tem cardinalidade de **1:N**. O outro relacionamento com a entidade **Playlist** é a eventualidade de **um Utilizador** seguir **uma ou mais Playlists** e de **uma Playlist** poder ser seguida por **diversos Utilizadores**. A cardinalidade deste relacionamento é de **N:M**.

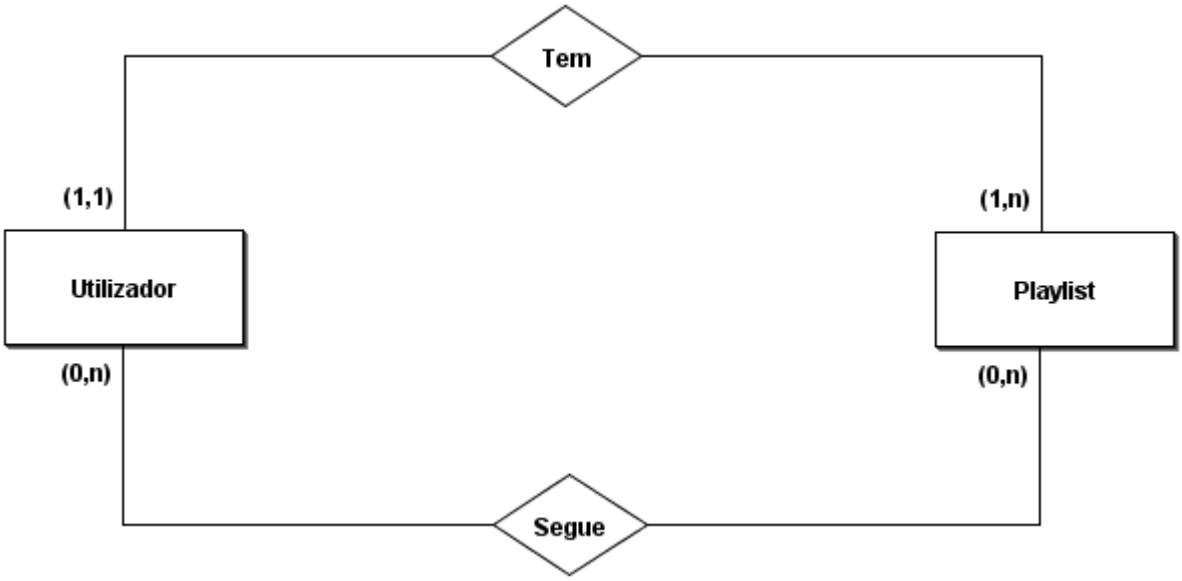


Ilustração 6: Relacionamento Utilizador - Playlist

Tabela 2: Quadro resumo de relacionamentos

Entidade	***Multiplicidade***	***Relacionamento***	***Multiplicidade***	***Entidade***
Utilizador	1	Tem	N	Playlist
Utilizador	1	Segue	N	Playlist
Utilizador	N	Segue	N	Artista
Playlist	1	Tem	N	Faixa
Faixa	N	Pertence	1	Album
Artista	1	Tem	N	Faixa
Artista	1	Tem	N	Album

3.4. Identificação e caracterização da associação dos atributos com as entidades e relacionamentos

Tabela 3: Associação dos atributos às entidades e relacionamentos

Entidade	*Atributo*	*Data type & length*	*Null*	*Multivalor*	*Derivado*
Utilizador	IdUtilizador(PK)	INT	Não	Não	Não
	Username	VARCHAR(45)	Não	Não	Não
	Data Nascimento	DATE	Não	Não	Não
	Pais	VARCHAR(45)	Não	Não	Não
	Telemovel	INT	Sim	Não	Não
	Email	VARCHAR(45)	Não	Não	Não
	Tipo	VARCHAR(45)	Não	Não	Não
Artista	IdArtista(PK)	INT	Não	Não	Não
	Nome	VARCHAR(45)	Não	Não	Não
	Biografia	VARCHAR(45)	Não	Não	Não
	Ranking	INT	Não	Não	Não
Faixa	IdFaixa(PK)	INT	Não	Não	Não
	Nome	VARCHAR(45)	Não	Não	Não
	Genero	VARCHAR(45)	Não	Não	Não
	Data	DATE	Não	Não	Não
	Duracao	TIME	Não	Não	Não
	Reproducoes	INT	Não	Não	Não
Playlist	IdPlaylist(PK)	INT	Não	Não	Não
	Nome	VARCHAR(45)	Não	Não	Não
	Descricao	VARCHAR(45)	Sim	Não	Não
	DuracaoTotal	TIME	Não	Não	Não
	NrFaixas	INT	Não	Não	Não
Album	IdAlbum(PK)	INT	Não	Não	Não
	Nome	VARCHAR(45)	Não	Não	Não
	Data	DATE	Não	Não	Não
	DuracaoTotal	TIME	Não	Não	Não

3.5. Detalhe ou generalização de entidades

Apresentamos agora uma descrição de todas as entidades, indicando os seus atributos, com uma breve caracterização destes, e a explicação da escolha da chave primária.

3.5.1. Chaves primárias e candidatas

3.5.1.1 Utilizador

Chaves candidatas: Username, Email, Telemovel
Chave primária: IdUtilizador

Para esta entidade foi escolhida como chave primária o IdUtilizador.

A chave candidata Username, apesar de ser única ao utilizador, não foi escolhida devido a possibilidade de ser bastante extensa.

Normalmente um email é único, mas podem ocorrer situações onde este é partilhado (família), logo também não pode ser usado como chave primária.

Por fim, o número de telemóvel, que apesar de identificar o cliente, pode sofrer alterações, não sendo assim usada a chave candidata Telemovel.

3.5.1.2 Artista

Chaves candidatas: Nome
Chave primária: IdArtista

Para esta entidade foi escolhida como chave primária o IdArtista.

A chave candidata Nome, não foi utilizada devido a poder ser bastante extenso ou conter caracteres especiais, o que a torna não desejável como chave primária

3.5.1.3 Playlist

Chaves candidatas: Nome
Chave primária: IdPlaylist

Para esta entidade a chave primária escolhida é IdPlaylist.

Tal como a chave candidata Nome da entidade Artista, esta chave candidata com o mesmo nome não pode ser utilizada pelo mesmo motivo, pode ser muito extensa.

3.5.1.4 Faixa

Chaves candidatas: Nome
Chave primária: IdFaixa

Para esta entidade a única chave que é capaz de representar a entidade Faixa é a chave primária IdFaixa.

A chave candidata Nome não pode ser utilizada pelos mesmos motivos apresentados na entidade Artista.

3.5.1.5 Album

Chaves candidatas: Nome
Chave primária: IdAlbum

Por fim, a entidade Album tem como chave primária IdAlbum.

A chave candidata Nome, como já referido acima, não pode ser usada devido a possibilidade de ser muito extensa.

3.5.2 Atributos

Apresentada-se agora uma breve descrição da entidade e dos atributos que a constituem.

3.5.2.1 Utilizador

- **IdUtilizador -** número que identifica um utilizador;
- **Username -** username da conta do utilizador;
- **Email -** email do utilizador;
- **Tipo -** tipo de utilizador (Free, Premium);
- **Data Nascimento -** data de nascimento do utilizador;
- **País -** país (atual) do utilizador;
- **Telemóvel -** número de telemóvel do utilizador.

O Utilizador, para aceder à sua conta, necessita de fazer login com o seu username ou email e a password. Dependendo da subscrição do utilizador, este vai ter um tipo associado que vai decidir se o utilizador tem acesso a determinadas funcionalidades. A data de nascimento do utilizador permite saber se o utilizador tem acesso a algumas secções, por exemplo utilizadores com menos de 18 anos não podem realizar pagamentos quando o controlo parental está ativo. A conta do utilizador, para além da data de nascimento, possui também o país atual do utilizador e, caso queira colocar, o número de telemóvel.

Por exemplo, o utilizador 45 representa a senhora Ana Oliveira que se inscreveu com o email "azeitona@gmail.com", o seu username é "phoebe" com a password "sushi", não possui uma subscrição paga (tipo = Free), nasceu no dia 1985/08/14 , reside atualmente em Inglaterra e o seu número de telefone é 913652120

3.5.2.2 Artista

- **IdArtista -** número que identifica um artista;
- **Nome -** o nome do artista/banda;
- **Biografia -** uma descrição da criação da banda ou vida do artista**;**
- **Ranking -** a posição atual do artista em relação a outros**.**

O Artista tem um nome que o identifica, uma pequena biografia sobre a sua vida e o caminho que seguiu na sua carreira, e a posição atual que detém no ranking.

Por exemplo (ex.), o artista 36 representa Nikita, a sua biografia contém a história da sua carreira (ex.: iniciou a sua carreira musical com um grupo de amigas onde aplicava uma técnica de "passa e toca" o que aumentou significativamente a sua capacidade musical) e está atualmente na posição 3 da tabela.

3.5.2.3 Playlist

- **IdPlaylist** - número que identifica uma playlist;
- **Nome** - nome da playlist;
- **Descrição** - pequena descrição da playlist;
- **Duração Total -** duração total da playlist;
- **NrFaixas -** número de faixas na playlist.

A Playlist tem um nome que a representa, uma pequena descrição, o autor que a criou, o número de faixas que possui, e a duração total da playlist.

Por exemplo, a playlist número 52 tem o nome de BDChill, possui uma descrição contendo a motivação do utilizador para a criar (ex.: esta playlist é boa para relaxar

enquanto o MySQL Workbench desiste de me perceber), tem uma duração total de 1:00:00 hora e 10 faixas.

