Universidade do Minho

Escola de Engenharia

Mestrado Integrado em Engenharia Informática

Unidade Curricular de

Bases de Dados

Ano Letivo de 2015/2016

Sistema de Gestão de Bases de Dados de um Hipermercado

A67646 - Bruno Barbosa

A67738 - Gil Gonçalves

A67709 - Sandra Ferreira

A67707 - Tiago Cunha

Janeiro, 2016

Data de Recepção	
Responsável	
Avaliação	
Observações	

Sistema de Gestão de Bases de Dados de um Hipermercado

A67646 - Bruno Barbosa

A67738 - Gil Gonçalves

A67709 - Sandra Ferreira

A67707 - Tiago Cunha

Janeiro, 2016

Resumo

O principal objetivo deste trabalho é a elaboração de um sistema de base de dados, capaz de gerir a informação de um supermercado, fornecendo suporte à manutenção, gestão e consulta da informação do mesmo.
Neste relatório são descritos todos os passos efetuados para a construção deste sistema,

Área de Aplicação: Análise de requisitos, desenho e arquitetura de Sistemas de Bases de Dados, Bases de Dados.
Palavras-Chave: Requisitos, relacionamentos, esquema concetual, base de dados, MySQL, funcionário, função, secção, produto, cliente, fornecedor, modelo lógico, modelo físico, relações, transações, *triggers*, normalização, integridade, vistas.

Índice

Índice de Figuras

Índice de Tabelas

1. Introdução

1.1. Contextualização

Um hipermercado corresponde a uma grande superfície de comércio tradicional baseado num sistema de autosserviço que oferece uma vasta gama de alimentos e produtos variados.
A definição de autosserviço surge na medida em que os clientes fazem as suas compras sem precisar do auxílio de um assistente, à exceção de algumas áreas, como por exemplo peixaria ou charcutaria. Normalmente, os hipermercados possuem áreas iguais ou superiores a 2000 metros quadrados, característica muito própria, daí a designação de "híper". Como é óbvio, dentro do hipermercado existem diversos sectores de forma a facilitar a procura dos produtos. Caso contrário, estaríamos perante um enorme caos e não iríamos conseguir fazer compras tão eficazmente.
Os hipermercados surgiram numa era onde as pessoas começaram a ser cada vez mais consumistas e o seu aparecimento foi crucial visto que as pequenas mercearias, que vemos frequentemente, não iram ser capazes de satisfazer as necessidades de tantos consumidores.

1.2. Apresentação do Caso de Estudo

Os hipermercados são infraestruturas cada vez mais requisitadas pela população dadas as condições que estes reúnem. Estes espaços conseguem concentrar uma grande quantidade e diversidade de produtos, das mais variadas marcas e preços.
Por detrás dos hipermercados está implementado um grande processo de logística onde o rigor e a precisão são essenciais à organização de todos os elementos nele interveniente.
Essa precisão engloba a gestão das compras de produtos aos fornecedores, das vendas dos produtos aos clientes, passando pela manutenção do espaço, salários dos funcionários, entre outros fatores.
Portanto, a necessidade de existir um sistema que possa ajudar o gestor do hipermercado a gerir o mesmo da forma mais rentável possível, é uma questão que deve ter surgido logo após do aparecimento dos primeiros hipermercados.
Contudo, hoje em dia esses sistemas não são novidade mas desempenham um papel fundamental no que diz respeito à otimização dos lucros de um estabelecimento como o que acabamos de referir.

1.3. Motivação e Objetivos

A maior parte da motivação para termos escolhido este tema para o nosso projeto da unidade curricular de Base de Dados advém do facto de um hipermercado ser um espaço muito requisitado quando precisamos de abastecer a nossa casa. Na verdade, estes estabelecimentos têm uma enorme diversidade de produtos, desde alimentos, equipamentos tecnológicos, vestuário, etc., o que os torna bastantes úteis. Todavia, é um grande desafio garantir a sua sustentabilidade e isso também despertou em nós um interesse especial porque reconhecemos que não é uma tarefa fácil. Atualmente vivemos num mundo bastante informatizado: onde quer que vamos, a informática está sempre presente. O caso dos hipermercados não é exceção. Seria totalmente impensável, nos dias de hoje, gerir um hipermercado sem recorrer a uma ferramenta informática que ajude e suporte no controlo de todas as vertentes do mesmo.
Nesta medida, os nossos objetivos visam desenvolver uma aplicação (Sistema de Bases de Dados) tal que, para além de poder informar o gestor do hipermercado sobre as quantidades de cada produto no armazém ou nas prateleiras, possa também alertá-lo atempadamente de forma a evitar quebras ou ruturas de *stock*, bem como eventuais excessos de mercadoria e assim tirar o lucro máximo do negócio.

