Apresentação do Trabalho Prático

MuDBa

Base de Dados Musical

Unidade Curricular de Bases de Dados

2017/2018

Janeiro 2018

Universidade do Minho

Francisco Oliveira | João Carvalho | Raul Vilas Boas | Vitor Peixoto

Base de Dados Relacional

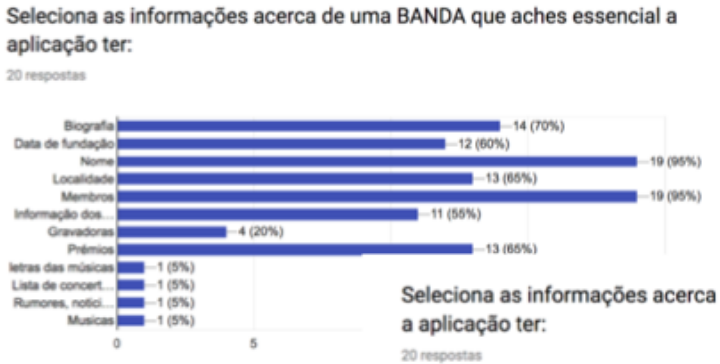
1ª Parte - MySQL

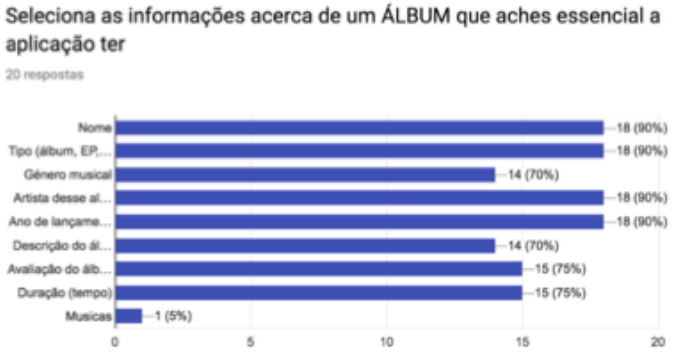
Fundamentação da implementação

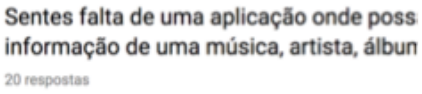
- A sociedade de hoje dedica bastantes horas do seu dia a ouvir música em todo o tipo diferente de plataformas, em qualquer sítio, a qualquer hora.
- É evidente que este público necessita de ter à sua disposição toda a informação que achem relevante acerca de uma música, artista ou álbum.
- Esta base de dados seria potenciada para um público jovem, mas também aficionados do mundo da música que sintam necessidade de saber mais sobre determinada música, artista ou álbum.
- Para analisar a vontade do público, criamos um questionário *online* que serviu como fundamentação da implementação e levantamento de requisitos.

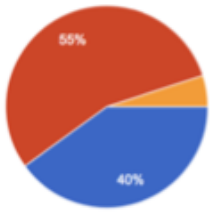
Levantamento de requisitos

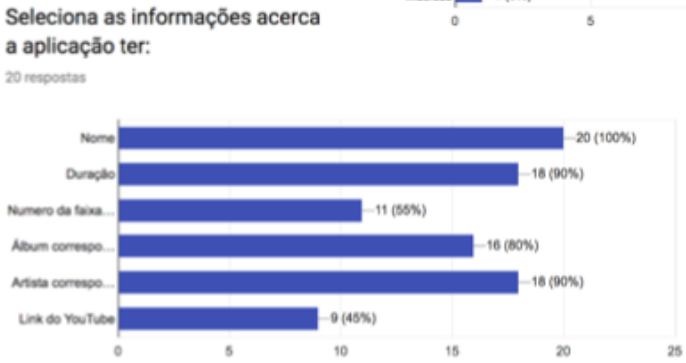
- Criamos um inquérito online para verificar a necessidade da existência desta base de dados e como levantamento de requisitos da base de dados.
- Tivemos 20 respostas de pessoas espalhadas por todo o país, de ambos os sexos e idades entre os 16 e 25 anos, sendo esta a faixa etária do público-alvo (o cliente desta base de dados), o inquérito será fidedigno acerca das necessidades dos nossos clientes.











Requisitos de descrição

- Analisando os requisitos definidos nos questionários recebidos, estas entidades foram definidas como essenciais pelo nosso cliente, pelo que terão de estar representadas na base de dados:
  - **Artista**
  - **Faixa**
  - **Gravação**
  - **Membro**

Requisitos de descrição

- Analisando os requisitos definidos nos questionários recebidos, estas entidades foram definidas como essenciais pelo nosso cliente, pelo que terão de estar representadas na base de dados:
  - **Artista**
  - **Faixa**
  - **Gravação**
  - **Membro**
- Também foi definido no questionário que a base de dados deveria distinguir os artistas em artistas com uma carreira a solo e bandas e que as gravações deveriam ser distinguidas entre álbuns, EP's e singles. Para os artistas criamos duas especializações e para as gravações definimos um atributos "tipo":
  - **Banda**
  - **Artista Solo**

Requisitos de descrição

- Analisando os requisitos definidos nos questionários recebidos, estas entidades foram definidas como essenciais pelo nosso cliente, pelo que terão de estar representadas na base de dados:
  - **Artista**
  - **Faixa**
  - **Gravação**
  - **Membro**
- Também foi definido no questionário que a base de dados deveria distinguir os artistas em artistas com uma carreira a solo e bandas e que as gravações deveriam ser distinguidas entre álbuns, EP's e singles. Para os artistas criamos duas especializações e para as gravações definimos um atributos "tipo":
  - **Banda**
  - **Artista Solo**
- A base de dados também deverá ter atributos suficientes para executar as *queries* necessárias às necessidades do cliente.

