Universidade do Minho

Escola de Engenharia

MIEI

Unidade Curricular de

Bases de Dados

Ano Lectivo de 2015/2016

Jonas' Discos

Afonso Silva A70387

Alfredo Gomes A71655

Humberto Vaz A73236

João Luís Vieira A71489

Janeiro 2016

Data de Recepção	
Responsável	
Avaliação	
Observações	

Jonas' Discos

Afonso Silva A70387

Alfredo Gomes A71655

Humberto Vaz A73236

João Luís Vieira A71489

Janeiro 2016

Resumo

No âmbito da unidade curricular de *Bases de Dados* propôs-se a criação de um sistema relacional de um tema à escolha, que seja útil e que possa ser implementado na vida real. O tema escolhido pelo grupo foi a gestão de uma cadeia de lojas de discos de vinil à qual se apelidou de Jonas' Discos.

Com este trabalho pretende-se, numa primeira fase, planear o sistema de base de dados escolhido, ou seja, fazer uma recolha e análise de requisitos e posteriormente um diagrama ER de maneira a relacionar as entidades e os seus atributos.

Em primeiro lugar, fez-se uma introdução ao tema, apresentando e explicando o caso que se pretende melhorar. Nesta situação, o objetivo é criar um sistema de bases de dados informatizado na cadeia de lojas em questão.

De seguida, realizou-se a recolha de informação através de pesquisas e debate em grupo para analisar os pré-requisitos. Concluída esta parte, seguiu-se a realização do modelo conceptual usando a aplicação `brModelo`.

Na segunda fase do projeto, foi realizado o modelo lógico com base no modelo conceptual e posteriormente a sua validação tanto a nível da normalização como também a nível das transações do utilizador. Depois de colocado este modelo no Workbench, procedeu-se à definição do tamanho inicial desta BD e análise do crescimento no futuro e também à sua revisão com os futuros utilizadores do sistema.

Finda a parte do modelo lógico, implementou-se o modelo físico e efetuou-se o povoamento. Também nesta fase fez-se uma análise às transações, estimativa do espaço em disco com base no SGBD e a definição das vistas dos utilizadores e regras de acesso.

Por fim, efetuaram-se as conclusões tiradas do trabalho realizado, concluindo-se que o projeto aqui apresentado é viável e robusto.

Área de Aplicação: desenho e arquitetura de sistemas de bases de dados;
Palavras-Chave: bases de dados relacionais; modelo conceptual; entidades; relacionamentos; atributos; modelo lógico; SQL;

Índice

Índice de Figuras

Índice de Tabelas

1. Introdução

Para este projeto optou-se por modelar e construir uma possível representação relacional de apoio a uma cadeia de lojas de discos *Vinil*.

1.1. Contextualização

O CEO da conhecida cadeia de venda de discos a retalho, a “Jonas' Discos”, sentiu a necessidade de informatizar o seu serviço. Para tal contactou os elementos deste grupo, encarregando-os da tarefa de criar uma base de dados expansível, segura e robusta que permita facilitar a gestão das suas lojas, fornecedores, clientes e discos.

1.2. Apresentação do Caso de Estudo

O caso de estudo aqui selecionado remete-nos para uma **cadeia de lojas de discos *Vinil*** fictícia. Esta cadeia, de nome “Jonas' Discos” apresenta um modelo de venda bastante tradicional, sendo que cada loja emprega um ou mais funcionários responsáveis pelo atendimento ao público. Tal como o nome indica, esta companhia é especialista na venda a retalho de discos *vinil*, sendo por isso importante manter os discos organizados por loja e devidamente catalogados.

Uma vez que foi fundada antes do advento dos computadores pessoais, os catálogos de música da “Jonas' Discos” consistem basicamente num conjunto de *dossiers* que contêm as informações relevantes de cada disco. Para além disso, sendo o consumidor de *Vinis* uma espécie cada vez mais em vias de extinção, torna-se imperativo para a “Jonas' Discos” manter uma relação mais próxima com os clientes, daí terem sido criados os estatutos de cliente *Normal*, *Golden* e *Platina*. O primeiro, tal como o nome indica, aplica-se aos clientes regulares que estão registados na loja. Estes clientes (tais como os de escalões superiores) recebem notificações dos novos lançamentos de discos (por correio através dos catálogos, e por telefone se assim o cliente consentir). Após ter comprado 50 ou mais discos, o estatuto do cliente passa a ser *Golden,* que permite ao cliente usufruir de um desconto de 50% num disco na compra de 5 discos. Ao atingir o número de 100 discos comprados, o cliente é presenteado com o estatuto *Platina*, que lhe permite a oferta de um disco, na compra de 5 discos.

O registo dos clientes é também bastante rudimentar, consistindo simplesmente numa lista que contém os nomes dos clientes e o respetivo estatuto.

Para além disso, está também registado num caderno a lista actual dos fornecedores de discos, sendo que é nesta lista que se encontram escritos os seus contactos, bem como as informações das compras a eles subjacentes.

É importante relembrar que a “Jonas' Discos” é composta por mais do que uma loja, daí ser de extrema importância monitorizar individualmente cada um dos estabelecimentos para fins estatísticos e de *profiling*. Por isso, cada loja está encarregue de tomar nota do seu total facturado, bem como o total gasto na aquisição de discos, para assim se poder calcular o lucro registado. Para além disso, o estatuto do cliente é transversal a todas as lojas, i.e., um cliente *Golden* é um cliente *Golden* em todas as lojas pertencentes à Jonas' Discos.

1.3. Motivação e Objectivos

Tal como foi exposto anteriormente, é possível concluir que a “Jonas' Discos” tem muito a ganhar com uma informatização dos seus serviços. Deste modo, o principal âmbito deste projecto é o de adaptar o modelo existente para um electrónico, tendo como principal recurso um sistema de bases de dados.

À primeira vista, uma loja de *vinis* pode beneficiar bastante de um catálogo electrónico dos discos que vende, seja para fins de consulta, seja para fins de gestão dos mesmos. Um catálogo electrónico dos discos pode revelar-se também útil para que os clientes possam pesquisar os discos disponíveis, e até talvez, obterem recomendações e novidades de novos lançamentos.

Numa vertente mais administrativa, a informatização deste serviço permite um maior controlo sobre os processos de compra e venda de discos. O registo electrónico dos clientes permite uma maior facilidade no acto de *profiling dos mesmos*, pois as suas informações encontrar-se-ão centralizadas e com uma maior facilidade de acesso.

1.4. Estrutura do Relatório

O relatório aqui apresentado abordará de uma forma sistemática a modelação e estruturação de uma possível solução para a resolução do problema apresentado. Numa primeira fase serão apresentados os requisitos levantados durante o estudo do caso, sendo estes seguidos pelo diagrama ER correspondente. Este diagrama será explicado ao detalhe, para que possa ser possível apresentar as diversas conclusões.

2. Pré-requisitos

Antes da realização do modelo conceptual foi necessário estruturar um pensamento sobre como este iria ser elaborado com base no processo explicado no capítulo 1. Para isto foi necessário a pesquisa sobre este tema e o debate dos elementos do grupo de maneira a chegar a um consenso. Depois do referido debate chegou-se às conclusões apresentadas de seguida.

2.1. Entidades

A nossa base de dados terá quatro entidades, sendo elas o Disco Vinil que se pretende vender, o Fornecedor que irá fornecer a loja, neste caso, com discos, o Cliente que frequenta a loja e compra discos e a Loja propriamente dita, local onde o processo entre todas as outras entidades acontece.

Entidade	**Descrição**	**Sinónimos**	**Ocorrência**
Loja	Termo geral para os estabelecimentos de venda de discos de vinil	Estabelecimento de venda	Cada loja pertence à nacional Jonas' Discos
Fornecedor	Termo geral para todos os fornecedores das lojas	Provedor	Cada fornecedor da Jonas' Discos
Disco Vinil	Termo geral para todos os discos de vinil vendidos nas lojas	Disco	Cada disco vendido pela Jonas' Discos
Cliente	Termo geral para todos os clientes das lojas	Freguês	Cada Cliente que compra na Jonas' Discos

Tabela 1 – Entidades.