1.4. Estrutura do Relatório

Este relatório descreve uma metodologia de criação de um SGBD. Essa metodologia é esta presente no livro *Database Systems – A Pratical Approach to Design, Implementation and Managment*, 4ª Edição (*Conolly* e *Begg*, 2005).
O primeiro capítulo contém a exposição do problema através da sua contextualização. Segue-se, depois, a apresentação do caso de estudo e a motivação/objetivos que nos levaram a escolher este tema para o nosso trabalho.
No segundo capítulo introduzimos a fase inicial da conceção e do *design* da base de dados. Neste capítulo encontra-se o não só o levantamento de requisitos mas também a parte referente à modulação conceptual. Este capítulo subdivide-se deste modo, na identificação das entidades, relacionamentos entre entidades, respetivos atributos e identificadores, passando ainda pela atribuição do domínio dos atributos.
O terceiro capítulo é dedicado à modelação lógica e inclui todas fases a ela associada. Tais como a derivação do modelo concetual para lógico, normalização, transações do utilizador, entre outras.
No quarto capítulo temos a parte referente à modelação física. Nela efetuamos a transição do modelo lógico para o modelo físico bem como a definição das vistas de cada utilizador.
No quinto capítulo expomos as ferramentas utilizadas para cada vertente deste trabalho desde o relatório até às ferramentas de desenvolvimento de um SGBD.
No sexto capítulo apresentamos as nossas conclusões depois de finalizarmos o trabalho, assim como o trabalho futuro que pode ainda ser realizado.

2. Modelação Concetual

2.1. Levantamento de requisitos

Um supermercado é uma infraestrutura que necessita de funcionários para o seu funcionamento. Cada funcionário tem uma ou várias funções, podendo para isso trabalhar numa ou em várias secções, num dado número de horas de dado dia. Para gerir os funcionários existe um administrador, que é igualmente um funcionário mas com esta função adicional. Para além disso existem também funcionários aos quais lhes foram incutidas a responsabilidade da gestão de uma ou várias secções.

Em cada secção do supermercado existe uma determinada quantidade de diversos produtos. Uma entidade imprescindível são os fornecedores que, num dado dia, fornecem uma determinada quantidade de um produto a um custo resultante de uma relação preço-quantidade. Por fim, depois de todos os requisitos cumpridos, os clientes efetuam a compra destes produtos. Com este sistema de base de dados será possível consultar o histórico de compras dos clientes, assim como, consultar todos os abastecimentos feitos pelo supermercado.

Após a análise dos requisitos aqui mencionados, foi possível nomear e concluir que as entidades em baixo apresentadas são as fundamentais para este projeto:

- Funcionário
- Função
- Secção
- Produto
- Fornecedor
- Cliente

2.2. Descrição dos relacionamentos entre entidades

Depois de já termos identificado entidades podemos, de imediato, seguir para a identificação dos relacionamentos existentes entre as mesmas.

- A entidade funcionário relaciona-se consigo mesma na medida em que existe um funcionário responsável pela gestão dos restantes funcionários. Para além disso, os funcionários têm, como a sua designação indica, uma ou várias funções e trabalham em diversas secções (de acordo com as necessidades do hipermercado) onde, para cada secção, há um funcionário responsável.

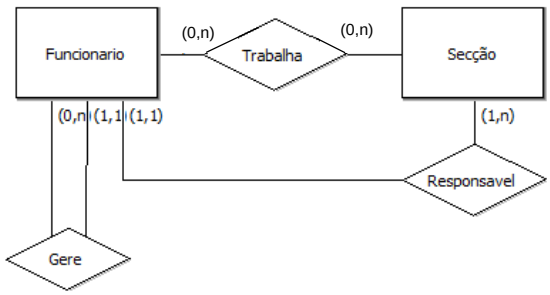


Figura 1. Relação entre Funcionário e Secção

- A entidade função relaciona-se unicamente com a entidade funcionário na perspetiva de que um funcionário está associado a uma ou várias funções.