Requisitos de exploração

- Os requisitos de exploração irão ter um papel importante nesta base de dados, visto que o cliente irá ter a necessidade de aceder a toda a informação armazenada.
- Assim, o cliente definiu um conjunto de interrogações às quais a base de dados deverá ser capaz de responder:
  1. **Qual o álbum com maior pontuação de uma determinada banda?**
  2. **Qual o artista que lançou determinada música?**
  3. **Quais as gravações lançadas num determinado ano pela banda X e qual o nome do tipo de gravação?**
  4. **Quais as músicas lançadas por todos os artistas de um determinado país, por ordem alfabética?**
  5. **Quais as músicas de todas as bandas cujo guitarrista é X?**
- A base de dados também deverá permitir a inserção e modificação dos dados por parte de utilizadores responsáveis por tal tarefa.

Requisitos de controlo

- A base de dados está desenhada para utilizada pelos visitantes, o cliente. A manutenção desta terá que ser feita por uma equipa responsável por tal tarefa, enquanto que a atualização de informação será feita pelos artistas, responsáveis por manter apenas a sua informação atualizada.
- Definimos assim os seguintes utilizadores com os respetivos acessos à base de dados:
  - **Visitantes**: podem visualizar conteúdo apenas;
  - **Artistas**: podem adicionar, consultar e modificar conteúdo relacionado a si, mas não apagar;
  - **Administradores**: têm acesso total;

Modelo concetual

- A modelação concetual é um passo essencial para a criação do SBD, pois ajuda-nos a identificar entidades e atributos e a estabelecer as relações entre estes para obter uma esquema inicial da base de dados.

Modelo concetual: entidades

- Tal como foi abordado nos requisitos de descrição, as principais entidades serão: Artista; Gravação e Faixa. Optamos por criar novas entidades (Banda e Artista Solo), para criar uma relacão de especialização de Artista. A Banda é também composta por diversos membros, também caraterizados numa entidade.

  - **Artista**
  - **Artista Solo**
  - **Banda**
  - **Gravação**
  - **Faixa**
  - **Membro**

Modelo concetual: relacionamentos

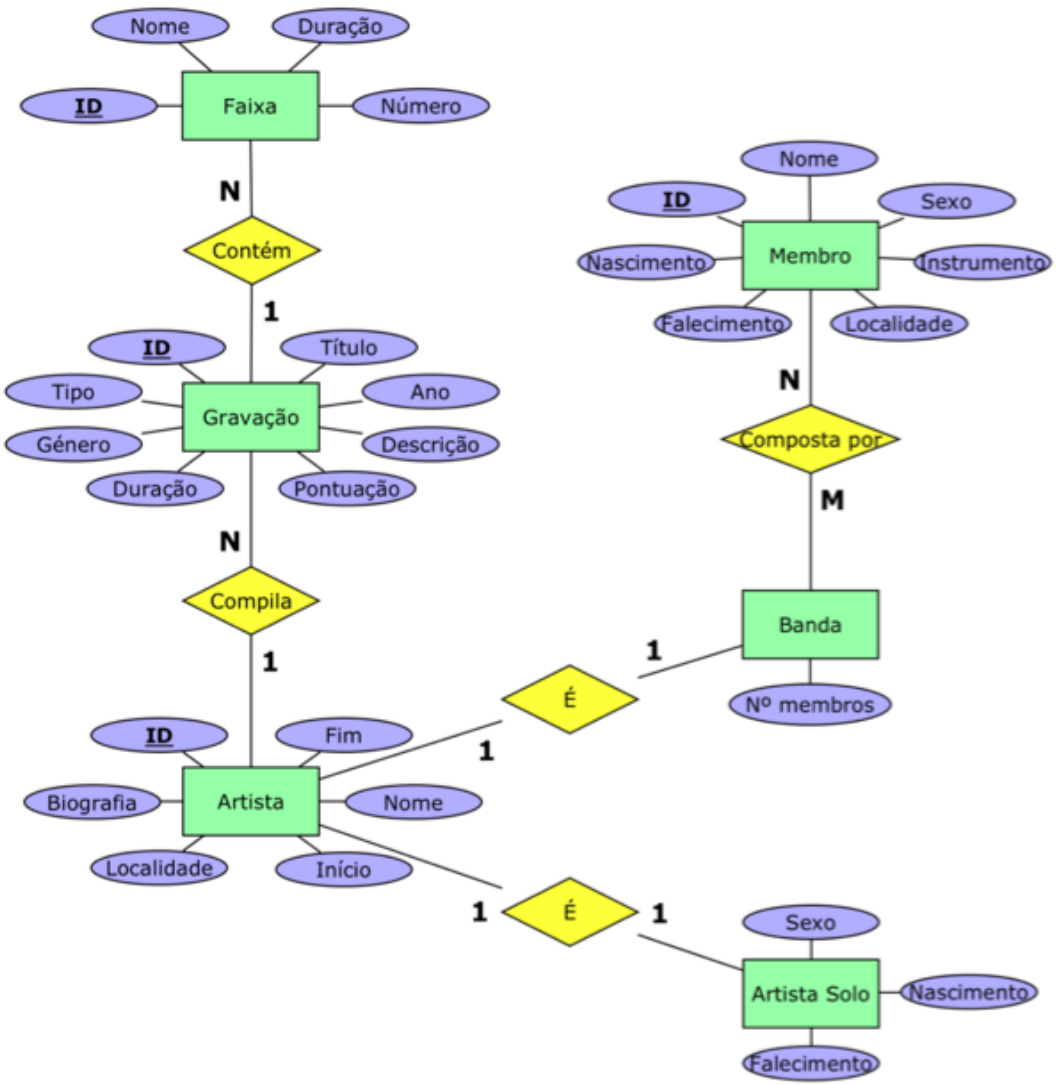
- Para conseguir responder às interrogações definidas pelo cliente, os relacionamentos entre as entidades têm de ser os mais corretos. Assim, definimos o seguinte conjunto de relacionamentos:

Entidade	Cardinalidade	Relação	Cardinalidade	Entidade
Gravação	1	Contém	N	Faixa
Artista	1	Compila	N	Gravação
Artista *Solo*	1	É	1	Artista
Banda	1	É	1	Artista
Banda	N	Composta por	N	Membros

Modelo concetual: atributos

- Para cada entidade definimos um atributo identificativo (ID). Os restantes atributos foram definidos de acordo com os requisitos de descrição definidos nos questionários:
  - **Faixa:** ID, Nome, Duração, Número;
  - **Gravação:** ID, Título, Ano, Descrição, Pontuação, Duração, Tipo, Género;
  - **Artista:** ID, Nome, Biografia, Localidade, Início, Fim;
  - **Artista Solo:** Sexo, Nascimento, Falecimento;
  - **Banda:** Nº de membros;
  - **Membro:** ID, Nome, Instrumento, Sexo, Localidade, Nascimento, Falecimento.

Modelo concetual: desenho

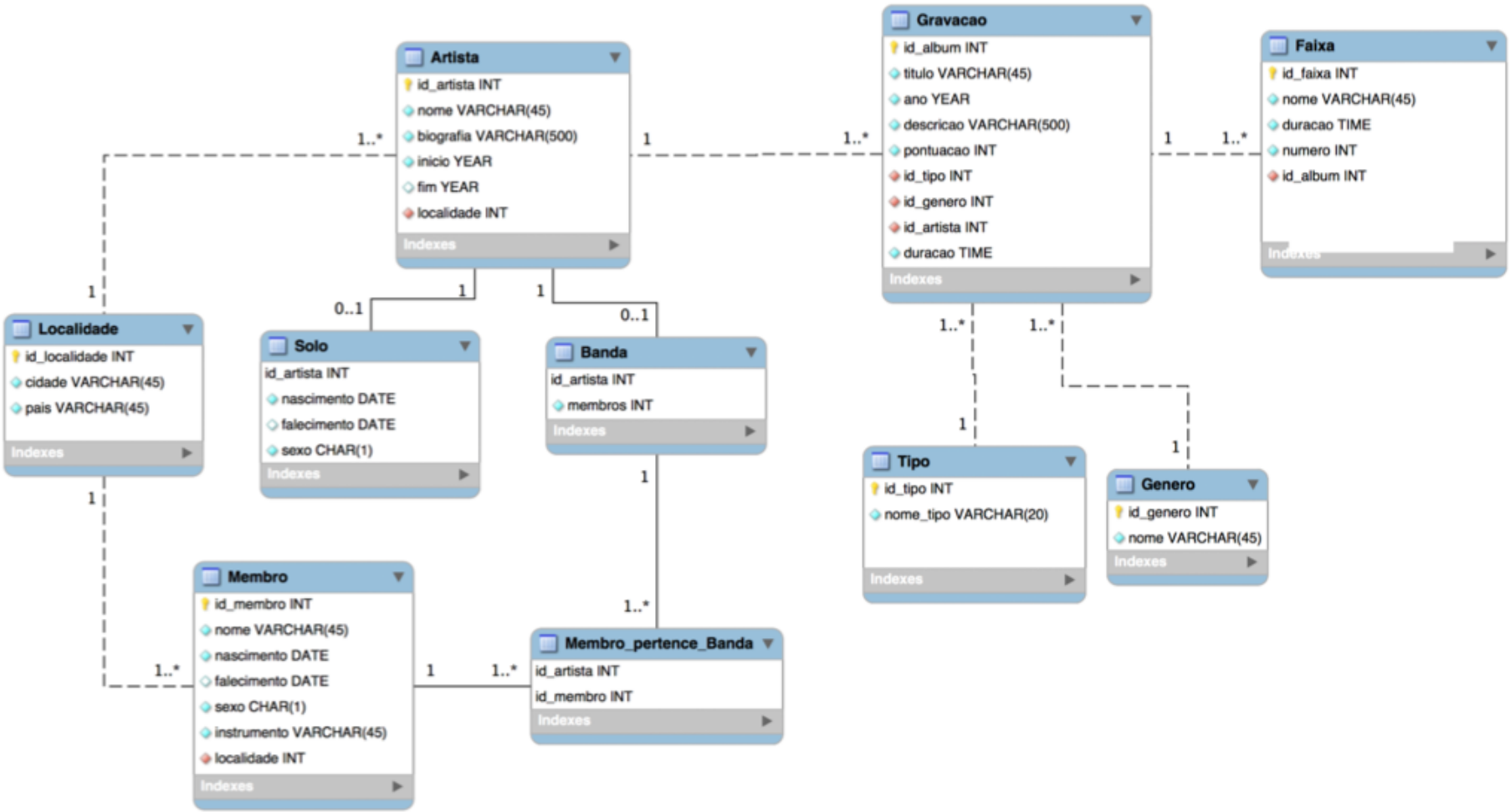


- Terminada a definição das entidades, relacionamentos e atributos, podemos desenhar o modelo concetual.
- Este modelo do nosso ponto de vista consegue responder aos requisitos levantados pelo cliente, pelo que o apresentamos, tendo sido validado.