Seguidamente, foi definido os atributos necessários a cada entidade.

2.2. Atributos

Nome da entidade	Atributos	Descrição	Data Type & Length
Loja	loja_id	Identifica a loja	Unsigned Integer
	morada		
	rua	Rua da loja	128 Varchar
	localidade	Localidade da loja	128 Varchar
	código postal	Código postal da loja	16 Varchar
	contabilidade		
	total faturado	Total faturado pela loja	Unsigned Float
	total gasto	Total gasto pela loja	Unsigned Float
Fornecedor	fornecedor_id	Identifica o fornecedor	Unsigned Integer
	nome	Nome do fornecedor	128 Varchar
	telefone	Numero de telefone	16 Varchar
	email	Email do fornecedor	128 Varchar
Disco Vinil	disco_id	Identifica o disco	Unsigned Integer
	nome	Nome do álbum	128 Varchar
	tipo	LP,EP ou SP	4 Enum
	preco de venda ao publico	Preço normal do disco	Unsiged Float
	genero	Género musical	128 Varchar
	rpm	33 ou 45 rotaçoes p/min	2 Integer
	duração	duração total do álbum	1 Time
	numero de faixas	numero de músicas	15 Integer
	artista	nome do artista	128 Varchar
Cliente	Cliente_id	Identifica o Cliente	Unsigned Integer
	Nome	Nome do Cliente	128 Varchar
	Telefone	Numero de telefone	16 Varchar
	Estatuto	Estatuto do cliente	3 Enum
	Discos comprados	Total discos comprados	15 Integer
	Morada		
	Rua	Rua onde o cliente vive	128 Varchar
	Localidade	Localidade do cliente	128 Varchar
	Codigo Postal	Codigo postal da casa	16 Varchar

Tabela 2 – Atributos.

Nota: nesta tabela, na coluna “Data Type & Length”, sempre que se escreve, por exemplo, “15 Integer” significa que o atributo é representado por um inteiro de 15 algarismos. Quando o atributo é do tipo String, o valor “Length” representa o número de caracteres. Já para o tipos enumerados, “Length” conta o número de elementos possíveis de representar com esse tipo.

O disco vinil é caracterizado pelo seu nome, pela sua duração total, o número de faixas, as *rotações por minuto* (RPM), o tipo (*long play*(LP), *single play* (SP) ou *extended play* (EP)), o *preço de venda ao público* (PVP). Para além disto, cada disco pode ter associado um ou mais

artistas e um ou mais gêneros. Para identificar a unicidade de cada disco vinil foi definido ainda um atributo disco_id pois pode haver discos com o mesmo nome.

Por outro lado, o fornecedor possui um nome, um email, um ou vários números de telefone e um fornecedor_id para identificar a unicidade de cada fornecedor. Cada cliente terá uma ficha com as suas informações, como um ou mais números de telefone, uma morada, que por sua vez terá uma rua, uma localidade e um código postal, um email, um estatuto (explicado no subcapítulo “Apresentação do Caso de Estudo”), o número total de discos comprados, o nome do cliente. Como já foi explicado anteriormente para as outras entidades, esta também tem um cliente_id.

Por último, a entidade loja é caracterizada por a contabilidade, sendo esta o total faturado, o total gasto e o lucro, e também uma morada com rua, localidade e código postal e também a loja_id.

2.3. Relacionamentos

Nome da entidade	Multiplicidade	Relacionamento	Multiplicidade	Nome da entidade
Loja	1	Tem	0..N	Disco de Vinil
Fornecedor	0..N	Abastece Abastece	1..N 1..N	Loja Disco de Vinil
Cliente	0..N	Compra Compra	1..N 1..N	Loja Disco de Vinil

Tabela 3 – Relacionamentos.

Nesta fase de recolha dos pré-requisitos foi necessário fazer os relacionamentos, isto é, como é que as entidades interagem e se relacionam umas com as outras, segundo a base de dados aqui proposta. Chegou-se à conclusão que ia ser necessário dois relacionamentos ternários e um binário.

O fornecedor vende o disco vinil à loja, por exemplo, o fornecedor A abastece a loja B com os discos x1, x2 e x3. Este relacionamento contém a informação relevante de cada venda, isto é, a quantidade de discos comprados, a data da venda, o custo, sendo este de cada disco e o total e um id, neste caso vende_id para identificar a venda.

O cliente compra um disco vinil na loja. Este processo de compra é caracterizado pela data, quantidade comprada, pelo preço, quer individual quer total e o id, compra_id de cada compra.

O relacionamento binário existe entre a loja e o disco vinil, de maneira a identificar os discos que cada loja contém bem como quantos discos iguais tem (com toda a informação igual, exceto o id).

3. Modelo conceptual

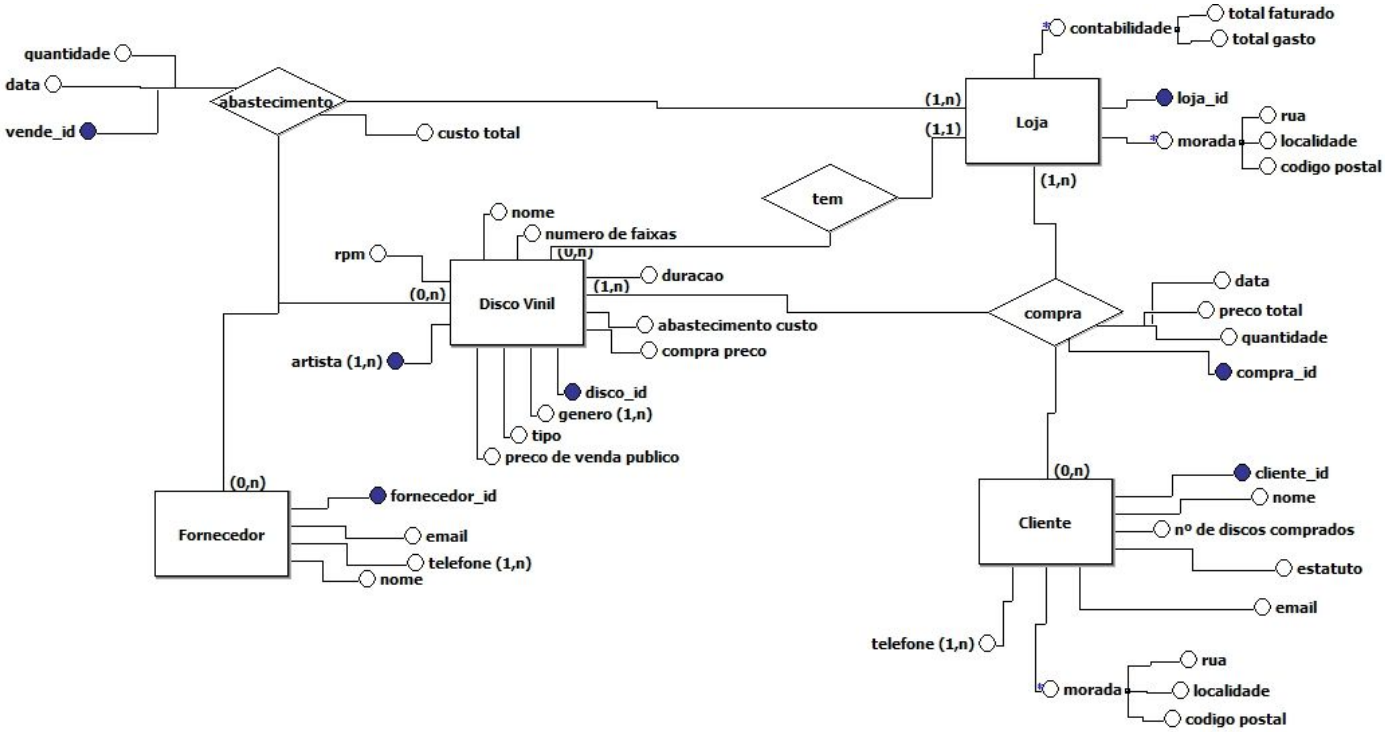

Figura 1 – Modelo conceptual.