3.5.2.4 Faixa

- **IdFaixa** - número que identifica uma faixa;
- **Nome** - nome da faixa;
- **Genero** - o género da faixa;
- **Reproduções** - o número de reproduções da faixa;
- **Data** - a data de lançamento da faixa;
- **Duração** - a duração da faixa.

A Faixa tem um nome dado pelo seu criador, o género, o número de reproduções (para efeitos de ranking), a data em que foi lançada e a sua duração.

Por exemplo, a faixa número 120 tem o nome de WillWeRockYou?, o seu género é Rock, conta com um elevado número de repetições (250), foi lançada em 7 de outubro de 1977 e tem duração de 2:02.

3.5.2.5 Album

- **IdAlbum -** número que identifica um álbum;
- **Nome -** o nome do álbum;
- **Data -** a data de lançamento do álbum;
- **Duração Total -** a duração total do álbum.

O Album tem um nome dado pelo artista, uma data que mostra quando foi lançado, o número de faixas que o constituem e a duração total.

Por exemplo, o álbum número 450 tem o nome de Back in White, foi lançado em 1980 e tem uma duração total 00:41:58.

3.6. Apresentação e explicitação do diagrama ER

Após a apresentação do diagrama ER e é feita uma breve explicação sobre esse mesmo diagrama.

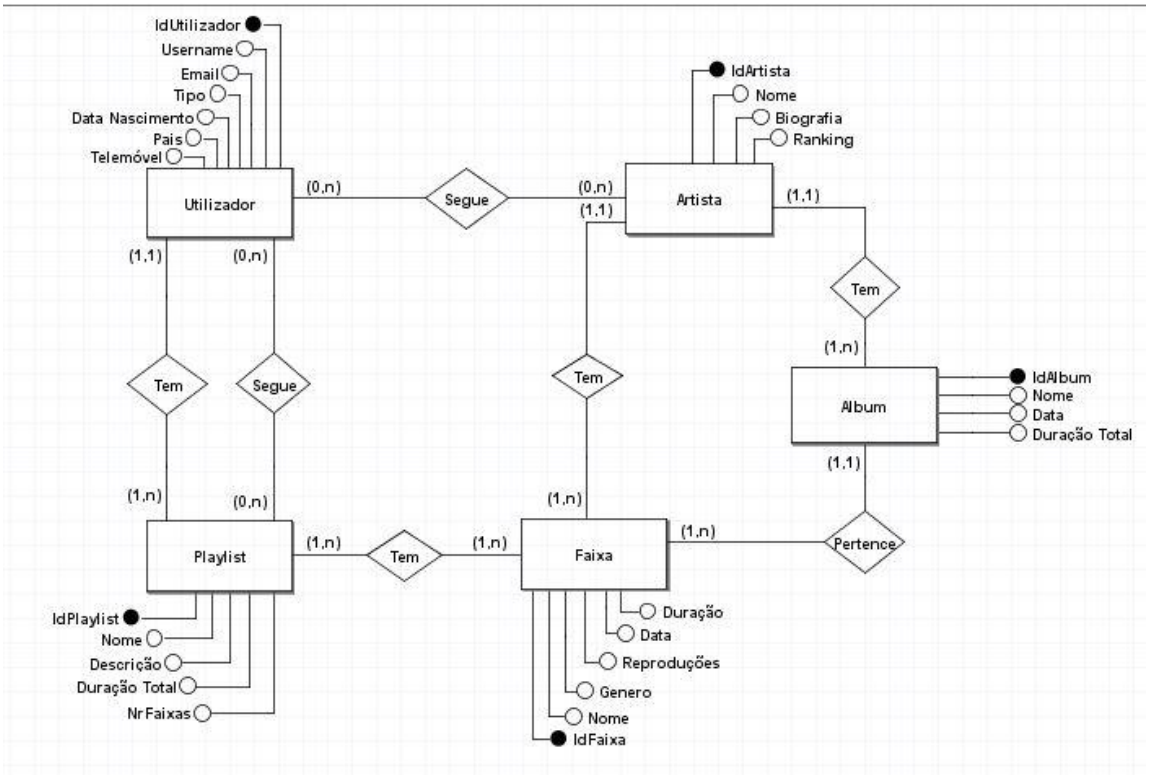


Ilustração 7: Diagrama ER

O Utilizador é identificado por um ID (IdUtilizador), por um username único, pelo seu email e pelo tipo de subscrição que possui. Outras características que o Utilizador possui na sua conta é a sua data de nascimento, o seu país de residência e o seu número de telefone. Um Utilizador tem três relacionamentos. O Utilizador tem a opção de seguir um ou vários Artistas, pode também criar uma ou mais Playlists ou pode optar antes por seguir Playlists criadas por outros Utilizadores.

No caso da Playlist, esta é identificada por um ID (IdPlaylist) e por um nome. Outros aspetos que a Playlist tem é uma descrição, a duração total e número de faixas. À exceção dos dois relacionamentos com Utilizador já explicados, a Playlist possui mais dois relacionamentos. Cada Playlist é constituída por uma ou várias Faixas.

A entidade Faixa é identificada também por um ID (IdFaixa) e por um nome. A Faixa tem várias características como o género a que pertence, o número de vezes que foi reproduzida, a data em que foi lançada e a sua duração. Além do relacionamento já explicado com Playlist, a Faixa possui mais dois relacionamentos. O primeiro indica que uma ou várias Faixas pertencem a um Album, e o outro diz-nos que um Artista tem (criou) uma ou várias Faixas.

Passando agora para a penúltima entidade que irá ser descrita, o Album é identificado também por um ID (IdAlbum) e pelo nome que o seu criador lhe dá. Outras características importantes do Album são a sua data de lançamento e a sua duração total. O Album tem dois relacionamentos, tendo um deles sido já explanado, passamos ao segundo relacionamento, que nos diz que um Artista pode ter um ou vários Albuns.

Por fim, temos o Artista que pode ser representado pelo seu ID (IdArtista) e pelo seu nome. O Artista tem também mais dois aspetos de caracterização: uma biografia sobre a sua vida/carreira e a posição atual no ranking.

3.7. Validação do modelo de dados com o utilizador

Nesta fase houve uma nova reunião com o nosso cliente, onde foi feita a validação dos requisitos e apresentado o modelo de dados, acompanhado de uma explicação detalhada sobre o mesmo. Não tendo havido problemas de aceitação e validação por parte do diretor da Arpeggio Musuc, prosseguimos para a próxima fase do projeto.

4. Modelação lógica

4.1. Construção e validação do modelo de dados lógico

Nesta secção apresenta-se o processo da transição do Modelo Conceptual para o Modelo Lógico.

Esta transição foi bastante simples, no entanto, houve um momento em que a tradução do Modelo Conceptual para o Lógico não foi direta. Isto deu-se no atributo País da entidade Utilizador. Com o objetivo de manter a integridade dos dados, o atributo País foi transformado em tabela, de maneira a evitar diferentes formas de identificar um país. As entidades País e Utilizador têm agora um relacionamento com cardinalidade 1:N entre elas.

4.1.1 Entidades Fortes

Uma Entidade Forte é caracterizada por possuir uma chave primária que a identifica e não apresentar dependência com outras chaves. Todos os atributos simples são considerados e incluídos no relacionamento.

Utilizador(IdUtilizador,Username,Data Nascimento, Telemovel, Email, Tipo)
Chave Primária: IdUtilizador

Playlist (IdPlaylist, Nome, Descricao, Autor, DuracaoTotal,NrFaixas)
Chave Primária: IdPlaylist

Faixa (IdFaixa, Nome, Genero, Data, Duracao, Reproducoes)
Chave Primária: IdFaixa

Album (IdAlbum, Nome, Data , DuracaoTotal)
Chave Primária: IdAlbum

Artista (IdArtista, Nome, Biografia, Ranking)
Chave Primária: IdArtista

4.1.2 Relacionamentos 1:N

Um relacionamento 1:N gera uma cópia da chave primária da Entidade com menor cardinalidade (entidade Pai) e coloca-a na Entidade de maior cardinalidade (entidade Filho). Esta cópia é designada por chave estrangeira e garante a integridade dos dados que referenciam a entidade Pai.