Modelo lógico: adaptação

- Depois de validado o modelo de dados concetual, passamos à modelação lógica do sistema.
- Relativamente às entidades, separamos as entidades fortes das fracas, optando por definir alguns atributos como entidades fracas (localidade, tipo e género) para aumentar a correção de dados, usando um mecanismo de coesão de dados denominado Integridade Referencial.
- Os atributos definidos na modelação concetual mantiveram-se os mesmos para o modelo lógico.
- Relativamente aos relacionamentos, todas as entidades geradas por Integridade Referencial estarão relacionadas com a sua entidade correspondente com um relacionamento de cardinalidade 1:N, tal como os relacionamentos entre Artista, Gravação e Faixa. As relações de especialização de Artista serão de cardinalidade 0..1:1. Por fim, o relacionamento entre Membro e Banda será de N:N.

Modelo lógico: desenho

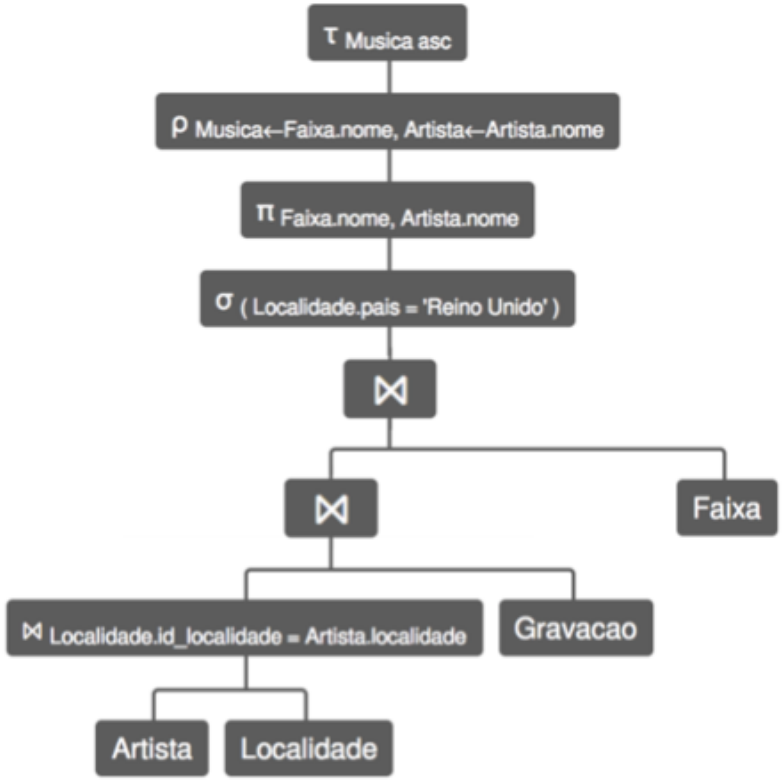


Modelo lógico: normalização

- Passamos para a validação do modelo através da normalização. Este processo ajuda a reduzir redundância e incoerências na base de dados.
- Este processo foi dividido em 2 fases: definição das dependências funcionais e aplicação das formas de normalização.
- **1ª Forma normal:** A nossa base de dados está normalizada à 1ª forma, visto que não existem tuplos com atributos compostos em nenhuma tabela (há atomicidade em todos os atributos de todos os tuplos).
- **2ª Forma normal:** Através da análise das dependências funcionais da base de dados verificamos que todos os atributos de uma tabela dependem unicamente do conjunto de chaves primárias de cada tabela, pelo que respeita a 2ª forma normal.
- **3ª Forma normal:** Conclui-se que a nossa base de dados está normalizada, uma vez que respeita a 3ª forma normal e as anteriores, visto que todos os atributos não podem ser obtidos através de outros dependentes da mesma chave primária.

Modelo lógico: interrogações

- Vamos demonstrar apenas a validação de uma interrogação (uma das mais complexas) usando a álgebra relacional.
- O facto de a álgebra desta interrogação permitir obter um resultado final valida o modelo. Todas as interrogações foram também validadas usando álgebra relacional.
- **Query:** Quais as músicas lançadas por todos os artistas de um determinado país (Reino Unido, pe.), por ordem alfabética?



$$\tau_{\text{Musica}}\,\rho_{\text{Musica}\leftarrow\text{Faixa.nome, Artista}\leftarrow\text{Artista.nome}}\,\pi_{\text{Faixa.nome, Artista.nome}}\,(\sigma_{\text{Localidade.pais = 'Reino Unido'}}$$

$$\text{LOCALIDADE} \bowtie_{\text{Localidade.id\_localidade = Artista.localidade}} (\text{ARTISTA} \bowtie (\text{GRAVACAO} \bowtie \text{FAIXA})))$$

Modelo lógico: transações

- Aqui verificamos a capacidade do modelo lógico efetuar transações na base de dados, utilizando para isso 3 transações, uma inserção, uma atualização e uma remoção:
  1. **Inserir:** adiciona uma nova gravação com os respetivos atributos;
  2. **Atualizar:** alterar atributos de um artista;
  3. **Remover:** remove um membro das bandas que pertence quando este falece.
- Concluímos que o modelo lógico permite efetuar as transações, respeitando as caraterísticas ACID das transações em bases de dados relacionais.

Modelo físico: escolha do SGBD

- O SGBD escolhido foi o *MySQL*, auxiliado pelo *MySQL Workbench*.
- Para além de ter sido o SGBD que foi usado nas aulas, não havendo necessidade de adaptação, a escolha teve em conta as caraterísticas do *MySQL*:
- Está orientado para aplicações *Web*, sendo esse uma das aplicações desta base de dados;
- A sua licença é gratuita e não necessita de *hardware* de grande nível, o que é ideal para este projeto, visto que não temos orçamento para custos;
- A ferramenta *Workbench* é outra das principais vantagens deste SGBD visto que permite gerar o modelo físico através do diagrama lógico e posteriormente alterar, desenvolver e consultar a base de dados.