4. Modelo Lógico

4.1. Passagem do modelo conceptual para o modelo lógico

Após analisado o modelo conceptual apresentado no capítulo anterior, foi possível efectuar uma tradução *quase* "direta" do mesmo para a sua representação lógica, que é apresentada de seguida:

Nota: as chaves primárias que são estrangeiras estão rodeadas por um rectângulo (não foi possível inserir trasejado duplo no editor de texto).

Fornecedor = { fornecedor_id, email, nome }
TelefoneFornecedor = { numero_fornecedor, fornecedor_id ↑FORNECEDOR }
Cliente = { cliente_id, nome, n discos comprados, estatuto, email, rua, localidade, cod. Postal }
TelefoneCliente = { numero_cliente, cliente_id ↑CLIENTE }
Disco = { disco_id, nome, nº faixas, duração, rpm, tipo, pvp, loja ↑DISCO }
ArtistaDisco = { artista, disco_id ↑DISCO }
GeneroDisco = { genero, disco_id ↑DISCO }
Loja = { loja_id, total facturado, total gasto, rua, localidade, cod. postal }
Compra = { compra_id, quantidade, data, custo total, cliente_id ↑CLIENTE, loja_id ↑LOJA }
DiscoCompra = { disco_id ↑DISCO, compra_id ↑COMPRA , preço }
Abastecimento = { abastecimento_id, quantidade, data, custo total, fornecedor_id ↑FORNECEDOR , loja ↑LOJA }
DiscoAbastecimento = { disco_id ↑DISCO , abastecimento_id ↑ABASTECIMENTO, preço }

Repare-se que este modelo ainda não se encontra normalizado, e que, durante a sua criação, os atributos compostos foram devidamente transformados: por exemplo, o atributo *morada* foi decomposto em: *rua*, *localidade* e *cod. postal*. Para além disso, os atributos multivalor foram representados em novas tabelas.

4.2. Normalização

4.2.1. Primeira Fórmula Normal

A primeira fórmula normal postula que, para uma determinada entidade, o domínio dos seus atributos apenas pode possuir valores atómicos, indivisíveis. Por outras palavras, um atributo não pode possuir conjuntos de valores. No caso aqui estudado, o modelo lógico anteriormente exposto já se encontra neste estado de normalização, pois não existem atributos representados por conjuntos (estes casos já se encontram separados em tabelas – por exemplo, a tabela *TelefoneCliente* atribui um número de telefone a um cliente, o que impede que a tabela *Cliente* possua representações de diversos números de telefone).

4.2.2. Segunda Fórmula Normal

A segunda fórmula normal garante que todos os atributos não-chave de uma tabela dependem apenas da chave da mesma (assumindo também que a tabela já se encontra na primeira fórmula normal). Para este caso de estudo, optou-se pelo estudo das possíveis dependências funcionais de cada uma das tabelas, com vista a eliminar tabelas e atributos redundantes.

Dependências funcionais

Para que o processo de normalização seja efetuado com sucesso, é importante analisar as relações entre atributos num relacionamento. No quadro seguinte estão apresentadas as principais dependências funcionais do projeto atual:

```
fornecedor_id     → email, nome
numero_fornecedor → fornecedor_id
cliente_id        → nome, ndicos comprados, estatuto, email, rua, localidade, cod. postal
numero_cliente    → cliente_id
disco_id          → nome, nº faixas, duração, RPM, tipo, pvp, loja_id
disco_id          → compra_id, preço
disco_id          → abastecimento_id, preço
loja_id           → total facturado, total gasto, rua, localidade, cod. postal
compra_id         → quantidade, data, preço total, cliente_id, loja_id
abastecimento_id  → quantidade, data, custo total, fornecedor_id, loja_id
```

Como é possível verificar à primeira vista, o identificador de um disco mapeia diferentes tabelas, e como foi visto, tal não pode acontecer. Para contornar este problema, procede-se simplesmente à junção dos atributos mapeados por *disco_id* numa única tabela. Deste modo, o modelo lógico poderia ser representado da seguinte forma:

```
Fornecedor         = { fornecedor_id, email, nome }
TelefoneFornecedor = { numero_fornecedor, fornecedor_id ↑FORNECEDOR }
```

Cliente = { cliente_id, nome, n discos comprados, estatuto, email, rua, localidade, cod. Postal }

TelefoneCliente = { numero_cliente, cliente_id $\uparrow^{CLIENTE}$ }

Disco = { disco_id, nome, nº faixas, duração, rpm, tipo, pvp , loja $\uparrow^{LOJA}$, compra_id $\uparrow^{COMPRA}$, compra_preço, abastecimento_id $\uparrow^{ABASTECIMENTO}$, abastecimento_preço }

ArtistaDisco = { artista, disco_id $\uparrow^{DISCO}$ }

GeneroDisco = { genero, disco_id $\uparrow^{DISCO}$ }

Loja = { loja_id, total facturado, total gasto, rua, localidade, cod. postal }

Compra = { compra_id, quantidade, data, custo total, cliente_id $\uparrow^{CLIENTE}$, loja_id $\uparrow^{LOJA}$ }

Abastecimento = { abastecimento_id, quantidade, data, custo total, fornecedor_id $\uparrow^{FORNECEDOR}$, loja $\uparrow^{LOJA}$ }

As tabelas *DiscoCompra* e *DiscoAbastecimento* foram eliminadas e os seus atributos pertencem agora à tabela *Disco,* não se verificando as lacunas identificadas anteriormente nas dependências funcionais.

4.2.3. Terceira Fórmula Normal

A terceira fórmula normal tem como objectivo garantir que, em todas as tabelas os seus atributos são apenas determinados pela chave das mesmas. Neste caso de estudo, após a aplicação da segunda fórmula normal, o sistema encontra-se automaticamente na terceira fórmula normal. O leitor mais desatento pode contrapôr, argumentando que na tabela *Disco*, os atributos *compra_preço* e *abastecimento_preço* são mapeados pelos atributos *compra_id* e *abastecimento_id*, respectivamente. Tal não é verídico, pois como ficou explicado na análise dos requisitos e conseguinte modelo conceptual, a entidade *Disco* refere-se a um disco fisicamente distinto e indiviual, e não a um conceito de álbum. Deste modo, será possível existirem na base de dados, *n* discos homónimos, sendo cada um representado numa linha diferente da tabela (possuindo entre si IDs distintos). Esta distinção permite uma melhor gestão de discos com o mesmo nome, mas cuja loja ou abastecimento, por exemplo são distintos.

4.2.4. Boyce–Codd, Quarta e Quintas Fórmulas Normais

O modelo aqui estudado, encontra-se actualmente (após a normalização com recurso à terceira fórmula) na fase final da normalização, pois já cumpre os requisitos definidos pelas quarta e quintas fórmulas normais, bem como as de Boyce-Codd. É possível então concluir que este modelo lógico será o final.

4.3. Transações e mapa correspondente

Com vista a ser possível imaginar um *overview* sobre as principais tarefas às quais a base de dados aqui projectada terá que responder, optou-se por selecionar algumas possíveis *queries* às quais a base de dados terá que dar resposta. Quando bem-sucedidas, o resultado destas transações deverá escrever um *commit* no *log* da base de dados, reportando que a

operação foi realizada com sucesso. Caso contrário, a base de dados deve estar preparada para efectuar um *rollback* que garanta a integridade da mesma.

As seguintes transações demonstram passo a passo o processo de resposta da base de dados a diversas *queries.* Nem todas possue um mapa associado, pois para algumas este é bastante simples. Para além disso, nem todas as transações se encontram aqui representadas, apenas as mais importantes e que melhor representam o funcionamento da base de dados.