Utilizador(IdUtilizador,Username,Data Nascimento, Telemovel, Email, Tipo)
Chave Primária IdUtilizador
Chave Estrangeira Pais **referência** Pais(IdPais)

Faixa (IdFaixa, Nome, Genero, Data, Duracao, Reproducoes)
Chave Primária IdFaixa
Chave Estrangeira Album **referência** Album(IdAlbum)
Chave Estrangeira Artista **referência** Artista(IdArtista)

Album (IdAlbum, Nome, Data, DuracaoTotal)
Chave Primária IdAlbum
Chave Estrangeira Artista **referência** Artista(IdArtista)

Playlist (IdPlaylist, Nome, Descricao, DuracaoTotal,NrFaixasl)
Chave Primária IdPlaylist
Chave Estrangeira Utilizador **referência** Utilizador(IdUtilizador)

4.1.3 Relacionamentos N:M

Em cada relacionamento N:M é criada uma tabela com as chaves estrangeiras de ambas as entidades. Cada entidade liga-se a essa tabela com uma cardinalidade de 1:N (gerando assim as chaves estrangeiras)

Utilizador(IdUtilizador,Username,Data Nascimento, Telemovel, Email, Tipo)
Chave Primária IdUtilizador

Artista(IdArtista, Nome, Biografia, Ranking)
Chave Primária IdArtista

Utilizador_follows_Artista
Chave Primária IdUtilizador,IdArtista
Chave Estrangeira IdUtilizador **referência** Utilizador(IdUtilizador)
Chave Estrangeira IdArtista **referência** Artista(IdArtista)

Playlist(IdPlaylist, Nome, Descricao, DuracaoTotal,NrFaixas)
Chave Primária IdPlaylist

Faixa(IdFaixa, Nome, Genero, Data, Duracao,Reproducoes)
Chave Primária IdFaixa

Playlist_has_Faixa
Chave Primária IdPlaylist,IdFaixa
Chave Estrangeira IdPlaylist **referência** Playlist(IdPlaylist)
Chave Estrangeira IdFaixa **referência** Faixa(IdFaixa)

4.2. Desenho do modelo lógico

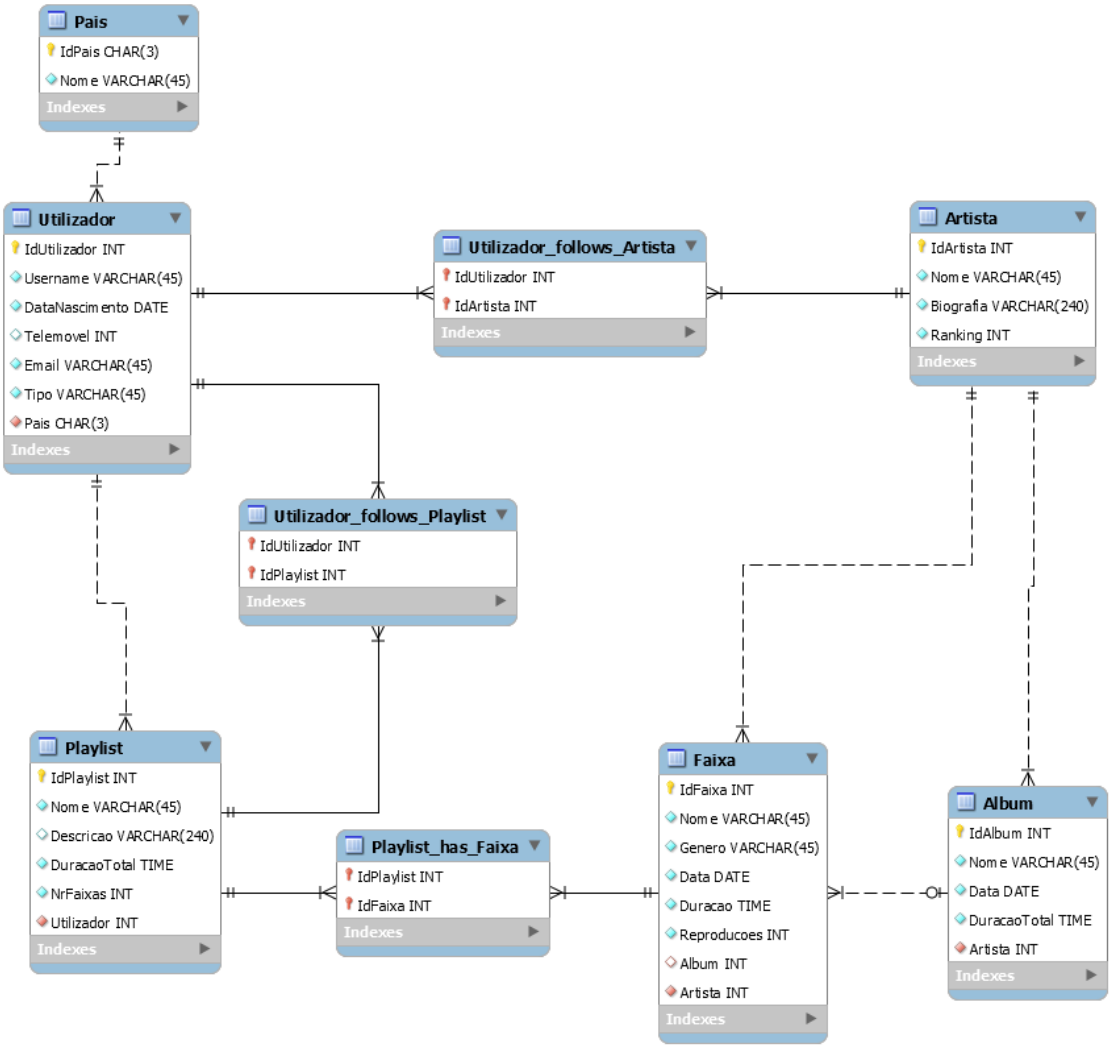


Ilustração 8: Representação do modelo lógico

4.3. Validação do modelo através da normalização

Ao passar do Modelo Conceptual para o Modelo Lógico, por vezes existem anomalias que podem tornar a base de dados inconsistente. Por isso, é necessário validar o Modelo Lógico através da normalização.

Para o projeto em questão utilizamos três formas normais.

4.3.1. Primeira forma normal

Uma tabela está na primeira forma normal, se todos os valores de todos os atributos forem atómicos, isto é, se não for possível decompô-los. Cada linha e cada coluna deve conter apenas um valor.

Analisamos as tabelas do modelo procurando atributos que seriam repetidos, identificando assim o caso do atributo País.

Criou-se uma tabela para o País do utilizador por diversas razões. Uma das razões é a possibilidade de saber quais os utilizadores de um determinado país, sendo mais fácil e rápido organizá-los. A segunda razão é a de evitar redundâncias no sistema. Um exemplo para esta situação seria a procura de utilizadores de Portugal. Um utilizador quando lhe é pedido para escrever em que país reside atualmente coloca Portugal, outro utilizador escreve apenas PT e apesar de ambos fazerem referência a Portugal o sistema iria considerar estes dois registos como sendo países diferentes. Ao criar uma tabela para os países evitam-se problemas deste tipo.

Relacionamento N:1 Utilizador-Pais:

Como vários utilizadores podem fazer parte do mesmo país, a cardinalidade do relacionamento Utilizador-Pais é de N:1

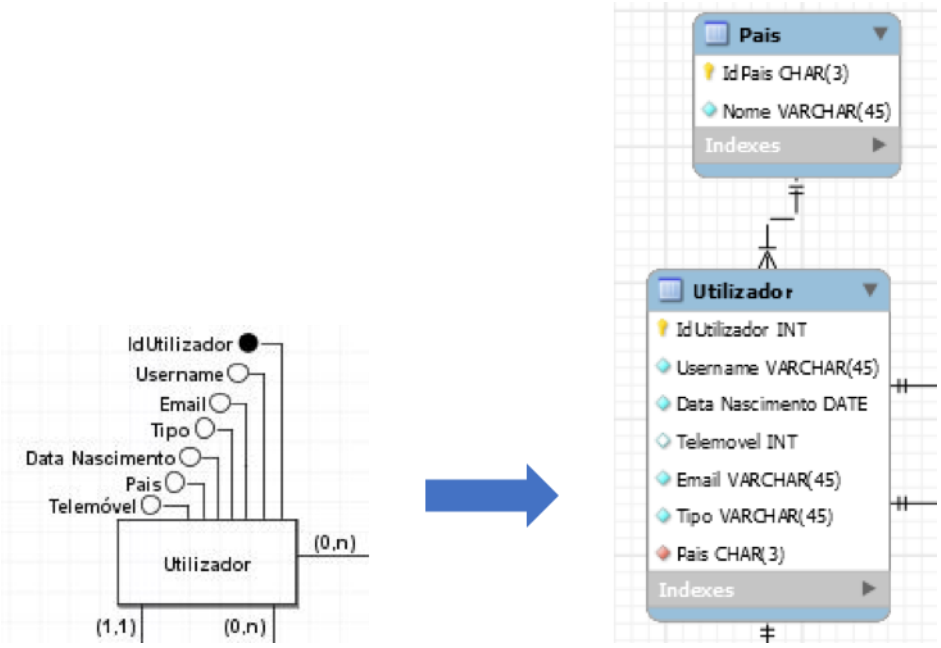


Ilustração 9: Representação do exemplo de cumprimento da primeira forma normal.

4.3.2. Segunda forma normal

Uma vez que o modelo verifica a primeira forma normal, passamos para a segunda forma normal. Esta indica que um atributo que não seja chave primária vai depender totalmente desta, ou seja, não possui uma dependência parcial**.** O nosso modelo já verifica a segunda forma normal pois não possui nenhuma dependência parcial.

Um exemplo desta dependência total na entidade Album, onde os atributos simples “Nome”, “Data” e “DuracaoTotal” dependem da chave primária “IdAlbum”

Ilustração 10: Representação do exemplo de cumprimento da segunda forma normal

4.3.3. Terceira forma normal

Por fim, como o nosso modelo já verifica as duas primeiras formas normais passaremos à restante. A terceira forma normal verifica se o modelo tem dependências transitivas. Todos os atributos da tabela devem ser independentes uns dos outros, sendo exclusivamente dependentes da chave primária. O nosso modelo verifica também a terceira forma normal.