Modelo físico: tradução do lógico

- O *MySQL Workbench* tem disponível a opção "*Forward Engineer"* que, após desenhar o modelo lógico, gera automaticamente o *script* para a criação de todas as tabelas, atributos, relacionamentos, chaves, etc. Esse *script* é depois executado automaticamente, criando o modelo físico da base de dados.

Modelo físico: *queries* em SQL

- As interrogações propostas pelo cliente através dos inquéritos têm de ser agora passadas para *SQL*, após terem sido validadas pela álgebra relacional.
- Neste caso iremos apenas demonstrar a interrogação que já havíamos mostrado na validação através da álgebra relacional (ver álgebra relacional):
- **Query:** Quais as músicas lançadas por todos os artistas de um determinado país (Reino Unido, pe.), por ordem alfabética?

```
SELECT f.nome AS Musica, a.nome AS Artista
FROM Artista a
  INNER JOIN Localidade AS l ON l.id_localidade = a.localidade
  INNER JOIN Gravacao AS g ON g.id_artista = a.id_artista
  INNER JOIN Faixa AS f ON f.id_album = g.id_album
    WHERE (l.pais = 'Reino Unido')
    ORDER BY Musica;
```

Modelo físico: transações

- Uma transação é uma operação em *SQL* caraterizada pelas propriedades ACID.
- As transações propostas foram as seguintes:
  1. **Inserir:** adiciona uma nova gravação com os respetivos atributos;
  2. **Atualizar:** alterar atributos de um artista;
  3. **Remover:** remove um membro das bandas quando este falece.
- Inicialmente mapeamos as transações numa tabela de relações:

	(A)				(B)				(C)			
	I	R	U	D	I	R	U	D	I	R	U	D
Artista							X					
Gravação	X											
Membro											X	
M_p_B												X
Banda											X*	

I – Insert; **R** – Read; **U** – Update; **D** – Delete; **M_p_B** – Tabela Membro_pertence_Banda;
* – Este update é feito pelo trigger a)

- De seguida escrevemos as transações em *SQL* criando *procedures*:

Modelo físico: transações

1. **Inserir:** adiciona uma nova gravação com os respetivos atributos:

```
CREATE PROCEDURE inserirGravacao (IN id_album INT,
                                  IN titulo VARCHAR(45),
                                  IN ano YEAR,
                                  IN descricao VARCHAR(500),
                                  IN pontuacao INT,
                                  IN id_tipo INT,
                                  IN id_genero INT,
                                  IN id_artista INT)
BEGIN
DECLARE erro BOOL DEFAULT 0;
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET erro = 1;
START TRANSACTION;
    INSERT INTO gravacao VALUES (id_album, titulo, ano, descricao, pontuacao,
    id_tipo, id_genero, id_artista);
IF erro
THEN ROLLBACK;
ELSE COMMIT;
END IF;
END
```

Modelo físico: transações

2. **Atualizar:** alterar atributos de um artista:

```
CREATE PROCEDURE atualizaArtista (IN nID INT,
                                  IN nNome VARCHAR(45),
                                  IN nBiografia VARCHAR(500),
                                  IN nInicio YEAR,
                                  IN nFim YEAR,
                                  IN nLocalidade INT)
BEGIN
DECLARE erro BOOL DEFAULT 0;
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET erro = 1;
START TRANSACTION;
    UPDATE Artista a SET a.nome = nNome,
                         a.biografia = nBiografia,
                         a.inicio = nInicio,
                         a.fim = nFim,
                         a.localidade = nLocalidade
                   WHERE a.id_artista = nID;
IF erro
THEN ROLLBACK;
ELSE COMMIT;
END IF;
END
```

Modelo físico: transações

3. **Remover:** remove um membro das bandas quando este falece.

```
CREATE PROCEDURE removeMembro (IN memID INT,
                               IN memFalecimento DATE)
BEGIN
DECLARE erro BOOL DEFAULT 0;
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET erro = 1;
START TRANSACTION;
    UPDATE Membro m SET m.falecimento = memFalecimento
                    WHERE id_membro = memID;
    DELETE FROM Membro_pertence_Banda
                    WHERE id_membro = memID;
IF erro
THEN ROLLBACK;
ELSE COMMIT;
END IF;
END
```

Espaço em disco e crescimento

- Para o cliente conseguir avaliar se o seu *hardware* suporta esta base de dados é essencial calcular o espaço ocupado em disco pela mesma e determinar uma estimativa de crescimento da mesma no futuro.
- Chegamos então ao resultado de 13 930 bytes para o espaço ocupado em disco pelo povoamento inicial.
- Definimos também estimativas de taxas de crescimento para cada tabela da base de dados e, tendo em conta as taxas de crescimento, calculamos um valor aproximado do espaço ocupado em disco por esta base de dados daqui a 1 e daqui a 5 anos, tendo obtido 68 553 bytes e 108 507 bytes, respetivamente.
- Ver estimativas >

Modelo físico: triggers

- Os *triggers* são essenciais para atualizar informações automaticamente quando uma informação da qual dependem é alterada ou para criar restrições na inserção de dados. Neste caso há alguns *triggers* que serão bastante úteis na base de dados:
  1. **Atualizar o número de membros de uma banda quando um elemento falece;** (ver SQL)
  2. **Declarar fim da carreira quando artista solo falece;** (ver SQL)
  3. **Impedir que um artista solo seja também uma banda;** (ver SQL)
  4. **Impedir que uma banda seja também um artista solo.** (ver SQL)