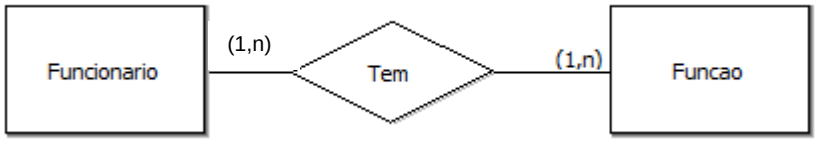


Figura 2. Relação entre Funcionário e Função

- A entidade secção está relacionada com as entidades funcionário e produto. Por um lado, como já foi referido, não só vários funcionários trabalham em várias secções como também existe um funcionário responsável por uma ou várias secções. Por outro lado, uma secção contém vários produtos.

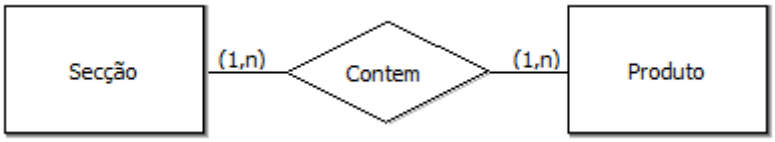


Figura 3. Relação entre Secção e Produto

- A entidade produto é uma entidade chave neste modelo. É ela que relaciona e interliga, basicamente, todo o modelo conceptual. Primeiramente, vários produtos estão contidos em várias secções. Segundo, vários produtos podem ser comprados por vários clientes assim como vários produtos são fornecidos por vários fornecedores.

- A entidade cliente participa neste universo relacionando-se apenas com a entidade produto. Isto é, vários clientes podem comprar vários produtos.

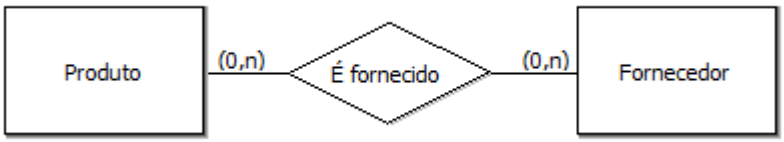


Figura 4. Relação entre Produto e Fornecedor

- Por último, vários fornecedores fornecem vários produtos. Este relacionamento justifica-se pelo facto de fornecedores diferentes poderem fornecer o mesmo produto.

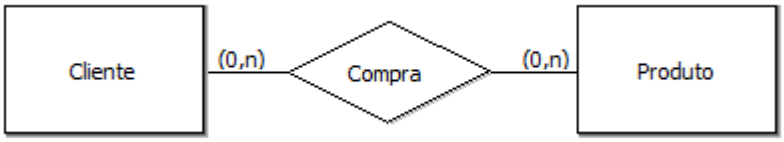


Figura 5. Relação entre Cliente e Produto

Nos anexos 1 encontra-se a tabela 1 respetiva ao dicionário de dados dos relacionamentos.

2.3. Descrição das entidades

Nesta secção procedemos à descrição das entidades. Nos anexos 1 está exposta a tabela 2 que é referente ao dicionário de dados das entidades.
Assim sendo, eis uma breve explicação de cada entidade.

- A entidade funcionário serve para representar todos os funcionários do sistema.
- A entidade função serve simplesmente para representar as várias funções que os funcionários podem assumir.
- A entidade secção representa as várias secções onde os produtos estão armazenados na área de venda.
- A entidade produto qualifica todos os produtos existentes e os aspetos a eles relevantes.
- A entidade cliente descreve as características adjacentes aos clientes do hipermercado.
- A entidade fornecedor especializa as qualidades alusivas aos dados dos múltiplos fornecedores.