4.4. Validação do modelo com interrogações do utilizador

É necessário realizar a validação com as interrogações do utilizador, de forma a verificar se o modelo corresponde às suas expectativas. Com base na aplicação que os utilizadores vão fazer sobre a base de dados chegamos a quatro interrogações importantíssimas que o nosso modelo deve conseguir responder:

- **Número de utilizadores, de um determinado país, que seguem um artista específico**

Para determinar o número de utilizadores de um determinado país que seguem um artista específico é necessário recolher dados das tabelas Pais, Utilizador e Artista. Através da relação das tabelas Utilizador e Pais é possível retirar os utilizadores de um determinado país. Com a relação entre as tabelas Utilizador e Pais é possível determinar quais os utilizadores que seguem um artista específico, determinando assim o número de utilizadores que seguem um artista distinto e que pertencem a um país específico.

- **Procura as faixas de uma playlist**

Tirando informação das tabelas Faixa e Playlist, é possível descobrir as faixas que constituem uma determinada playlist.

- **Procura quais são os álbuns de um determinado artista**

É possível determinar os nomes dos álbuns através da relação entre a tabela Album e a tabela Artista, isto pois, é necessário verificar quem é o artista que o utilizador procura e os álbuns que por ele foram lançados.

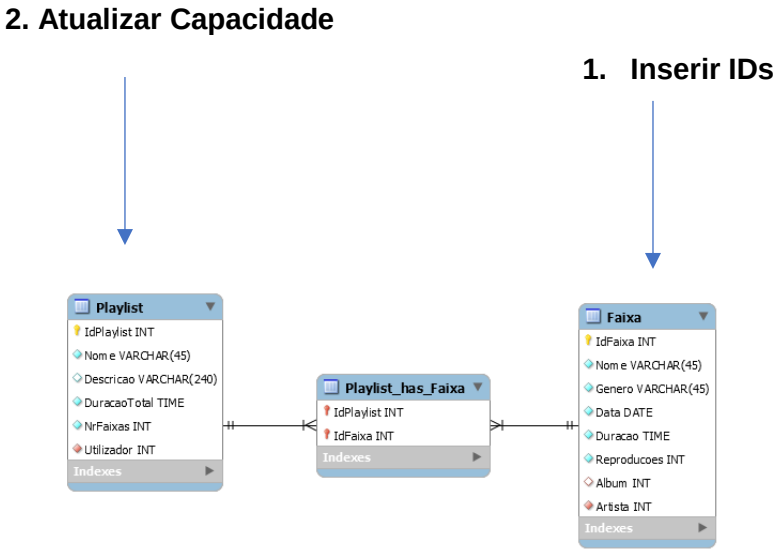
- **Procura as faixas, de um dado artista, que foram lançadas entre duas datas**

Para ser possível determinar que faixas foram lançadas entre duas datas por um dado artista é necessário usar a relação que as tabelas Artista e Faixa possuem, retirando a informação necessária de cada uma.

4.5. Validação do modelo com as transações estabelecidas

A validação das transações estabelecidas foi realizada através de mapas de transações. Assim, é possível ver as interações entre as diferentes tabelas de forma a obter o resultado pretendido. Destacamos duas das mais importantes:

- **Inserção de uma faixa numa playlist**



Quando é adicionado uma faixa à playlist, é necessário realizar duas ações. A primeira ação que se vai realizar é inserir na tabela Playlist_has_Faixa o ID da faixa que se pretende inserir na playlist e o ID da playlist onde se quer inserir, por fim atualizamos o atributo NrFaixas na tabela Playlist.

- **Remoção de uma Faixa de uma Playlist**

Raciocínio é o mesmo da inserção.

4.6. Revisão do modelo lógico com o utilizador

De forma a verificar se a estrutura criada no modelo lógico satisfaz os requisitos desejados, foi feita uma nova reunião com o cliente.

Após rever as especificações, o modelo foi aprovado tendo cumprido todos os requisitos que haviam sido impostos.

5. Implementação física

5.1. Seleção do sistema de gestão de base de dados

De forma a escolher o Sistema de Gestão de Bases de Dados que melhor se adequa ao problema, levantaram-se diversos critérios que este tinha de cumprir:

- Um SGBD deve permitir a um utilizador autorizado definir uma estrutura de dados que mais se adapta ao problema atual. Deve também poder definir permissões de acesso, incluindo criação, leitura, modificação e eliminação.

- Permitir a consulta eficiente dos dados, compete ao SGBD escolher a forma mais eficiente de executar a consulta e de devolver o resultado ao utilizador.

- Deve permitir que a base de dados seja recuperada, na eventualidade de uma falha crítica, com a menor perda de informação possível.

- O SGBD deve permitir operações simultâneas sobre os dados por parte de vários utilizadores.

O SGBD escolhido foi o *MySQL Workbench*, não só por ser o sistema usado nas aulas práticas da Unidade Curricular, mas também devido às suas características de simplicidade e fácil utilização.

5.2. Tradução do esquema lógico para o SGDB escolhido em SQL

O processo de modelação do esquema físico envolve a tradução dos relacionamentos previamente definidos no Modelo Lógico, para um código que possa ser implementado no SGBD. Este procedimento é constituído por duas fases, sendo a primeira responsável pelo tratamento da informação de todos os relacionamentos e todos os atributos do Modelo Lógico em conjunto com a informação do levantamento e da análise de requisitos do Modelo Conceptual. A segunda fase do processo vai usar toda a informação recolhida para produzir os relacionamentos base e restrições gerais.

Uma vez que a BD não possui restrições gerais nem atributos derivados, esta vai ser descrita por relacionamentos base. Os relacionamentos base são constituídos pelos seguintes parâmetros: nome do relacionamento, chave primária, que por vezes pode possuir uma chave estrangeira, conjunto de restrições de integridade referencial para cada chave estrangeira identificada. Cada atributo detém os seguintes parâmetros: domínio, valor por defeito, se o atributo é derivado ou não (caso seja apresentamos como é calculado) e por fim verifica-se se o atributo suporta valores nulos.

- **Relação Utilizador**

Domínio IdUtilizador: Inteiro
Domínio Username: String de tamanho variável, tamanho máximo 45
Domínio Data Nascimento: Data, formato: AAAA-MM-DD
Domínio Telemovel: Inteiro
Domínio Email: String de tamanho variável, tamanho máximo 45
Domínio Tipo: String de tamanho variável, tamanho máximo 45
Domínio País: String de tamanho fixo, tamanho 3

Utilizador(
IdUtilizador: NOT NULL,
Username: NOT NULL,
DataNascimento: NOT NULL,
Telemovel: NULL,
Email: NOT NULL,
Tipo: NOT NULL,
Pais: NOT NULL,
PRIMARY KEY(IdUtilizador),
FOREIGN KEY(Pais) **REFERENCES** País(IdPais)**)**;

```
-- -----------------------------------------------------
-- Table `Arpeggio`.`Utilizador`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Arpeggio`.`Utilizador` (
  `IdUtilizador` INT NOT NULL,
  `Username` VARCHAR(45) NOT NULL,
  `Data Nascimento` DATE NOT NULL,
  `Telemovel` INT NULL,
  `Email` VARCHAR(45) NOT NULL,
  `Tipo` VARCHAR(45) NOT NULL,
  `Pais` CHAR(3) NOT NULL,
  PRIMARY KEY (`IdUtilizador`),
  INDEX `fk_Utilizador_Pais1_idx` (`Pais` ASC),
  CONSTRAINT `fk_Utilizador_Pais1`
    FOREIGN KEY (`Pais`)
    REFERENCES `Arpeggio`.`Pais` (`IdPais`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Ilustração 11: Criação da tabela Utilizador

- **Relação País**

Dominio IdPais: String de tamanho fixo, tamanho 3,
Dominio Nome: String de tamanho variável, tamanho máximo 45,

Pais(
IdPais: NOT NULL,
Nome: NOT NULL,
PRIMARY KEY(IdPais)**)**;

```
-- -----------------------------------------------------
-- Table `Arpeggio`.`Pais`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Arpeggio`.`Pais` (
  `IdPais` CHAR(3) NOT NULL,
  `Nome` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`IdPais`))
ENGINE = InnoDB;
```

Ilustração 12: Criação da tabela País

- **Relação Artista**

Dominio IdArtista: Inteiro,
Dominio Nome: String de tamanho variável, tamanho máximo 45,
Dominio Biografia: String de tamanho variável, tamanho máximo 240,
Dominio Ranking: Inteiro,

Artista(
IdArtista: NOT NULL,
Nome: NOT NULL,
Biografia: NOT NULL,
Ranking: NOT NULL,
PRIMARY KEY(IdArtista)**)**;

```
-- -----------------------------------------------------
-- Table `Arpeggio`.`Artista`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Arpeggio`.`Artista` (
  `IdArtista` INT NOT NULL,
  `Nome` VARCHAR(45) NOT NULL,
  `Biografia` VARCHAR(240) NOT NULL,
  `Ranking` INT NOT NULL,
  PRIMARY KEY (`IdArtista`))
ENGINE = InnoDB;
```

Ilustração 13: Criação da tabela Artista

- **Relação Playlist**

Dominio IdPlaylist: Inteiro,
Dominio Nome: String de tamanho variável, tamanho máximo 45,
Dominio Descrição: String de tamanho variável, tamanho máximo 240,
Dominio DuracaoTotal: Hora, formato: HH:MM:SS,
Domínio NrFaixas: Inteiro,
Dominio Utilizador: Inteiro,

Playlist(
IdPlaylist: NOT NULL,
Nome: NOT NULL,
Descricao: NULL,
DuracaoTotal: NOT NULL,
NrFaixas: NOT NULL,
Utilizador: NOT NULL,
PRIMARY KEY(IdPlaylist),
FOREIGN KEY(Utilizador) **REFERENCES** Utilizador(IdUtilizador),