A) Registo de um novo cliente:

1. Preenchimento da informação relativa ao novo cliente (nome, email, estatuto, etc...).
2. Inserção de uma nova linha contendo o registo da alínea 1. na tabela *Cliente.*
3. Na tabela *Telefone_Cliente* criar novos registos conforme os números de telefone relativos ao cliente acabado de registar.

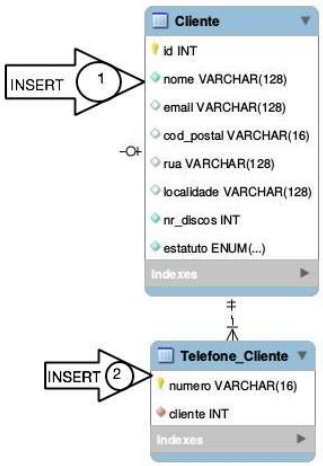

Figura 2.1 – Registo de um novo cliente

B) Realização de um abastecimento por um fornecedor a uma determinada loja:

1. Preenchimento da informação relativa ao novo abastecimento (data, custo total, identificador do fornecedor, identificador da loja, etc...).
2. Inserção de uma nova linha contendo o registo da alínea 1. na tabela *Abastecimento*.

C) Determinação da lista dos discos que foram vendidos numa determinada data numa determinada loja:

1. Leitura da tabela *Compra* e seleção daquelas cuja data e loja se pretende pesquisar.
2. Leitura da tabela Disco e seleção daqueles cujo identificador da compra se encontra presente na lista de compras obtida na alínea 2.

D) Registo de uma compra:

1. Inserção na tabela *Compra* de uma nova linha contendo as informações relativas a uma compra (data, cliente, loja, lista de discos comprados e respectivos preços).
2. Leitura da tabela *Disco* e selecção daqueles cujo identificador está presente na lista dos discos comprados.
3. Para cada um dos discos da lista gerada pela alínea 2., alterar o valor do campo *compra_preço*, e adicionar a nova chave relativa à compra.
4. Leitura da tabela *Cliente* e seleção do cliente que efectuou a compra.
5. Alteração no cliente gerado pela alínea 4. o número de discos comprados.
6. Seleção da loja na qual foi realizada a compra (i.e, leitura da tabela *Loja* e seleção daquela cujo identificador é igual ao da compra).
7. Alteração do valor do total facturado (adicionando o custo total da compra) na compra obtida pela alínea 6.

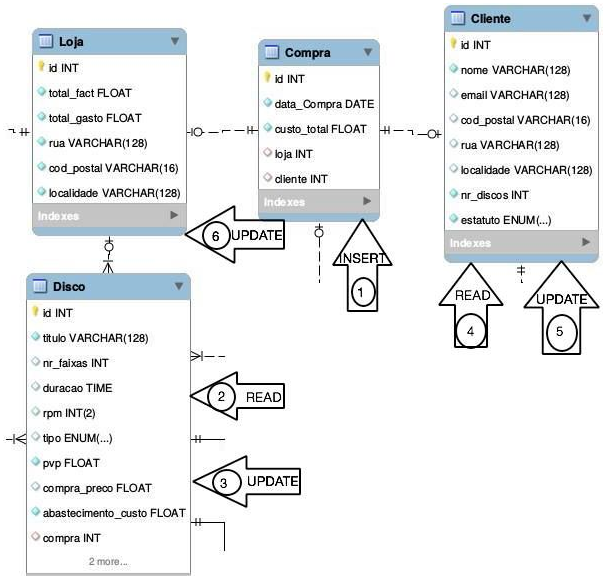

Figura 2.2 – Registo de uma compra

E) Determinação do número de discos vendidos de um determinado artista nas lojas de uma localidade:

1. Leitura da tabela *Loja*, selecionando aquelas cuja localidade se pretende pesquisar.
2. Leitura da tabela *Disco_Artista*, selecionando os discos correspondentes ao artista que se pretende pesquisar.
3. Leitura da tabela *Disco*, selecionando aqueles cujo identificador esteja presente na lista gerada pela alínea 2.
4. Contagem dos discos obtidos na etapa 3, cuja loja está presente na tabela resultante do passo 1.

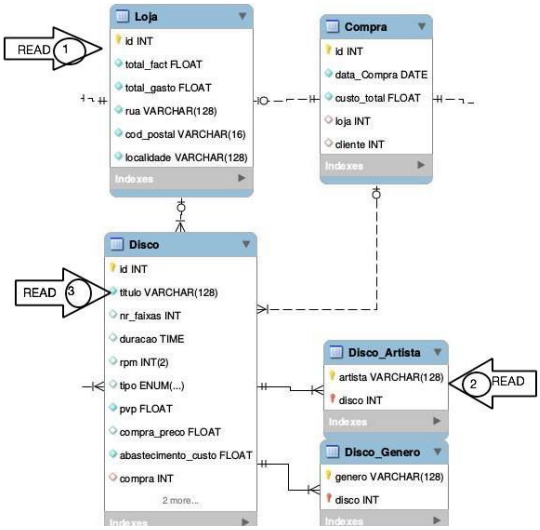

Figura 2.3 – Número de discos vendidos de um determinado artista nas lojas de uma localidade

F) Cálculo do lucro originado por um determinado cliente:

1. Leitura da tabela *Compra* e seleção daquelas que foram efectuadas pelo cliente desejado.
2. Leitura da tabela *Disco* e seleção daqueles cujo identificador da compra está contido no resultado do passo 1.
3. Dos discos obtidos no passo 2, calcular a soma das diferenças entre o preço de venda do disco e o preço de compra, para cada um dos discos.

4.4. Validação do esquema lógico

Após a inserção no MySQL Worbench, o modelo lógico aparenta-se desta forma:

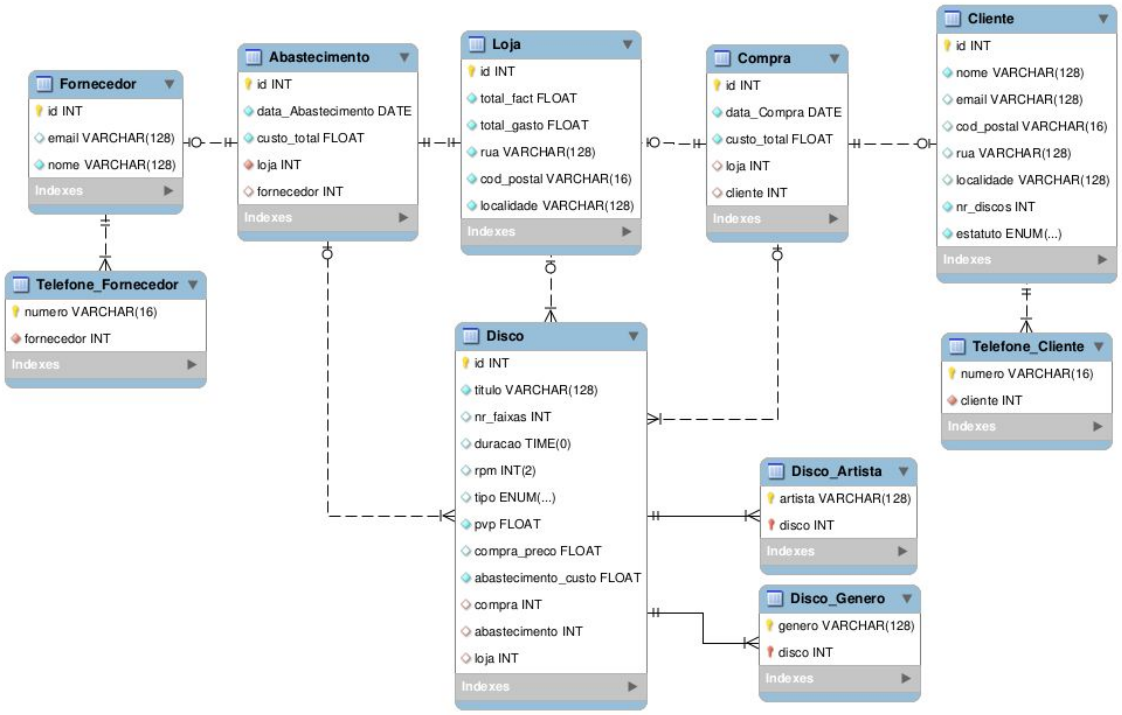

Figura 2.4 – Modelo lógico.