2.4. Descrição dos atributos das entidades

Este segmento foca-se na atribuição e na descrição dos atributos. Uma vez que vamos abordar os atributos das entidades, aproveitamos também já para atribuir os atributos identificadores.
Desta forma, a entidade funcionário é caracterizada pelos seguintes atributos:

- Nome
- NIF – Número de identificação fiscal que será o seu identificador único
- Correio eletrónico – Email para eventuais avisos e notificações
- Salario – Corresponde ao salário mensal
- Contacto (multi-valor) – Diversos contactos telefónicos
- Endereço (composto) – Corresponde à morada estendida
- Data de Nascimento

Seguimos para a entidade função que é caracterizada por dois atributos:

- ID – Identificador único da função
- Descrição – Uma breve descrição da função

A próxima entidade a ser qualificada é a secção que tem como atributos:

- ID – Identificador único da secção
- Nome – Nome da secção

Segue-se a entidade cliente com os atributos:

- Nome
- NIF – Número de identificação fiscal e identificador único
- Data de nascimento
- Contacto telefónico (simples)
- Correio eletrónico – Email
- Endereço (composto) – Corresponde à morada estendida

A entidade produto possui os atributos que se seguem:

- ID – Identificador único (num contexto real, poderia ser o código de barras)
- Nome
- Preço – Preço de venda
- *Stock* – Quantidade de produto (exposta e armazenada)

Finalmente, resta-nos a entidade fornecedor que é especificada pelos atributos:

- NIF – Número de identificação fiscal e identificador único
- Nome
- Contacto (multi-valor) – Múltiplos contactos telefónicos
- Endereço (composto) – Morada estendida do estabelecimento fornecedor

Veja, nos anexos 1, a tabela 3 relativa ao dicionário de dados dos atributos das entidades.

2.5. Modelo conceptual

Dadas todas estas considerações iniciais, apresentamos agora, o modelo concetual. Como é óbvio, estas considerações foram determinantes na conceção do mesmo, pelo que sem elas, não seria possível estruturar-mos bem o modelo conceptual de modo a que fosse, no mínimo, consistente e robusto. A imagem abaixo trata-se da versão mais recente, portanto, antes desta versão surgiram outras que ao longo do tempo, foram devidamente refinadas em conjunto com os esclarecimentos do professor. Nos anexos 2 encontram-se as versões mais antigas que mostram como o esquema concetual foi evoluindo.