```
47
48    -- -----------------------------------------------------
49    -- Table `Arpeggio`.`Playlist`
50    -- -----------------------------------------------------
51  CREATE TABLE IF NOT EXISTS `Arpeggio`.`Playlist` (
52    `IdPlaylist` INT NOT NULL,
53    `Nome` VARCHAR(45) NOT NULL,
54    `Descricao` VARCHAR(240) NULL,
55    `DuracaoTotal` TIME NOT NULL,
56    `NrFaixas` INT NOT NULL,
57    `Utilizador` INT NOT NULL,
58    PRIMARY KEY (`IdPlaylist`),
59    INDEX `fk_Playlist_Utilizador1_idx` (`Utilizador` ASC),
60    CONSTRAINT `fk_Playlist_Utilizador1`
61      FOREIGN KEY (`Utilizador`)
62      REFERENCES `Arpeggio`.`Utilizador` (`IdUtilizador`)
63      ON DELETE NO ACTION
64      ON UPDATE NO ACTION)
65  ENGINE = InnoDB;
66
```

Ilustração 14: Criação da tabela Playist

- **Relação Faixa**

Dominio IdFaixa: Inteiro,
Dominio Nome: String de tamanho variável, tamanho máximo 45,
Dominio Genero: String de tamanho variável, tamanho máximo 45,
Dominio Data: Data, formato:AAAA-MM-DD,
Dominio Duracao: Hora, formato: HH:MM:SS,
Dominio Reproducoes: Inteiro,
Dominio Album: Inteiro,
Dominio Artista: Inteiro,

Faixa(
IdFaixa: NOT NULL,
Nome: NOT NULL,
Genero: NOT NULL,
Data: NOT NULL,
Duracao: NOT NULL,
Reproducoes: NOT NULL,
Album: NULL,
Artista: NOT NULL,
PRIMARY KEY(IdFaixa),
FOREIGN KEY(Album) **REFERENCES** Album(IdAlbum),
FOREIGN KEY(Artista) **REFERENCES** Artista(IdArtista)**)**;

```
-- -----------------------------------------------------
-- Table `Arpeggio`.`Faixa`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Arpeggio`.`Faixa` (
  `IdFaixa` INT NOT NULL,
  `Nome` VARCHAR(45) NOT NULL,
  `Genero` VARCHAR(45) NOT NULL,
  `Data` DATE NOT NULL,
  `Duracao` TIME NOT NULL,
  `Reproducoes` INT NOT NULL,
  `Album` INT NULL,
  `Artista` INT NOT NULL,
  PRIMARY KEY (`IdFaixa`),
  INDEX `fk_Faixa_Album1_idx` (`Album` ASC),
  INDEX `fk_Faixa_Artista1_idx` (`Artista` ASC),
  CONSTRAINT `fk_Faixa_Album1`
    FOREIGN KEY (`Album`)
    REFERENCES `Arpeggio`.`Album` (`IdAlbum`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Faixa_Artista1`
    FOREIGN KEY (`Artista`)
    REFERENCES `Arpeggio`.`Artista` (`IdArtista`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Ilustração 15: Criação da tabela Faixa

- **Relação Album**

Dominio IdAlbum: Inteiro,
Dominio Nome: String de tamanho variável, tamanho máximo 45,
Dominio Data: Data, formato: AAAA-MM-DD,
Dominio DuracaoTotal: Hora, formato: HH:MM:SS,
Dominio Artista: Inteiro,

Album(
IdAlbum: NOT NULL,
Nome: NOT NULL,
Data: NOT NULL,
DuracaoTotal: NOT NULL,
Artista: NOT NULL,
PRIMARY KEY(IdAlbum)
FOREIGN KEY(Artista) **REFERENCES** Artista(IdArtista)**)**;

```
79  -- -----------------------------------------------------
80  -- Table `Arpeggio`.`Album`
81  -- -----------------------------------------------------
82  CREATE TABLE IF NOT EXISTS `Arpeggio`.`Album` (
83    `IdAlbum` INT NOT NULL,
84    `Nome` VARCHAR(45) NOT NULL,
85    `Data` DATE NOT NULL,
86    `DuracaoTotal` TIME NOT NULL,
87    `Artista` INT NOT NULL,
88    PRIMARY KEY (`IdAlbum`),
89    INDEX `fk_Album_Artista1_idx` (`Artista` ASC),
90    CONSTRAINT `fk_Album_Artista1`
91      FOREIGN KEY (`Artista`)
92      REFERENCES `Arpeggio`.`Artista` (`IdArtista`)
93      ON DELETE NO ACTION
94      ON UPDATE NO ACTION)
95  ENGINE = InnoDB;
96
```

Ilustração 16: Criação da tabela Album

5.3. Tradução das interrogações do utilizador para SQL (alguns exemplos)

As interrogações usadas anteriormente para a validação do modelo lógico, irão ser agora traduzidas para SQL:

- **Número de utilizadores, de um determinado país, que seguem um artista específico:**

```
USE arpeggio;

-- ---QUERY 1---
-- Número de utilizadores, de um determinado país, que seguem um artista específico
DELIMITER $$
CREATE PROCEDURE UtilizadoresDePaisSegArtista(IN nomeP VARCHAR(45), IN nomeA VARCHAR(45) )
BEGIN
SELECT Count(Utilizador.IdUtilizador) AS Numero_Utilizadores,nomeA AS Nome_Artista,nomeP AS Nome_Pais
    FROM Pais
        INNER JOIN Utilizador
        ON IdPais = Pais
            INNER JOIN utilizador_follows_artista
            ON
            (SELECT IdArtista FROM Artista WHERE Artista.Nome = nomeA) = utilizador_follows_artista.IdArtista
    WHERE nomeP = Pais.Nome;
END $$

DELIMITER ;

SET @nomeP = "Portugal";
SET @nomeA = "HotPlay";

CALL UtilizadoresDePaisSegArtista(@nomeP,@nomeA);
DROP Procedure UtilizadoresDePaisSegArtista;
```

Ilustração 17: Criação da interrogação acima referida(1)

- **Procurar o número de faixas de uma playlist:**

```
-- ---QUERY 2---
-- Procura o número de faixas de uma playlist
DELIMITER $$

CREATE PROCEDURE NrFaixasPlaylist ( IN ID_P INT)
BEGIN
SELECT Count(Faixa.IdFaixa) AS NúmeroFaixas, Playlist.Nome AS NomePlaylist
    FROM Playlist_has_Faixa
    INNER JOIN Faixa
    ON Faixa.IdFaixa = Playlist_has_Faixa.IdFaixa
    INNER JOIN playlist
    ON playlist.IdPlaylist=ID_P
    WHERE playlist_has_faixa.IdPlaylist=ID_P;
END $$

DELIMITER;

SET @id = 2;
CALL NrFaixasPlaylist (@id);
DROP PROCEDURE NrFaixasPlaylist;
```

Ilustração 18: Criação da interrogação acima referida(2)

- **Procurar quais são os álbuns de um determinado artista:**

```
-- ---QUERY 3---
-- Procura quais são os álbuns de um determinado artista

DELIMITER $$
CREATE procedure AlbunsArtista ( IN ID_Art INT)
BEGIN
SELECT IdAlbum AS ID_Album, Album.Nome AS Album, Artista.Nome AS Artista
    FROM Album
    INNER JOIN artista
    ON IdArtista = Artista
    WHERE Artista = ID_Art;
END $$

DELIMITER ;

SET @id = 2;

CALL AlbunsArtista (@id);
DROP PROCEDURE AlbunsArtista;
```

Ilustração 19: Criação da interrogação acima referida(3)

- **Procura as faixas, de um dado artista, que foram lançadas entre duas datas:**

```
-- ---QUERY 4---
-- Procura as faixas, de um dado artista, que foram lançadas entre duas datas
DELIMITER $$
CREATE PROCEDURE FaixasEntreDatas ( IN ID_Art INT, IN data_ini DATE, IN data_fim DATE )
BEGIN
SELECT IdFaixa, Faixa.Nome FROM Faixa
    INNER JOIN Artista
    ON faixa.artista = artista.IdArtista
    WHERE faixa.artista = ID_Art AND faixa.`Data` BETWEEN data_ini AND data_fim;
END $$

DELIMITER ;

SELECT * FROM ARTista;
SELECT * FROM Faixa;

SET @id = 1;
SET @data_ini = '2002-10-22';
SET @data_fim = '2012-03-16';

CALL FaixasEntreDatas( @id, @data_ini, @data_fim );
DROP PROCEDURE FaixasEntreDatas ;
```

Ilustração 20: Criação da interrogação acima referida(4)

5.4. Tradução das transações estabelecidas para SQL (alguns exemplos)

```
USE arpeggio;

DELIMITER $$

CREATE PROCEDURE insereFaixa (IN play_id INT,
                              IN faixa_id INT)
BEGIN
    DECLARE Erro BOOL DEFAULT 0;
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET Erro = 1;
    START TRANSACTION;


INSERT INTO playlist_has_faixa -- inserir faixa em playlist
    (IdPlaylist,IdFaixa)
    VALUES
    (play_id, faixa_id);

IF Erro
    THEN ROLLBACK; -- caso ocorra um erro, anula todas as instruções MySQL que executou
ELSE COMMIT;
END IF;


END $$

DELIMITER ;

SET @play_id = 9;
SET @faixa_id = 2;
CALL insereFaixa(@play_id,@faixa_id);
SELECT * FROM playlist_has_faixa ORDER BY IdPlaylist;
```

Ilustração 21: Exemplo de criação de transação

Esta transação foi criada com o objetivo de adicionar uma faixa a uma playlist. Caso a transação seja realizada com sucesso é feito um COMMIT dessa mesma

transação, se ocorrer algum erro durante a transação fazemos o ROLLBACK da transação, voltando ao estado anterior da Base de Dados antes de ser realizada esta transação.