4.5. Tamanho inicial da BD

Com base no modelo lógico apresentado em cima, é possível definir agora um tamanho inicial da base de dados aqui proposta.

O primeiro passo para este cálculo é descobrir o tamanho de cada registo nas tabelas. De acordo com a especificação do MySQL, cada atributo do tipo INT ou FLOAT ocupa 4 bytes, DATE 3 bytes, ENUM 2 bytes (apenas existem 3 ou 4 opções para este tipo, logo apenas serão precisos 2 bytes para o representar) e por fim o VARCHAR onde cada char ocupa 1 byte.

A partir dos valores apresentados em cima, podemos retirar os seguintes cálculos para cada tabela:

- Fornecedor = 4 (id) + 128 (email) + 128 (email) = 260 bytes
- Cliente = 4 (id) + 128 (nome) + 128 (email) + 16 (cod_postal) + 128 (rua) + 128 (localidade) + 4 (nr_discos) + 2 (estatuto) = 538 bytes
- Disco = 4 (id) + 128 (titulo) + 4 (nr_faixas) + 3 (duração) + 4 (rpm) + 2 (tipo) + 4 (pvp) + 4 (compra_preco) + 4 (abastecimento_custo) + 4 (compra) + 4 (abastecimento) + 4 (loja) = 169 bytes
- Loja = 4 (id) + 4 (total_fact) + 4 (total_gasto) +128 (rua) + 16 (cod_postal) + 128 (localidade) = 284 bytes

- Abastecimento = 4 (id) + 3 (data_abastecimento) + 4 (custo_total) + 4 (loja) + 4 (fornecedor) = 19 bytes
- Compra = 4 (id) + 3 (data_Compra) + 4 (custo_total) + 4 (loja) + 4 (cliente) = 19 bytes
- Disco_Artista = 128 (artista) + 4 (disco) = 132 bytes
- Disco_Genero = 128 (genero) + 4 (disco) = 132 bytes
- Telefone_Cliente = 16 (numero) + 4 (cliente) = 20 bytes
- Telefone_Fornecedor = 16 (numero) + 4 (fornecedor) = 20 bytes

De seguida assume-se a quantidade dos seguintes registos iniciais na base de dados:

- Loja: 24
- Cliente: 32
- Telefone_Cliente (média de números de telefone por cliente) : 1
- Fornecedor: 3
- Telefone_Fornecedor (média de números de telefone por fornecedor) : 2
- Abastecimento: 5
- Compra: 15
- Disco: 22
- Disco_Artista (média de artistas por disco) : 1.5
- Disco_Genero (média de géneros por disco): 1

Tamanho total de cada entidade/relacionamento:

- Loja: 24*284 = 6816 bytes
- Cliente: 32*538 + 20*1*32 = 17856 bytes
- Fornecedor: 3*260 + 20*2*3 = 900 bytes
- Abastecimento: 5*19 = 95 bytes
- Compra: 15*19 = 285 bytes
- Disco: 22*169 + 132*1.5*22 + 132*1*22 = 10978 bytes

Somando todos os tamanhos obtém-se um total de 36930 bytes, ou seja, aproximadamente 37KB.

De maneira a analisar o crescimento da base de dados no futuro, com base nas médias anteriormente apresentadas, pode-se afirmar que se adicionar 1000 clientes, 2000 discos, e por conseguinte adicionarmos 1000 abastecimentos e 500 compras, os tamanhos de cada entidade/relacionamento ficarão:

- Loja: 6816 bytes
- Cliente: 1032*538 + 30*1*1032 = 586176 bytes
- Fornecedor: 900 bytes
- Abastecimento: 1005*19 = 19095 bytes
- Compra: 515*19 = 9785 bytes

- Disco: 2022*169 + 132*1.5*2022 + 132*1*2022 = 1008978 bytes

Fazendo assim um total de 1631750 bytes, ou seja, aproximadamente 1.6MB.

4.6. Revisão

Após ter o modelo lógico validado segundo os requisitos apresentados foi tempo de comunicar com os futuros utilizadores da base de dados. Estes futuros utilizadores são o *CEO* desta cadeia de lojas assim como os fincionários das mesmas. Foi então explicado o modelo lógico e a sua utilidade numa posterior implementação num sistema de base de dados. De seguida, apresentaram-se algumas das transações possíveis de efetuar com base naquelas que iriam ser mais utilizadas. O modelo lógico foi então aceite pelos utilizadores, uma vez que cumpria todos os requisitos apresentados inicialmente.

Assim sendo, pode-se afirmar que é possivel avançar para a próxima fase: a implementação do sistema de base de dados.

5. Projeto Físico

5.1. Implementação

Neste passo, foi gerada automaticamente a implementação do esquema físico a partir do modelo lógico existente no MySQL Workbench. O script de inicialização encontra-se aqui em baixo apresentado, e representa o domínio dos atributos de cada entidade e relacionamento e sua estrutura.

Algumas convenções foram escolhidas para garantir que a criação da base de dados era efectuada com bases sólidas, entre elas:

- Todas as chaves primárias representadas por números deverão ser do tipo UNSIGNED INT;
- Todos as as chaves primárias representadas por números deverão possuir as propriedade NOT NULL, e AUTO_INCREMENT (que permite a atribuição de chaves de forma automática);
- Os atributos do tipo VARCHAR deverão possuir 128 caracteres de comprimento para *strings* de tamanho médio/longo (tais como nomes, emails, por exemplo.) e 16 caracteres para *strings* de tamanho curto (tais como números de telefone e códigos de postalm por exemplo);
- O estatuto de um cliente será representado por um tipo enumerado, composto pelos elementos ‘NORMAL’, ‘GOLDEN’ e ‘PLATINUM’;
- O tipo de um disco será representado também por um tipo enumerado composto pelos elementos ‘LP’, ‘EP’, ‘Single’ e ‘Maxi’;