Modelo físico: vistas

- As vistas são importantes na otimização de consultas e nos mecanismos de segurança, podendo juntar várias informações de várias tabelas em tabelas fictícias.
- Criamos duas vistas:
  1. **Tabela com as bandas e os seus membros;** (ver SQL)
  2. **Tabela com todos os álbuns "Rock".** (ver SQL)

Modelo físico: segurança

- Para além das vistas, podemos definir mais mecanismos de segurança como a criação de utilizadores e as respetivas permissões. Analisando os requisitos de controlo fornecidos pelo cliente, podemos definir três tipos de utilizador: Visitante; Artista e Administrador, com as seguintes permissões:

Tabela	Administrador				Artista				Visitante			
	S	I	U	D	S	I	U	D	S	I	U	D
Faixa	X	X	X	X	X	X			X			
Gravação	X	X	X	X	X	X			X			
Artista	X	X	X	X	X		X		X			
Solo	X	X	X	X	X		X		X			
Banda	X	X	X	X	X		X		X			
Membro	X	X	X	X	X	X	X	X	X			
M_p_B	X	X	X	X	X	X	X	X	X			
Tipo	X	X	X	X	X				X			
Género	X	X	X	X	X				X			
Localidade	X	X	X	X	X				X			

Modelo físico: segurança

- Utilizadores definidos:

```
CREATE USER 'admin'@'localhost';
       SET PASSWORD FOR 'admin'@'localhost' = 'mudbadmin';

CREATE USER 'artista1'@'localhost';
       SET PASSWORD FOR 'artista1'@'localhost' = 'artista1';
-- repetir para cada artista

CREATE USER 'visitante'@'localhost';
  SET PASSWORD FOR 'visitante'@'localhost' = 'visit';
```

- Privilégios dos utilizadores:

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

GRANT SELECT, UPDATE ON mudba.Artista TO 'artista1'@'localhost';
REVOKE INSERT, DELETE ON mudba.Artista FROM 'artista1'@'localhost';
-- repetir o processo de GRANT e REVOKE para as outras tabelas.

GRANT SELECT ON mudba.* TO 'visitante'@'localhost';
REVOKE UPDATE, INSERT, DELETE ON mudba.* FROM 'visitante'@'localhost';
```

Modelo físico: validação

- Após concluirmos a modelação física e termos respondido com sucesso a todos os requisitos apresentamos o projeto ao cliente que o validou.
- Concluímos assim o desenvolvimento da base de dados e podemos finalmente tirar algumas conclusões acerca do projeto.

Base de Dados Não Relacional

2ª Parte – Neo4j

Fundamentação da transição

- Quais os motivos para migrar uma base de dados relacional para não relacional?
  1. Enquanto que em *SQL* o desempenho das *queries* piora conforme a quantidade de dados aumenta, em bases de dados orientadas por grafos, a *performance* mantém-se relativamente constante.
  2. Os grafos não estão restritos a um determinado modelo de dados, podendo adicionar novas “entidades”, “atributos” e “relacionamentos” de livre vontade, se acharmos que for benéfico para o desempenho da base de dados.
  3. Visto que a nossa base de dados irá armazenar toneladas de informação de música, artistas, álbuns, etc., é essencial a BD estar preparada para grandes quantidades de dados, que é uma das vantagens das bases de dados *NoSQL*.

Construção do sistema

- O *MySQL Workbench* permite criar rapidamente *.csv* com toda a informação da base de dados e o *Neo4j* disponibiliza comandos capazes de carregar todos os tuplos como nós para o grafo. (ver *Cypher*)
- No entanto isto iria obrigar a eliminar propriedades que não seriam necessárias (chaves primárias e estrangeiras, etc.) podendo até não tirarmos proveito máximo das propriedades de uma base de dados *NoSQL*.
- Optamos então por desenhar todos os nós e relações do princípio, acrescentando ainda mais informação do que a que se encontrava no povoamento inicial em *SQL*.
- Esta opção pode, no entanto, levar a erros na transição. Por isso, numa base de dados de grande dimensão, o melhor método a ser adotado seria mesmo carregar os *.csv*.

Construção do sistema: nós

- **Artista:** As propriedades definidas para o artista foram as mesmas definidas no modelo relacional, excetuando a biografia do artista (removida para simplificar):

Banda →
- Nome
- Início
- Fim (se for o caso)
- Nº membros

Solo →
- Nome
- Início
- Fim (se for o caso)
- Nascimento
- Falecimento (se for o caso)
- Sexo

```
CREATE (BonIver:Banda {nome:'Bon Iver', inicio:2006, membros:4}),
       (Radiohead:Banda {nome:'Radiohead', inicio:1985, membros:5}),
       (RHCP:Banda {nome:'Red Hot Chili Peppers', inicio:1983, membros:4}),
       (Coldplay:Banda {nome:'Coldplay', inicio:1996, membros:4}),
       (TheKillers:Banda {nome:'The Killers', inicio:2001, membros:4}),
       (TheChainsmokers:Banda {nome:'The Chainsmokers', inicio:2012, membros:2})

CREATE (Kanye:Solo {nome:'Kanye West', inicio:1996, nascimento:19770608, sexo:'m'}),
       (DavidBowie:Solo {nome:'David Bowie', inicio:1964, fim:2016, nascimento:19470108,
       falecimento:20160110, sexo:'m'})
```

Construção do sistema: nós

- **Gravação:** As propriedades definidas para a gravação foram as mesmas definidas no modelo relacional, excetuando a descrição (removida para simplificar), a duração e as chaves:
  - Nome
  - Ano
  - Pontuação (0-10)