Quando realizamos a transação percebemos que também teria de haver um trigger que fosse despoletado quando ocorresse a inserção. O trigger teria o intuito de fazer a atualização do número de faixas da playlist onde fazemos a inserção.

```
USE arpeggio;

DELIMITER $$

CREATE TRIGGER NrFaixas_update -- atualiza o número de faixas da playlist onde foi inserida a faixa
AFTER INSERT ON Playlist_Has_Faixa -- é ativado após uma inserção na tabela que possui a associacao das playlists com faixas
FOR EACH ROW -- executado uma vez para cada linha afetada pelo evento
BEGIN

    UPDATE Playlist AS P
        INNER JOIN Playlist_Has_Faixa AS PF
        ON P.IdPlaylist = NEW.IdPlaylist
    SET P.NrFaixas = P.NrFaixas+1;

END $$


DELIMITER ;
DROP TRIGGER NrFaixas_update;
```

Ilustração 22:

5.5. Escolha, definição e caracterização de índices em SQL (alguns exemplos)

Quando a dimensão de uma base de dados é grande, pode surgir a necessidade de otimizar as consultas das tabelas que possuem um elevado número de registos.

Introduzem-se assim, índices, que têm vários benefícios para base de dados com grande tamanho:

- Capazes de aumentar a velocidade de procura de resultados
- Capazes de ajudar a ordenar os registos mais facilmente e rapidamente

Em relação à nossa base de dados nesta fase inicial, não haveria um ganho, em termos de velocidade, que justificasse a adição de índices. Apesar de isto, é sempre importante pensar no crescimento que a base de dados vai ter e se futuramente não deveriam ser usados índices.

Analisando a nossa base de dados, conseguimos ver que existe pelo menos uma situação onde o uso de índices seria extremamente benéfico para a velocidade das consultas do nosso sistema, o agrupamento de faixas pelo género musical.

```
-- Agrupa as faixas pelo genero apresentando o identificador da faixa e o nome
SELECT Genero, IdFaixa, Nome
    FROM Faixa
GROUP BY Genero;
```

Ilustração 23: Exemplo de query para Faixas

Apesar da query ser bastante simples, num elevado número de registos vai ser extremamente demorada. Como tal decidimos criar índices para esta situação, para uma possível futura implementação.

```
-- Indice para ajudar no agrupamento
CREATE INDEX by_genero ON Faixa (`Genero`);
```

Ilustração 24: Exemplo de índice para género

5.6. Estimativa do espaço em disco necessário e taxa de crescimento anual

Um dos fatores mais importantes numa base de dados que é necessário ter em conta é o espaço que está ou que será usado para armazenar toda a informação da base de dados. Cada registo de uma determinada tabela ocupa espaço na memória que depende do tipo de dados que o constituem.

Necessitamos então de calcular uma estimativa do tamanho que irá ser utilizado pela base de dados. Para isso começamos por determinar quanto espaço ocupa cada atributo calculando depois uma previsão futura do seu crescimento.

De seguida apresentamos uma tabela com o tamanho que cada tipo de dados dentro da nossa base de dados ocupa:

Tabela 4: Representação do tamanho de cada tipo de dados ocupa

Data Type	***Tamanho (bytes)***
INTEGER	4
CHAR(N)	N x w *
VARCHAR(N)	N+1 **
DATE	3
TIME	3

* O cálculo do tamanho do caracter depende do N(número de bytes da string) e do w. w é o tamanho máximo de bytes usados no character set. O character set usado é utf8, como só é necessário representar os primeiros 128 bits (correspondem um para um aos valores do código de ASCII) só necessitamos de 7 bits. Devido ao mínimo usado em utf8 ser 8 bits vamos ter: w=1 byte.

******Como os VARCHAR presentes na SGBD têm menos de 255 bytes, o tamanho é sempre N (número de bytes da string) +1, caso contrário seria N+2.

A seguir são apresentadas as diversas tabelas com o tamanho de um registo, em cada tabela da Arpeggio:

- **Tabela Utilizador**

Tabela 5:Representação do tamanho que cada tipo de dados ocupa na tabela Utilizador

Atributo	***Data Type***	***Tamanho(bytes)***
IdUtilizador	INT	4 bytes
Username	VARCHAR(45)	46 bytes
DataNascimento	DATE	3 bytes
Telemovel	INT	4 bytes
Email	VARCHAR(45)	46 bytes
Tipo	VARCHAR(45)	46 bytes
Pais	CHAR(3)	3 bytes
TOTAL		Tamanho dos dados para um Utilizador = 152 bytes

- **Tabela País**

Tabela 6: Representação do tamanho que cada tipo de dados ocupa na tabela País

Atributo	Data type	Tamanho(bytes)
IdPais	***CHAR(3)***	***3 bytes***
Nome	**VARCHAR(45)**	**46 bytes**
TOTAL		***Tamanho dos dados para um Pais = 49 bytes***

- **Tabela Artista**

Tabela 7: Representação do tamanho que cada tipo de dados ocupa na tabela Artista

Atributo	***Data Type***	***Tamanho(bytes)***
IdArtista	INT	4 bytes
Nome	VARCHAR(45)	46 bytes
Biografia	VARCHAR(240)	240 bytes
Ranking	INT	4 bytes
TOTAL		Tamanho dos dados para um Artista = 294 bytes

- **Tabela Playlist**

Tabela 8: Representação do tamanho que cada tipo de dados ocupa na tabela Playlist

Atributo	***Data Type***	***Tamanho(bytes)***
IdPlaylist	INT	4 bytes
Nome	VARCHAR(45)	46 bytes
Descricao	VARCHAR(240)	241 bytes
DuracaoTotal	TIME	3 bytes
NrFaixas	INT	4 bytes
Utilizador	INT	4 bytes
TOTAL		Tamanho dos dados para uma Playlist = 302 bytes

- **Tabela Faixa**

Tabela 9: Representação do tamanho que cada tipo de dados ocupa na tabela Faixa

Atributo	***Data Type***	***Tamanho(bytes)***
IdFaixa	INT	4 bytes
Nome	VARCHAR(45)	46 bytes
Genero	VARCHAR(45)	46 bytes
Data	DATE	3 bytes
Duracao	TIME	3 bytes
Reproducoes	INT	4 bytes
Album	INT	4 bytes
Artista	INT	4 bytes
TOTAL		Tamanho dos dados para uma Faixa= 114 bytes

- **Tabela Album**

Tabela 10: Representação do tamanho que cada tipo de dados ocupa na tabela Album

Atributo	***Data Type***	***Tamanho(bytes)***
IdAlbum	INT	4 bytes
Nome	VARCHAR(45)	46 bytes
Data	DATE	3 bytes
DuracaoTotal	TIME	3 bytes
Artista	INT	4 bytes
TOTAL		Tamanho dos dados para um Album = 60 bytes

- **Tabela Utilizador_follows_Artista**

Tabela 11: Representação do tamanho que cada tipo de dados ocupa na tabela Utilizador_follows_Artista

Atributo	***Data Type***	***Tamanho(bytes)***
IdUtilizador	INT	4 bytes
IdArtista	INT	4 bytes
TOTAL		Tamanho dos dados para o relacionamento Utilizador_follows_Artista = 8 bytes

- **Utilizador_follows_Playlist**

Tabela 12: Representação do tamanho que cada tipo de dados ocupa na tabela Utilizador_follows_Playlist

Atributo	***Data Type***	***Tamanho(bytes)***
IdUtilizador	INT	4 bytes
IdPlaylist	INT	4 bytes
TOTAL		Tamanho dos dados para o relacionamento Utilizador_follows_Playlist= 8 bytes

- **Playlist_has_Faixa**

Tabela 13: Representação do tamanho que cada tipo de dados ocupa na tabela Playlist_has_Faixas

Atributo	***Data Type***	***Tamanho(bytes)***
IdPlaylist	INT	4 bytes
IdFaixa	INT	4 bytes
TOTAL		Tamanho dos dados para o relacionamento Playlist_has_Faixa = 8 bytes

Resumindo:

Tabela 14: Resumo da representação do tamanho que cada tabela ocupa

Tabela	***Tamanho(bytes)***
Utilizador	152 bytes
Pais	49 bytes
Artista	294 bytes
Playlist	302 bytes
Faixa	114 bytes
Album	60 bytes
Utilizador_follows_Artista	8 byte
Utilizador_follows_Playlist	8 byte
Playlist_has_Faixa	8 bytes

Considerando que a base de dados irá começar com uma pequena quantidade de informação, podemos estimar um tamanho inicial para a base de dados.

Assumindo que no inicio a base de dados continha 10 clientes de 3 países diferentes, 5 artistas com 1 album cada um, 10 faixas e 10 playlists, temos então o seguinte tamanho inicial:

Tabela 15: Cálculo do tamanho aproximado da base de dados (inicial)

Tabela	***Tamanho(bytes)***
Utilizador	152 x 10 = 1520 bytes
Pais	3 x 49 = 147 bytes
Artista	294 x 5 = 1470 bytes
Playlist	302 x 10 = 3020 bytes
Faixa	114 x 10 = 1140 bytes
Album	60 x 5 = 300 bytes
Utilizador_follows_Artista	8 x 0 = 0 bytes *
Utilizador_follows_Playlist	8 x 0 = 0 bytes *
Playlist_has_Faixa	8 x 10 = 80 bytes
Total	7667 bytes

*No estado inicial nenhum utilizador segue um artista nem uma playlist.