O *script* de inicialização da base de dados encontra-se aqui apresentado e a sua leitura é bastante *straightforward,* como se pode verificar:

```
-- MySQL Workbench Forward Engineering
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';

-- -----------------------------------------------------
-- Schema Jonas Discos
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `Jonas Discos` DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci ;
USE `Jonas Discos` ;
```

```
-- -----------------------------------------------------
-- Table `Jonas Discos`.`Fornecedor`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Jonas Discos`.`Fornecedor` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `email` VARCHAR(128) NULL,
  `nome` VARCHAR(128) NOT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `Jonas Discos`.`Telefone_Fornecedor`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Jonas Discos`.`Telefone_Fornecedor` (
  `numero` VARCHAR(16) NOT NULL,
  `fornecedor` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`numero`),
  INDEX `fk_telefone_fornecedor_fornecedor_idx` (`fornecedor` ASC),
  CONSTRAINT `fk_telefone_fornecedor_fornecedor`
    FOREIGN KEY (`fornecedor`)
    REFERENCES `Jonas Discos`.`Fornecedor` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `Jonas Discos`.`Cliente`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Jonas Discos`.`Cliente` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `nome` VARCHAR(128) NOT NULL,
  `email` VARCHAR(128) NULL,
  `cod_postal` VARCHAR(16) NULL,
  `rua` VARCHAR(128) NULL,
  `localidade` VARCHAR(128) NULL,
  `nr_discos` INT UNSIGNED NOT NULL,
  `estatuto` ENUM('NORMAL', 'GOLDEN', 'PLATINUM') NOT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `Jonas Discos`.`Telefone_Cliente`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Jonas Discos`.`Telefone_Cliente` (
  `numero` VARCHAR(16) NOT NULL,
  `cliente` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`numero`),
  INDEX `fk_telefone_cliente_cliente1_idx` (`cliente` ASC),
  CONSTRAINT `fk_telefone_cliente_cliente1`
    FOREIGN KEY (`cliente`)
    REFERENCES `Jonas Discos`.`Cliente` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `Jonas Discos`.`Loja`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Jonas Discos`.`Loja` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `total_fact` FLOAT UNSIGNED NOT NULL,
  `total_gasto` FLOAT UNSIGNED NOT NULL,
  `rua` VARCHAR(128) NOT NULL,
  `cod_postal` VARCHAR(16) NOT NULL,
  `localidade` VARCHAR(128) NOT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `Jonas Discos`.`Compra`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Jonas Discos`.`Compra` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `data_Compra` DATE NOT NULL,
  `custo_total` FLOAT NOT NULL,
  `loja` INT UNSIGNED NULL,
  `cliente` INT UNSIGNED NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_compra_loja1_idx` (`loja` ASC),
  INDEX `fk_compra_cliente1_idx` (`cliente` ASC),
  CONSTRAINT `fk_compra_loja1`
    FOREIGN KEY (`loja`)
    REFERENCES `Jonas Discos`.`Loja` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
```

```
  CONSTRAINT `fk_compra_cliente1`
    FOREIGN KEY (`cliente`)
    REFERENCES `Jonas Discos`.`Cliente` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `Jonas Discos`.`Abastecimento`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Jonas Discos`.`Abastecimento` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `data_Abastecimento` DATE NOT NULL,
  `custo_total` FLOAT UNSIGNED NOT NULL,
  `loja` INT UNSIGNED NOT NULL,
  `fornecedor` INT UNSIGNED NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_abastecimento_loja1_idx` (`loja` ASC),
  INDEX `fk_abastecimento_fornecedor1_idx` (`fornecedor` ASC),
  CONSTRAINT `fk_abastecimento_loja1`
    FOREIGN KEY (`loja`)
    REFERENCES `Jonas Discos`.`Loja` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_abastecimento_fornecedor1`
    FOREIGN KEY (`fornecedor`)
    REFERENCES `Jonas Discos`.`Fornecedor` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `Jonas Discos`.`Disco`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Jonas Discos`.`Disco` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `titulo` VARCHAR(128) NOT NULL,
  `nr_faixas` INT UNSIGNED NULL,
  `duracao` TIME NULL,
  `rpm` INT(2) NULL,
  `tipo` ENUM('EP', 'LP', 'SINGLE', 'MAXI') NULL,
  `pvp` FLOAT UNSIGNED NOT NULL,
  `compra_preco` FLOAT NULL,
  `abastecimento_custo` FLOAT UNSIGNED NOT NULL,
  `compra` INT UNSIGNED NULL,
  `abastecimento` INT UNSIGNED NULL,
  `loja` INT UNSIGNED NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_disco_compra1_idx` (`compra` ASC),
  INDEX `fk_disco_abastecimento1_idx` (`abastecimento` ASC),
  INDEX `fk_Disco_Loja1_idx` (`loja` ASC),
  CONSTRAINT `fk_disco_compra1`
    FOREIGN KEY (`compra`)
    REFERENCES `Jonas Discos`.`Compra` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_disco_abastecimento1`
    FOREIGN KEY (`abastecimento`)
    REFERENCES `Jonas Discos`.`Abastecimento` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Disco_Loja1`
    FOREIGN KEY (`loja`)
    REFERENCES `Jonas Discos`.`Loja` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `Jonas Discos`.`Disco_Artista`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Jonas Discos`.`Disco_Artista` (
  `artista` VARCHAR(128) NOT NULL,
  `disco` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`artista`, `disco`),
  INDEX `fk_disco_artista_disco1_idx` (`disco` ASC),
  CONSTRAINT `fk_disco_artista_disco1`
    FOREIGN KEY (`disco`)
    REFERENCES `Jonas Discos`.`Disco` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `Jonas Discos`.`Disco_Genero`
```

```
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Jonas Discos`.`Disco_Genero` (
  `genero` VARCHAR(128) NOT NULL,
  `disco` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`genero`, `disco`),
  INDEX `fk_disco_genero_disco1_idx` (`disco` ASC),
  CONSTRAINT `fk_disco_genero_disco1`
    FOREIGN KEY (`disco`)
    REFERENCES `Jonas Discos`.`Disco` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

SET SQL_MODE=@OLD_SQL_MODE;SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;SET
UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

5.2. Esquema físico

Com vista a efectuar diversos testes na base de dados, foi necessário realizar um povoamento da mesma com dados fictícios. Para tal, recorreu-se a um script SQL, cujos excertos se encontram aqui apresentados:

Nota: nas secções do código que contêm reticências (...) foi-lhes retirada uma parte substancial do mesmo (pois este é bastante extenso) e, como tal, podem surgir algumas incongruências (utilização de chaves ainda não existentes, por exemplo).

```
-- Indicação do esquema físico da base de dados
USE `Jonas Discos` ;

-- Permissão para fazer operações de remoção de dados
SET SQL_SAFE_UPDATES = 0;

-- Povoamento da tabela `Jonas Discos`.`Loja`INSERT INTO Loja
    (total_fact,total_gasto,rua,cod_postal,localidade)
    VALUES
    (0.0,0.0,'Rua dos Aliados','4400-000','Porto'),
    (100.0,200.0,'Rua da Ponte','2680-625','Lisboa'),
    (1500,300.0,'Avenida S.Joao','4710-100','Braga'),

                                                (...)

-- Povoamento da tabela `Joans Discos`.`Cliente`INSERT INTO Cliente
    (nome,email,cod_postal,rua,localidade,nr_discos,estatuto)
    VALUES
    ('João Faria','joaofaria@gmail.com','4232-123','Rua Americana','Porto',1,'NORMAL'),
    ('João Cunha','joaocunha@gmail.com','2323-323','Rua da Inglaterra','Lisboa',52,'GOLDEN'),
    ('António Alves','antonioalves@gmail.com','535-123','Rua Amarela','Porto',102,'PLATINUM'),

                                                (...)

-- Povoamento da tabela `Jonas Discos`.`Telefone_Cliente`INSERT INTO Telefone_Cliente
    (numero,cliente)
    VALUES
    (919191919,1),
    (918923764,1),
    (969863398,2),

                                                (...)

-- Povoamento da tabela `Jonas Discos`.`Fornecedor` INSERT INTO Fornecedor
    (email,nome)
    VALUES
    ('discos80@gmail.com','Discos 80s'),
    ('disconight@gmail.com','Disco Night!'),
    ('bertodiscos@gmail.com','Berto Discos');

-- Povoamento da tabela `Jonas Discos`.`Telefone_Fornecedor`INSERT INTO Telefone_Fornecedor
    (numero,fornecedor)
    VALUES
    (229062659,1),
    (229400939,1),
```

```
    (219049235,2);

-- Povoamento da tabela `Jonas Discos`.`Abastecimento`
    INSERT INTO Abastecimento
    (data_abastecimento,custo_total,loja,fornecedor)
    VALUES
    ('2015-01-01',2000,1,1),
    ('2014-01-18',3000,3,2),
    ('2015-01-19',4400,3,3),

                                                (...)

-- Povoamento da tabela `Jonas Discos`.`Abastecimento` INSERT INTO Compra
    (data_Compra,custo_total,loja,cliente)
    VALUES
    ('2014-03-02',120.0, 17, 30),
    ('2013-01-19',30.0, 17, 4),
    ('2012-12-20',25.0, 12, 8),
    ('2010-06-06',40.0, 17, 16),

                                                (...)