```
CREATE (OKComputer:Album {nome:'OK Computer', ano:1997, pontuacao:10}),
       (ParachutesAlbum:Album {nome:'Parachutes', ano:1999, pontuacao:7}),
       (Collage:EP {nome:'Collage EP', ano:2016, pontuacao:3}),
       (SJLT:Single {nome:'Something Just Like This', ano:2017, pontuacao:5}),
       (AROBTTH:Album {nome:'A Rush Of Blood To The Head', ano:2002, pontuacao:9}),
       (HeroesAlbum:Album {nome:'Heroes', ano:1977, pontuacao:8}),
       ...
```

Construção do sistema: nós

- **Faixa:** As propriedades definidas para as faixas foram as mesmas definidas no modelo relacional, excetuando as chaves:
  - Nome
  - Duração
  - Número (ordem da faixa na sua gravação)

```
CREATE (Yellow:Faixa {nome:'Yellow', duracao:4.45, numero:5}),
       (Heroes:Faixa {nome:'Heroes', duracao:6.12, numero:3}),
       (VivaLaVida:Faixa {nome:'Viva La Vida', duracao:4.02, numero:7}),
       (NoSurprises:Faixa {nome:'No Surprises', duracao:3.85, numero:10}),
       ...
```

Construção do sistema: nós

- **Localidade:** Optamos por manter as localidades como um tipo de dados próprio por questões de integridade de dados. As propriedades são as mesmas que foram definidas no modelo relacional, excetuando as chaves:
  - Cidade
  - País

```
CREATE (LA:Localidade {cidade:'Los Angeles', pais:'Estados Unidos da América'}),
       (Londres:Localidade {cidade:'Londres', pais:'Reino Unido'}),
       (NovaIorque:Localidade {cidade:'Nova Iorque', pais:'Estados Unidos da América'}),
       ...
```

Construção do sistema: nós

- **Género musical:** Optamos também por manter os géneros como um tipo de dados próprio pelos mesmo motivos. As propriedades são as mesmas que foram definidas no modelo relacional, excetuando as chaves:
    - Nome

```
CREATE (Rock:Genero {nome:'Rock'}),
       (Pop:Genero {nome:'Pop'}),
       (Indie:Genero {nome:'Indie'}),
       (HipHop:Genero {nome:'Hip Hop'}),
       (Alt:Genero {nome:'Alternativo'})
```

Construção do sistema: nós

- **Membro:** Aqui, removemos o instrumento do membro, que passará para o relacionamento e nesse mesmo relacionamento será adicionada a propriedade “ano de entrada na banda”**.** As propriedades foram as seguintes:
  - Nome
  - Nascimento
  - Falecimento (se for o caso)
  - Sexo