Com esta previsão e sabendo o tamanho que cada registo ocupa na BD, é possível deduzir o tamanho inicial da base dados, que será, aproximadamente, 7667 bytes (7.4873 kbytes)

Temos agora de estimar o crescimento anual que a empresa vai ter. O crescimento da empresa vai implicar o crescimento do número de utilizadores. Cada registo de um utilizador ocupa (como visto em cima) 152 bytes.

Tabela 16: Representação do aumento de utilizadores

Ano	***Nº Utilizadores***	***Espaço Ocupado(bytes)***
2018	70	10640 bytes
2019	110	16720 bytes
2020	150	22800 bytes
2021	170	25840 bytes
2022	200	30400 bytes

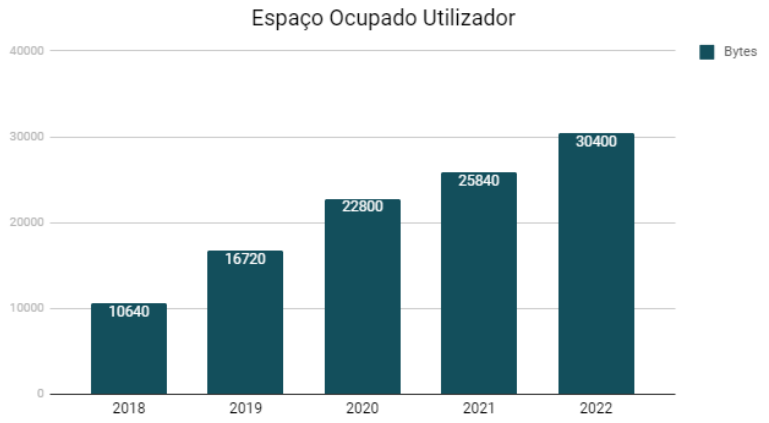


Ilustração 25: Representação do aumento de utilizadores

Consideremos agora que o serviço, como consequência da sua expansão, é utilizado em diversos países. Cada registo país vai ocupar 3 bytes.

Tabela 17: Representação do aumento de utilizadores em diferentes países

Ano	***Nº Paises***	***Espaço Ocupado(bytes)***
2018	15	45 bytes
2019	25	75 bytes
2020	50	150 bytes
2021	60	180 bytes
2022	75	225 bytes

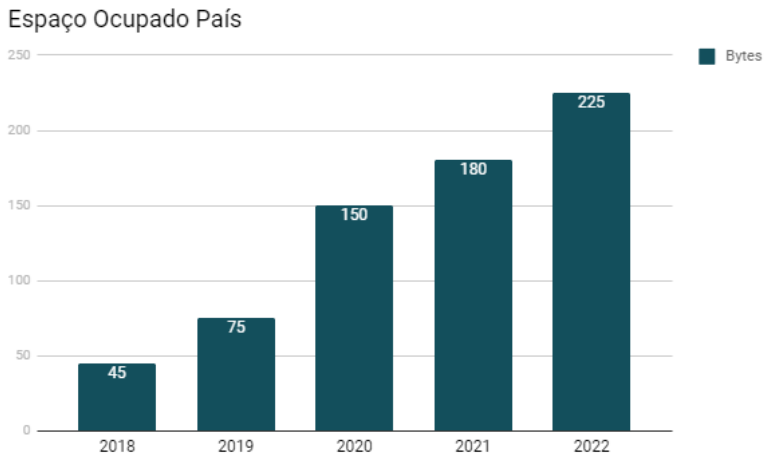


Ilustração 26: Representação do aumento de utilizadores em diferentes países

Com o aumento de utilizadores no serviço, vários artistas começaram também a fazer parte dele. Cada registo artista ocupa 294 bytes.

Tabela 18: Representação do aumento de artistas a utilizar a BD

Ano	***Nº Artistas***	***Espaço Ocupado(bytes)***
2018	50	14700 bytes
2019	80	23520 bytes
2020	100	29400 bytes
2021	120	35280 bytes
2022	150	44100 bytes

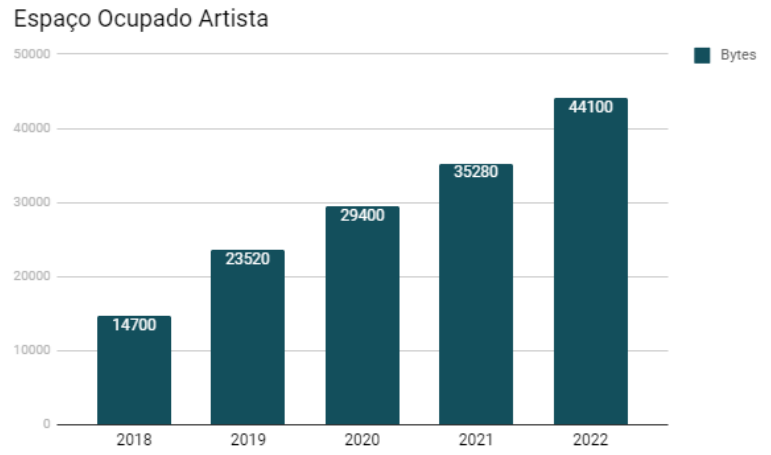


Ilustração 27: Representação do aumento de artistas a utilizar a BD

O número de playlists vai também aumentar com o número de utilizadores, como se pode ver no gráfico seguinte. Cada registo de uma playlist ocupa 302 bytes

Tabela 19: Representação do aumento de Playlistes existentes

Ano	*Nº Playlist*	*Espaço Ocupado(bytes)*
2018	80	24160 bytes
2019	125	37750 bytes
2020	175	52850 bytes
2021	200	60400 bytes
2022	250	75500 bytes

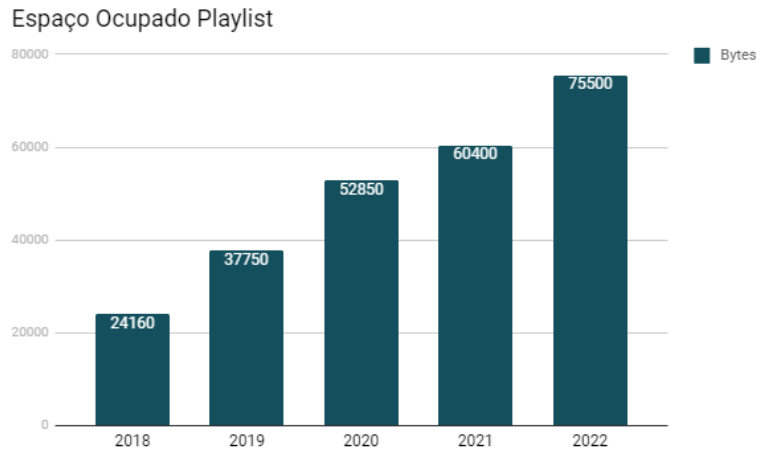


Ilustração 28: Representação do aumento de Playlistes existentes

De seguida mostramos o crescimento do número de faixas que o serviço vai ter. Cada registo Faixa ocupa 114 bytes.

Tabela 20: Representação do aumento do número de faixas

Ano	*Nº Faixas*	*Espaço Ocupado (bytes)*
2018	100	11400 bytes
2019	200	22800 bytes
2020	250	28500 bytes
2021	300	34200 bytes
2022	400	45600 bytes

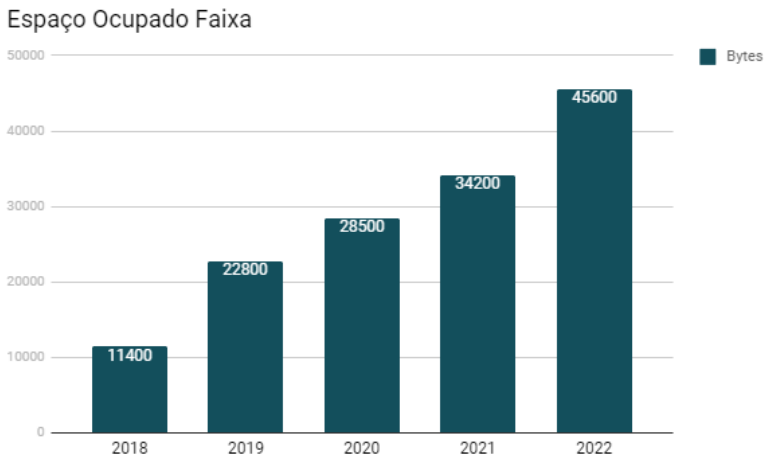


Ilustração 29: Representação do aumento do número de faixas

Era de prever que com o aumento de artistas o número de albuns iriam crescer também. Cada registo album ocupa 60 bytes.