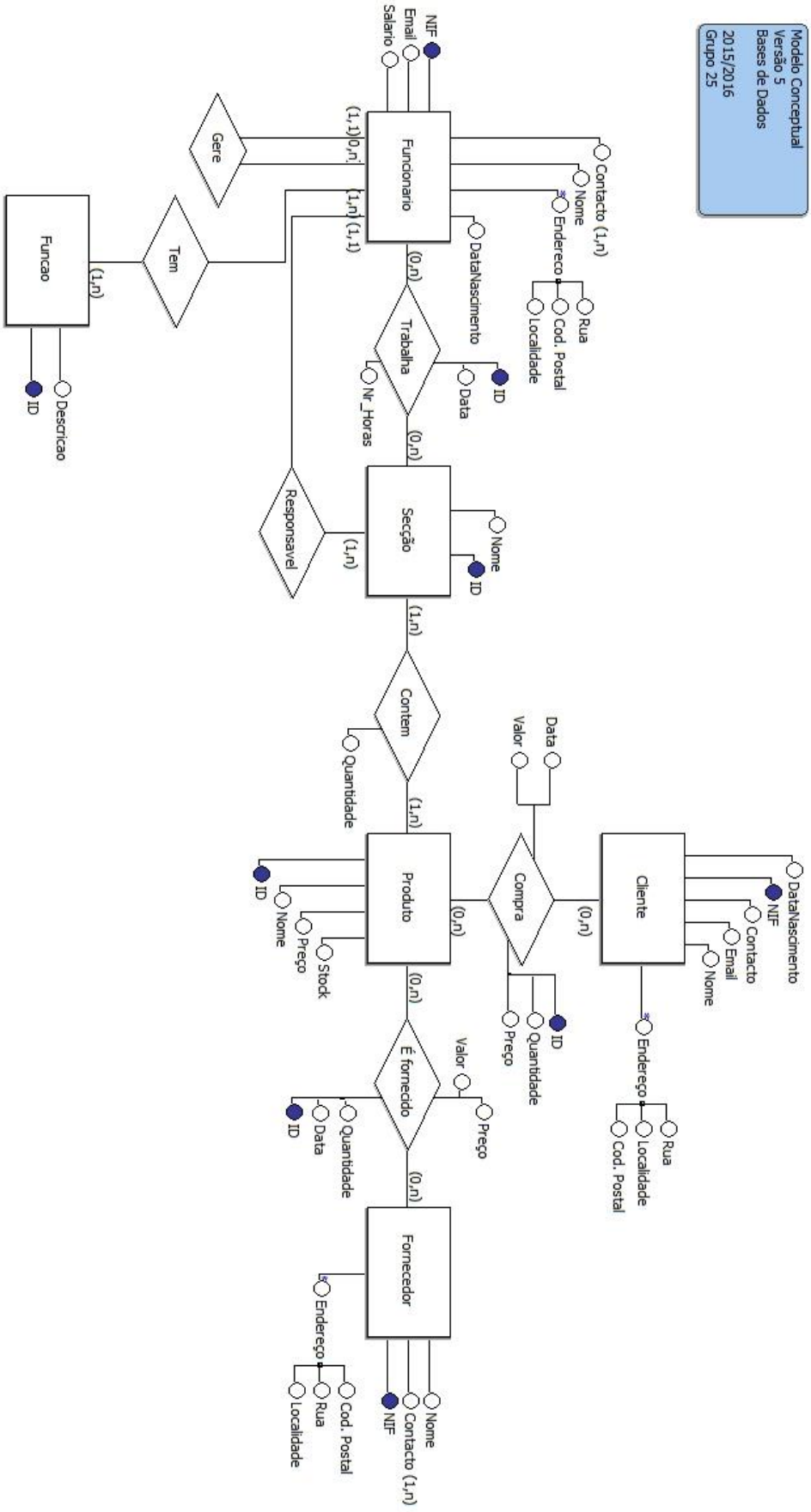


Figura 6. Versão mais recente do modelo conceptual

3. Modelo Lógico

No seguimento deste processo iterativo de construção de uma base de dados, geramos agora o nosso modelo lógico de acordo com o modelo conceptual anteriormente apresentado. Nesta etapa surgem as tabelas correspondentes às diversas entidades assim como dos vários tipos de relacionamentos. Através da conclusão desta fase, já teremos um modelo representativo dos requisitos do sistema.

O modelo lógico encontra-se apresentado imediatamente abaixo.