-- Povoamento da tabela  `Jonas Discos`.`Disco`INSERT INTO Disco
    (titulo,nr_faixas,duracao,rpm,tipo,pvp,compra_preco,abastecimento_custo,compra,abastecimento,loja)
    VALUES
    ('Another Brick in the Wall',20,'1:03:56',33,'LP',34.90, 30,20,1,1,17),
    ('Another Brick in the Wall',20,'1:03:56',33,'LP',34.90, null,20,null,1,2),
    ('Another Brick in the Wall',20,'1:03:56',33,'LP',34.90, null,20,null,1,3),
    ('The Dark Side of the Moon',20,'1:02:20',33,'LP',99.90, null,50,null,1,1),
    ('The Dark Side of the Moon',20,'1:02:20',33,'LP',99.90, 90,50,1,1,17),
    ('Obscured by Clouds',15,'0:59:56',33,'LP',34.90, null,20,null,1,1),
    ('Obscured by Clouds',15,'0:59:56',33,'LP',34.90, null,20,null,1,2),
    ('Obscured by Clouds',15,'0:59:56',33,'LP',34.90, null,20,null,1,1),
    ('Atom Mother Earth',10,'0:44:30',33,'EP',14.90, null,10,null,4,1),
    ('Atom Mother Earth',10,'0:44:30',33,'EP',14.90, null,10,null,4,1),
    ('Meddle',20,'01:20:00',33,'LP',34.90, null,20,null,1,1),
    ('Meddle',20,'01:20:00',33,'LP',34.90, null,20,null,1,1),
    ('Greatest Hits',18,'01:00:32',33,'LP',35, null,22,null,3,2),
    ('Greatest Hits',18,'01:00:32',33,'LP',35, null,22,null,3,2);

                                                (...)

-- Povoamento da tabela  `Jonas Discos`.`Disco_Artista`INSERT INTO Disco_Artista
    (artista,disco)
    VALUES
    ('Pink Floyd',1),
    ('Elvis Presley',17),
    ('Pink Floyd',17),
    ('Queen',21),
    ('Queen',22);

                                                (...)

-- Povoamento da tabela  `Jonas Discos`.`Disco_Genero`INSERT INTO Disco_Genero
    (genero,disco)
    VALUES
    ('Rock Classico',1),
    ('Rock Classico',5),
    ('Hard Rock',6),
    ('Rock Progressivo',12),
    ('Rock Progressivo',13),
    ('Blues',14),
    ('Jazz',19),

                                                (...)

-- Inibição das operações de remoção de dados
SET SQL_SAFE_UPDATES = 1;
```

5.3. Análise de transações

Tal como foi visto no capítulo anterior, as transações são uma componente bastante importante de qualquer base de dados. Com elas é possível efectuar diversas manipulações na base de dados, mantendo a consistência da mesma. No caso aqui estudado, como muitas das *queries* executadas são algo complexas e serão utilizadas frequentemente, foi feito um esforço para favorecer a reutilização do código. Para tal recorreu-se à utilização de *procedures*, que se

encontram aqui em baixo apresentados. Todos eles, depois de testados, foram armazenados na própria base de dados (tornando-se *stored procedures*) com vista a permitir uma maior facilidade de acesso aos mesmos.

```
-- Criar um novo cliente
-- Utilização:
--   CALL cria_cliente ("Rui Fernandes", "rfernandes@iol.pt", "4760-213", "Rua da Bombaça, 45", "Ribeirão", 2,
'NORMAL', "912345678");
DELIMITER //
CREATE PROCEDURE `cria_cliente` (nome VARCHAR(128), email VARCHAR(128), cod_postal VARCHAR(128),
                                 rua VARCHAR(128), localidade VARCHAR(128), nr_discos INT,
                                 estatuto ENUM('NORMAL', 'GOLDEN', 'PLATINUM'), numero VARCHAR(16))
BEGIN
INSERT INTO Cliente
    (nome, email, cod_postal, rua, localidade, nr_discos, estatuto)
    VALUES
    (nome, email, cod_postal, rua, localidade, nr_discos, estatuto);
INSERT INTO Telefone_Cliente
    (numero, cliente)
    VALUES
    (numero, LAST_INSERT_ID());
END //

-- Calculo do lucro originado por um determinado cliente
-- Utilização:
--   CALL lucro_cliente (32);
DELIMITER //
CREATE PROCEDURE `lucro_cliente` (cliente INT)
BEGIN
    SELECT SUM(compra_preco - abastecimento_custo) FROM Disco INNER JOIN Compra
    ON Disco.compra = Compra.id
    WHERE Compra.cliente = cliente;
END //

-- Calculo do lucro origindado de uma loja
-- Utilização:
--   CALL lucro_loja (2);
DELIMITER //
CREATE PROCEDURE `lucro_loja` (loja INT)
BEGIN
    SELECT total_fact - total_gasto AS Lucro FROM Loja
    WHERE Loja.id = loja;
END //

-- Calculo do lucro originado por cada loja
DELIMITER //
CREATE PROCEDURE `lucro` ()
BEGIN
    SELECT Loja.id AS Loja, total_fact - total_gasto AS Lucro FROM Loja;
END //

-- Insere vários discos na base de dados (o número de discos a inserir é definido pela quantidade) que
partilham as mesmas características (nome, artista, género, etc...)
-- Utilização:
--   CALL insere_disco ('O Monstro Precisa de Amigos', 'Ornatos Violeta', 'Rock', 13, '1:03:56', 33, 'LP',
23.25 ,10.2, 1, 1, 10);
DELIMITER //
CREATE PROCEDURE `insere_disco` (titulo VARCHAR(128), artista VARCHAR(128), genero VARCHAR(128),
                                 nr_faixas INT, duracao TIME, rpm INT(2),
                                 tipo ENUM('EP', 'LP', 'SINGLE', 'MAXI'), pvp FLOAT,
                                 abastecimento_custo FLOAT, abastecimento INT, loja INT, quantidade INT)
BEGIN
    DECLARE i INT DEFAULT 0;
    DECLARE disco_id INT;

    WHILE i < quantidade DO
        INSERT INTO Disco (titulo, nr_faixas, duracao, rpm, tipo, pvp, compra_preco, abastecimento_custo,
compra, abastecimento, loja)
        VALUES (titulo, nr_faixas, duracao, rpm, tipo, pvp, NULL, abastecimento_custo, NULL, abastecimento,
loja);

        SET disco_id = LAST_INSERT_ID();

        INSERT INTO Disco_Artista (artista, disco)
        VALUES (artista, disco_id);

        INSERT INTO Disco_Genero (genero, disco)
        VALUES (genero, disco_id);

        SET i = i + 1;
    END WHILE;
```

```
END //

-- Registar uma compra
-- Utilização
--   CALL resgista_compra('2013-01-19', 30.23, 1, 1)
DELIMITER //
CREATE PROCEDURE `regista_compra` (data_Compra DATE, custo_total FLOAT, nr_discos INT, loja INT, cliente INT)
BEGIN
    INSERT INTO Compra (data_Compra, custo_total, loja, cliente)
    VALUES (data_Compra, custo_total, loja, cliente);

    UPDATE Loja
    SET Loja.total_fact = Loja.total_fact + custo_total
    WHERE Loja.id = loja;

    UPDATE Cliente
    SET Cliente.nr_discos = Cliente.nr_discos + nr_discos,
        Cliente.estatuto  = IF (Cliente.nr_discos > 99, 'PLATINUM', IF (Cliente.nr_discos > 49, 'GOLDEN',
'NORMAL'))
    WHERE Cliente.id = cliente;
END //

-- Lista dos discos que foram vendidos numa determinada data numa determinada loja
-- Utilização:
--   CALL compra_por_data_numa_loja ('2014-03-02', 17);
DELIMITER //
CREATE PROCEDURE `compra_por_data_numa_loja` (data_Compra DATE, loja INT)
BEGIN
    SELECT D.* FROM Disco as D INNER JOIN Loja AS L
    ON D.loja = L.id
    INNER JOIN Compra AS C
    ON D.compra = C.id
    WHERE C.data_compra = data_compra;
END //