Tabela 21: Representação do aumento do número de albuns

Ano	*Nº Album*	*Espaço Ocupado (bytes)*
2018	50	3000 bytes
2019	100	6000 bytes
2020	120	7200 bytes
2021	125	7500 bytes
2022	150	9000 bytes

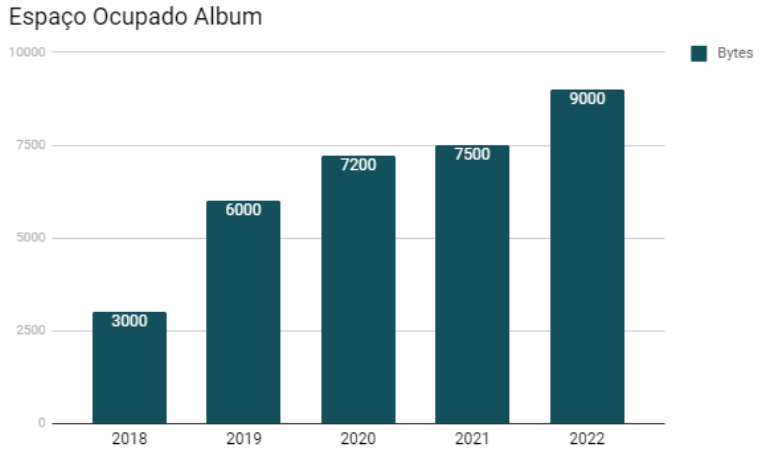


Ilustração 30: Representação do aumento do número de albuns

5.7. Definição e caracterização das vistas de utilização em SQL (alguns exemplos)

Um dos aspetos mais importantes no desenvolvimento de um SGBD são as vistas dos utilizadores, ou seja, os acessos e as ações que cada um pode executar.

Normalmente uma vista é considerada como uma tabela que vem da consulta de uma outra tabela. As vistas permitem simplificar a escrita de consultas frequentes, reduzir o seu número e esconder certos setores que não devem ser acedidos pelo utilizador.

Apresentamos de seguida algumas vistas do nosso modelo físico:

(1) View do Utilizador. Mostra os paises de todos os utilizadores ordenados pelo código ISO

```sql
USE arpeggio;

-- View dos Utilizadores

CREATE VIEW vwUtilizador AS
SELECT Pais,  Nome,IdUtilizador, Username, Email
    FROM Utilizador
    INNER JOIN Pais
        ON Pais = IdPais
ORDER BY IdPais;

SELECT * FROM vwUtilizador;

DROP VIEW vwUtilizador;
```

Ilustração 31: View 1

(2) View Playlists. Mostra todas as playlists,e suas faixas, possuídas pelos utilizadores

```sql
-- Mostra todas as playlists,e suas faixas, possuídas pelos Utilizadores

CREATE VIEW vwPlayUtilizador AS
SELECT IdUtilizador, Username, P.IdPlaylist,P.Nome AS Playlist, F.IdFaixa,F.Nome AS Faixa
    FROM Utilizador
    INNER JOIN Playlist AS P
        ON Utilizador = IdUtilizador
        INNER JOIN playlist_has_faixa AS PF
            ON P.IdPlaylist = PF.IdPlaylist
            INNER JOIN Faixa AS F
            ON PF.IdFaixa = F.IdFaixa
    ORDER BY IdUtilizador;
```

Ilustração 32: View 2

(3) View Artista. Mostra os álbuns, e faixas desses álbuns, de todos os artistas

```sql
-- Mostra os albuns, e faixas desses albuns, de todos os artistas

CREATE VIEW vwArtista AS
SELECT IdArtista,Art.Nome AS Artista, IdAlbum,Al.Nome AS Album, IdFaixa,F.Nome AS Faixa
    FROM Artista AS Art
    INNER JOIN Faixa AS F
        ON IdArtista = Artista
        INNER JOIN Album AS Al
            ON IdAlbum = Album
    ORDER BY IdArtista;
```

Ilustração 33: View 3

(4) View Seguidor. Mostra os artistas seguidos por Utilizadores e mostra as faixas ordenadas pelas suas Reproduções (da mais reproduzida para a menos).

```sql
-- Mostra os artistas seguidos por Utilizadores e mostra as faixas ordenadas pelas suas Reproducoes c

CREATE VIEW vwSeguidor AS
SELECT Reproducoes,IdFaixa,F.Nome AS Faixa, Art.IdArtista, Art.Nome AS Artista, Uti.IdUtilizador, Uti.Username AS Utilizador
    FROM Utilizador AS Uti
    INNER JOIN utilizador_follows_artista AS UFA
        ON Uti.IdUtilizador = UFA.IdUtilizador
        INNER JOIN Artista AS Art
            ON UFA.IdArtista = Art.IdArtista
            INNER JOIN Faixa AS F
                ON Art.IdArtista = F.Artista
    ORDER BY Reproducoes DESC;
```

Ilustração 34: View 4

5.8. Definição e caracterização dos mecanismos de segurança em SQL (alguns exemplos)

Esta secção do relatório dedica-se à especificação de mecanismos de segurança dos tipos de utilizadores que podemos identificar na nossa base de dados.

- **Administrador:**

O administrador tem permissão para realizar qualquer ação sobre a base de dados (SELECT,INSERT,UPDATE,DELETE,CREATE,DROP), tendo assim as seguintes permissões:

```sql
-- Administrador, acesso total a base de dados

CREATE USER 'admin'@'localhost';
SET PASSWORD FOR 'admin'@'localhost' = PASSWORD('admin');

GRANT ALL ON arpeggio.* TO 'admin'@'localhost';
```

Ilustração 35:

- **Cliente:**

O cliente tem permissões mais complexas, tem a possibilidade de consultar todas as playlists,de inserir playlists novas e apagá-las. Pode também ver todos os artistas, os álbuns e as faixas.

```
-- Utilizador

CREATE USER 'utilizador'@'localhost';
SET PASSWORD FOR 'utilizador'@'localhost' = PASSWORD('utilizador');

GRANT SELECT,INSERT,DELETE ON arpeggio.Playlist TO 'utilizador'@'localhost';
GRANT SELECT ON arpeggio.Artista TO 'utilizador'@'localhost';
GRANT SELECT ON arpeggio.Album TO 'utilizador'@'localhost';
GRANT SELECT ON arpeggio.Faixa TO 'utilizador'@'localhost';
```

Ilustração 36:

- **Artista**
  O artista tem permissão para consultar e inserir faixas e álbuns:

```
-- Artista

CREATE USER 'artista'@'localhost';
SET PASSWORD FOR 'artista'@'localhost' = PASSWORD('artista');

GRANT SELECT,INSERT ON arpeggio.Faixa TO 'artista'@'localhost';
GRANT SELECT,INSERT ON arpeggio.Album TO 'artista'@'localhost';
```

Ilustração 37:

5.9. Revisão do sistema implementado com o utilizador

No fim da implementação do sistema foi marcada uma reunião final com o cliente, cujo propósito era demonstrar as funcionalidades implementadas na base de dados. A reunião foi concluída com sucesso, após a confirmação da satisfação total dos requisitos cliente pelo projeto.

6. Conclusões e trabalho futuro

A implementação da base de dados proposta exigiu um elevado nível de cuidado de forma a satisfazer todos os requisitos do cliente.Consideramos esta proposta de solução ao problema, como sendo uma forma útil de gerir uma base de dados de um serviço de música com as diferentes funcionalidades que este pode ter.

De maneira a assegurar o correto desenvolvimento da base de dados, foi necessário seguir diversos passos que permitiram o sucesso desta implementação. Estes passos constam neste relatório, sendo eles a análise de requisitos com o cliente, onde foi possível definir as entidades e relacionamentos do projeto que permitiram melhorar a gestão e o armazenamento de dados deste serviço.

Com o objetivo de diminuir os erros relativos à organização do grupo, foi seguida uma metodologia que permitiu sistematizar o processo de criação. Como guia para este processo usamos a metodologia sugerida pelo livro Database Systems: A Practical Approach to Design, Implementation, and Management 4th Edition, by Thomas Connolly and Carolyn Begg.

Começamos o projeto pela elaboração do modelo conceptual. Este modelo foi validado pelo cliente (satisfazendo todos os seus requisitos), que seguiu todo o processo de realização deste modelo. Numa segunda fase foi elaborado o modelo lógico que foi traduzido através do modelo conceptual mantendo o cumprimento dos requisitos (foi também validada pelo cliente). Por fim, foi implementado o esquema físico, chegando assim a conclusão do projeto, pois os objetivos foram todos atingidos.

Analisando a base de dados foi possível ver que existem certas funcionalidades que poderiam ser implementadas futuramente, como, por exemplo, um sistema de recomendações entre utilizadores e artistas, sendo uma maneira eficaz de transformar a ArpeggioMusic numa plataforma social.

Por último, podemos afirmar que as bases de dados são fundamentais, não só em serviços como este implementado, mas também para diversos outros tipos de serviços e para empresas, já que permite uma armazenação e tratamento de dados extremamente eficiente.

7. Referências bibliográficas

Connolly, T. and Begg, C. (2005). *Database Systems: A Practical Approach to Design, Implementation, and Management*. 4th ed.

Dev.mysql.com. (2017). *MySQL :: MySQL 5.7 Reference Manual :: 11.8 Data Type Storage Requirements*. [online] Available at: https://dev.mysql.com/doc/refman/5.7/en/storage-requirements.html [Accessed 27 Nov. 2017].

Gouveia, F. (2014). *Fundamentos de Bases de Dados*. 1st ed.

Sutori.com. (2017). *Sutori*. [online] Available at: https://www.sutori.com/story/history-of-music-streaming [Accessed 27 Nov. 2017].

8. Lista de Siglas e Acrónimos

BD	Base de Dados
ER	Entidade-Relacionamento
Ex	Exemplo
SGBD	Sistema de Gestão de Base de Dados