```
CREATE (JustinVernon:Membro {nome:'Justin Vernon', nascimento:'19810430', sexo:'m'}),
       (MichaelNoyce:Membro {nome:'Michael Noyce', nascimento:'19850112', sexo:'m'}),
       (SeanCarey:Membro {nome:'Sean Carey', nascimento:'19800922', sexo:'m'}),
       (MattMcCaughan:Membro {nome:'Matt McCaughan', nascimento:'19820801', sexo:'m'}),
       ...
```

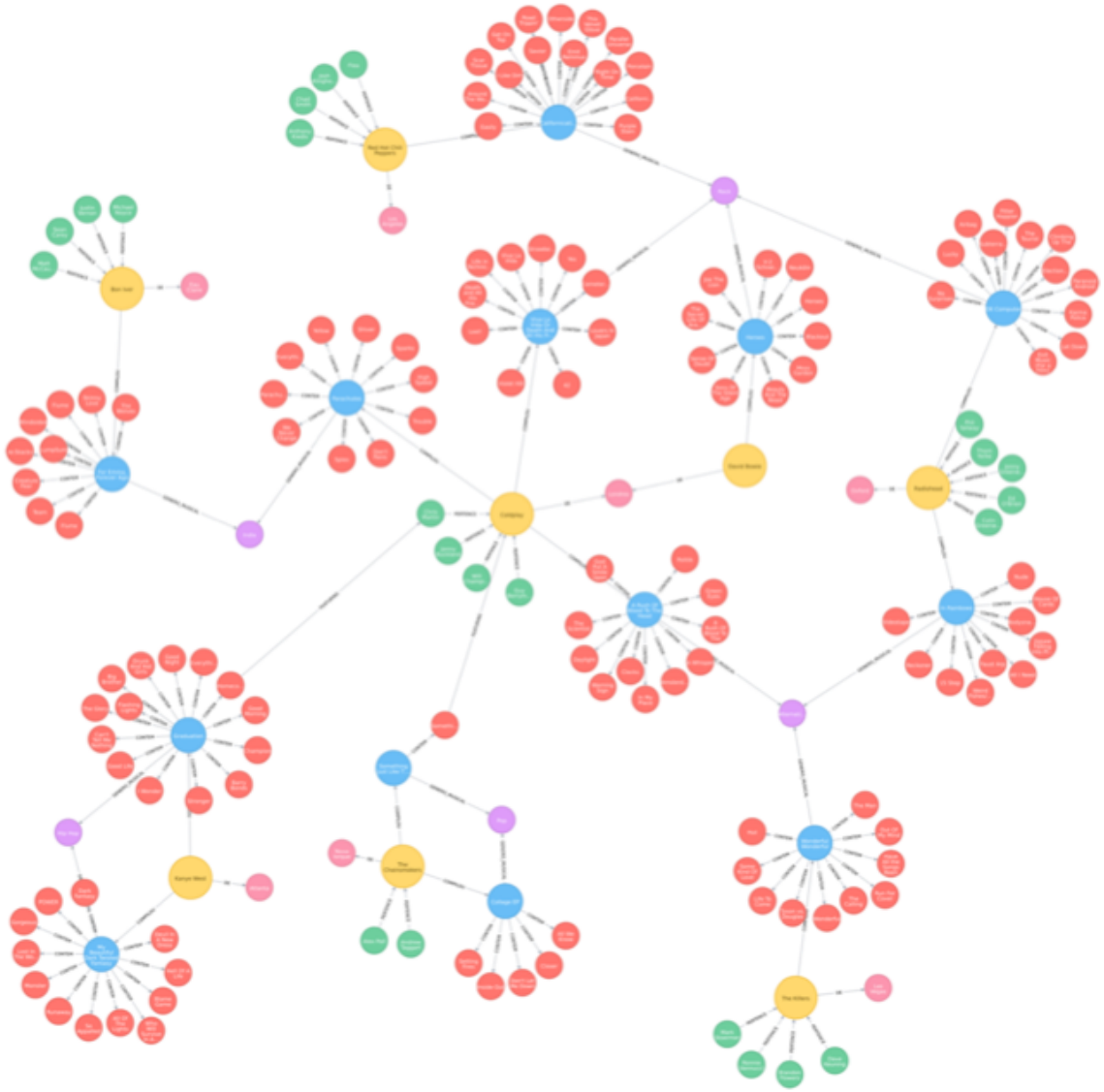
Construção do sistema: relacionamentos

- Os relacionamentos definidos seguiram a estrutura definida na modelação relacional, sendo que adicionamos alguns relacionamentos, dada a flexibilidade das bases de dados não relacionais.

  Assim, definimos os seguintes relacionamentos:

  1. **Artista → compila → Gravação;** (ver *Cypher*)
  2. **Artista → de → Localidade;** (ver *Cypher*)
  3. **Gravação → tem → Género musical;** (ver *Cypher*)
  4. **Gravação → contém → Faixa;** (ver *Cypher*)
  5. **Membro → pertence a → Banda;** (ver *Cypher*) (este relacionamento irá ter duas propriedades: ano de entrada e função)
  6. **Artista / Membro → colabora em → Faixa;** (ver *Cypher*) (novo relacionamento)

Grafo final



Índices e restrições

- Dada a importância dos índices na execução de *queries* em *Neo4j* optamos por definir um conjunto de índices nesta base de dados.

Banda:
- **Nome (*unique*)**
- Início
- Fim
- Número de membros

Membro:
- **Nome**
- Nascimento
- Falecimento
- Sexo

Gravação (álbum, *EP*, *single*):
- **Nome**
- Ano
- Pontuação

Faixa:
- **Nome**
- Duração
- Número

Localidade:
- **Cidade (*unique*)**
- **País (*unique*)**

Género musical:
- **Nome (*unique*)**

Solo:
- **Nome (*unique*)**
- Início
- Fim
- Nascimento
- Falecimento
- Sexo

- Achamos também conveniente definir restrições de unicidade para algumas das propriedades.

Índices e restrições

- *Node key* é uma propriedade que só se encontra disponível na versão *Enterprise* e é usado aqui visto que *Unique* não permite usar duas propriedades simultaneamente.

```
CREATE CONSTRAINT ON (c:Banda) ASSERT c.nome IS UNIQUE;
CREATE CONSTRAINT ON (s:Solo) ASSERT s.nome IS UNIQUE;
CREATE CONSTRAINT ON (g:Genero) ASSERT g.nome IS UNIQUE;
CREATE CONSTRAINT ON (l:Localidade) ASSERT (n.pais, n.cidade) IS NODE KEY;
```

- A definição de propriedades como únicas já as define como índices, logo só restam as seguintes:

```
CREATE INDEX ON :Localidade(cidade);
CREATE INDEX ON :Localidade(pais);
CREATE INDEX ON :Album(nome);
CREATE INDEX ON :EP(nome);
CREATE INDEX ON :Single(nome);
CREATE INDEX ON :Faixa(nome);
CREATE INDEX ON :Membro(nome);
```

Tradução das interrogações

- As interrogações traduzidas foram as mesmas usadas no modelo relacional deste projeto, e como tal iremos apenas demonstrar uma das interrogações:
- **Query:** Quais as músicas lançadas por todos os artistas de um determinado país (Reino Unido, pe.), por ordem alfabética?

CYPHER: (ver resultado)

```
MATCH (a)-[:COMPILOU]->(g), (g)-[:CONTEM]->(f:Faixa), (a)-[:DE]->(l:Localidade)
WHERE l.pais = 'Reino Unido'
RETURN DISTINCT a.nome AS Artista, g.nome AS Gravacao, f.nome AS Faixa
ORDER BY a.nome, g.nome, f.nome;
```

SQL: (ver resultado)

```
SELECT a.nome AS Artista, g.titulo AS Gravacao, f.nome AS Musica
FROM Artista a
   INNER JOIN Localidade AS l ON l.id_localidade = a.localidade
   INNER JOIN Gravacao AS g ON g.id_artista = a.id_artista
   INNER JOIN Faixa AS f ON f.id_album = g.id_album
      WHERE l.pais = 'Reino Unido'
      ORDER BY Artista, Gravacao, Musica;
```

Conclusão

- Os resultados obtidos nas *queries* foram os esperados e coincidem com os obtidos em *SQL*, pelo que podemos afirmar que tanto a implementação do sistema como a tradução das *queries* para *Cypher* estão corretas.
- Validado o sistema podemos concluir o desenvolvimento do sistema e tirar ilações acerca do trabalho desenvolvido.