-- Determinação do número de discos vendidos de um determinado artista nas lojas de uma localidade
-- Utilização:
--   CALL ndiscos_artista_localidade ('Pink Floyd', 'Gaia');
DELIMITER //
CREATE PROCEDURE `ndiscos_artista_localidade` (artista VARCHAR(128), localidade VARCHAR(128))
BEGIN
    SELECT COUNT(D.id) FROM Disco AS D INNER JOIN Disco_Artista AS DA
    ON D.id = DA.disco INNER JOIN Compra AS C
    ON D.compra = C.id INNER JOIN Loja AS L
    ON C.loja = L.id
    WHERE L.localidade = localidade AND DA.artista = artista;
END //
```

Após a criação deste *procedures*, é possível efectuar algumas das transações definidas na parte 4 de uma forma bastante mais simplificada, pois a manipulação directa com a base de dados é efectuada pelo *procedure*:

```
USE `Jonas Discos`;

-- Criar um novo cliente
START TRANSACTION;
CALL cria_cliente (‘Rui Fernandes’, ‘rfernandes@iol.pt’, ‘4760-213’, ‘Rua da Bombaça, 45’, ‘Ribeirão’, 2,
'NORMAL', '912345678');
COMMIT;

-- Calculo do lucro originado por um determinado cliente
START TRANSACTION;
CALL lucro_cliente (26);
COMMIT;

-- Calculo do lucro origindado de uma loja
START TRANSACTION;
CALL lucro_loja (2);
COMMIT;

-- Calculo do lucro originado por cada loja
START TRANSACTION;
CALL lucro ();
COMMIT;

-- Insere vários discos na base de dados (o número de discos a inserir é definido pela quantidade) que
partilham as mesmas características (nome, artista, género, etc...)
START TRANSACTION;
CALL insere_disco ('CAO!', 'Ornatos Violeta', 'Funk', 13, '1:03:56', 33, 'LP', 23.25 ,10.2, 1, 1, 13);
COMMIT;
```

```sql
-- Registar uma compra
START TRANSACtION;
CALL resgista_compra('2013-01-19', 30.23, 1, 1);
COMMIT;

-- Lista dos discos que foram vendidos numa determinada data numa determinada loja
START TRANSACTION;
CALL compra_por_data_numa_loja ('2014-03-02', 17);
COMMIT;

-- Determinação do número de discos vendidos de um determinado artista nas lojas de uma localidade
START TRANSACTION;
CALL ndiscos_artista_localidade ('Pink Floyd', 'Gaia');
COMMIT;
```

5.4. Triggers

No projecto em causa não foi dada muita relevância à utilização de *triggers*, pois a maioria das interações do utilizador com a base de dados seriam efectuadas através dos *procedures*. No entanto, a criação de *triggers* é possível e em alguns casos podem revelar-se bastante úteis. O exemplo aqui apresentado, demonstra como um trigger pode garantir a segurança da remoção de um fornecedor (removendo os seus números de telefone também - que se encontram numa outra tabela, como é sabido).

```sql
-- Ao apagar um fornecedor, apagar também os seus números de telefone
DELIMITER //
CREATE TRIGGER apaga_fornecedor AFTER DELETE ON `Fornecedor`
FOR EACH ROW
BEGIN
    DELETE FROM Telefone_Fornecedor
    WHERE OLD.id = Telefone_Fornecedor.fornecedor;

    DELETE FROM Abastecimento
    WHERE OLD.id = Abastecimento.fornecedor;
END //
```

saas

5.5. Espaço em disco

A partir do tamanho das tabelas, e do número de entidades que povoaram a base de dados, é possível estimar o espaço em disco necessário pazra o armazenamento da base de dados. É possível automatizar a contagem do tamanho da base de dados recorrendo a um script SQL:

```sql
SELECT table_schema 'DB Name', SUM(data_length + index_length) / 1024 'DB Size in KB'
FROM information_schema.tables
WHERE table_schema = 'Jonas Discos';
```

Executando o script, obteve-se como resultado 0.336 MB, que representa o espaço em disco ocupado pela base de dados actualmente.

Nota: o script aqui apresentado não foi escrito pelos autores deste projecto, mas sim encontrado no endereço: http://www.mkyong.com/mysql/how-to-calculate-the-mysql-database-size/

5.6. Vistas dos utilizadores

Como já foi mencionado nas partes anteriores, a criação de *stored procedures* e *views* facilitou bastante a execução de determinadas tarefas. O acesso aos dados passa então a ser "encapsulado", realizando-se prefencialmente através da invocação das rotinas anteriormente definidas. Isto permite ofuscar dos utilizadores a implementação das diversas queries, e garantir que a manipulação dos dados é efectuada de forma previsível e controlada. Os utilizadores, deste modo, interagem com a base de dados de uma forma mais acessível, sem necessitarem de conhecer os detalhes da mesma.

Exemplificando, aquando a utilização do procedimento "regista_compra", o utilizador não necessita de actualizar manualmente o total facturado de uma loja, nem o número de discos comprados por esse cliente, pelo que estes detalhes ficaram escondidos da vista do utilizador.

É também possível armazenar o resultado de um *procedure* numa tabela, para facilitar uma futura consulta. Para tal basta criar uma *view* em SQL, como a que aqui se apresenta (que armazena numa tabela de nome *lucros_ate_hoje* o valor do lucro de cada loja no momento actual):

```
use `Jonas Discos`;

CREATE VIEW lucros_ate_hoje AS
SELECT Loja.id, total_fact - total_gasto FROM Loja;

SELECT * FROM lucros_ate_agora;
```

6. Conclusões e Trabalho Futuro

Neste trabalho decidiu-se elaborar uma base de dados para uma loja, ou conjunto de lojas, dedicada à venda de discos vinil. Para facilitar o dia-a-dia nestas empresas, este projeto concentra-se no fluxo do produto para as diversas lojas bem como a sua venda e compra, em guardar os contactos dos fornecedores, os produtos que estes venderam e custos associados, e diversa informação sobre os clientes e as compras que estes fizerem. Este foco permite uma fácil visualização das transações, necessidades e viabilidade de uma loja ou disco, permite manter a fidelização de vários clientes através de estatutos e descontos associados, manter registos dos discos que cada loja tem e das compras e vendas, também separadas por loja.

No entanto, esta base de dados carece em informações especificas de cada loja, por exemplo, sobre os seus empregados ou sobre a localização de discos específicos (dificultando o manuseamento de lojas maiores), e em flexibilidade pois só está equipada para a venda de um tipo de produto.

Caso seja viável, a expansão deste trabalho para incluir uma maior diversidade de produtos e a possibilidade de tratar a informação contabilística necessária para uma empresa tornaria esse projeto mais genérico e abrangente e, como tal, mais útil para várias empresas.
Em suma, apesar de ser relativamente restrita, esta base de dados é capaz de manter toda a informação essencial para o funcionamento da empresa em questão.

É importante referir que na segunda fase deste trabalho foi necessário efectuar algumas alterações no trabalho realizado na primeira fase. Nomeadamente: uma pequena alteração no modelo conceptual e também uma melhoria na contextualização apresentada na secção 1.1.

Por fim, pode-se concluir que, apesar das dificuldades sentidas ao longo da realização deste projecto, este foi realizado com sucesso. Conseguindo assim um projecto viável e robusto. Também é correto afirmar que este trabalho serviu para uma boa consolidação de todos os conhecimentos adquiridos ao longo deste semestre quer nas aulas teóricas como também nas aulas práticas.

Bibliografia

Connoly, T. Begg, C., Database Systems, A Pratical Approach to Design, Implementation, and Management.

Lista de Siglas e Acrónimos

LP *long play*

SP *single play*

EP *extended play*

RPM *rotações por minuto*

PVP *preço de venda ao público*