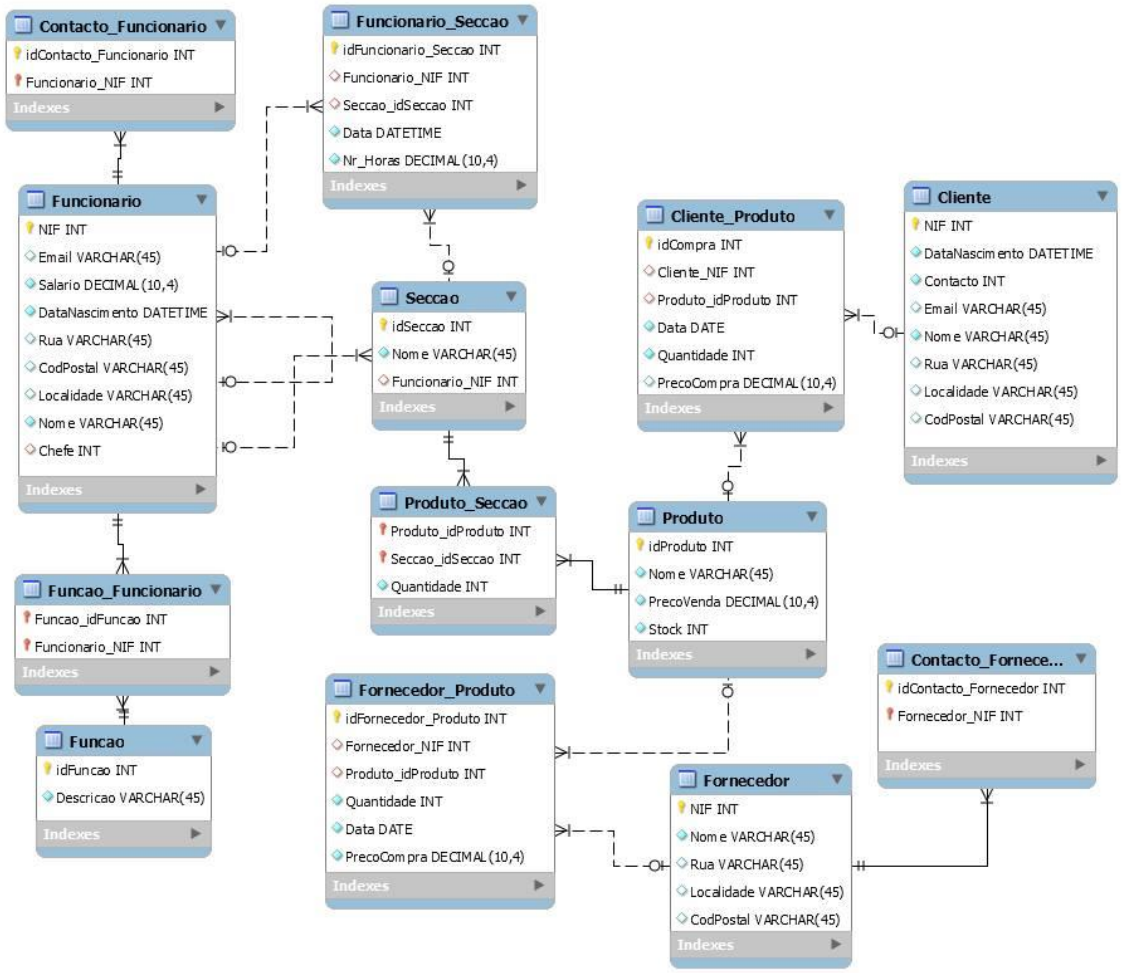


Figura 7. Versão do modelo lógico mais atualizado

3.1. Derivação das relações

Esta secção é dedicada a uma breve explicação das relações segundo as suas composições.

Funcionario(NIF,Email,Salario,DataNascimento,Rua,CodPostal,Localidade,Nome,Chefe)

Chave primária: NIF
Chave estrangeira: Chefe

Contacto_Funcionario(idContacto_Funcionario,Funcionario_NIF)

Chave primária: idContacto_Funcionario, Funcionario_NIF
Chave estrangeira: Funcionario_NIF

Funcao_Funcionario(Funcao_idFuncao,Funcionario_NIF)

Chave primária: Funcao_idFuncao,Funcionario_NIF
Chave estrangeira: Funcao_idFuncao,Funcionario_NIF

Funcao(idFuncao,Descricao)

Chave primária: idFuncao

Funcionario_Seccao(idFuncionario_Sec,Funcionario_NIF,Seccao_idSeccao,Data,Nr_Horas)

Chave primária: idFuncinario_Seccao
Chave estrangeira: Funcionario_NIF,Seccao_idSeccao

Seccao(idSeccao,Nome,Funcionario_NIF)

Chave primária: idSeccao
Chave estrangeira:Funcionario_NIF

Produto_Seccao(Produto_idProduto,Seccao_idSeccao,Quantidade)

Chave primária: Produto_idProduto,Seccao_idSeccao
Chave estrangeira: Produto_idProduto,Seccao_idSeccao

Fornecedor_Produto(idFornecedor_Produto,Fornecedor_NIF,Produto_idProduto,Quantidade,

Data,Preco)

Chave primária: idFornecedor_Produto
Chave estrangeira: Fornecedor_NIF,Produto_idProduto

Cliente_Produto(idCompra,Cliente_NIF,Produto_idProduto,Data,Quantidade)

Chave primária: idCompra
Chave estrangeira: Cliente_NIF,Produto_idProduto

Produto(idProduto,Nome,Preco,Stock)

Chave primária: idProduto

Fornecedor(NIF,Nome,Rua,Localidade,CodPostal)

Chave primária: NIF

Cliente(NIF,DataNascimento,Contacto,Email,Nome,Rua,Localidade,CodPostal)

Chave primária: NIF

Contacto_Fornecedor(idContacto_Fornecedor,Fornecedor_NIF)

Chave primária: idContacto_Fornecedor,Fornecedor_NIF
Chave estrangeira: Fornecedor_NIF

3.2. Validação das relações através da normalização

Na tradução do modelo conceptual para o modelo lógico existem, por vezes, algumas anomalias que podem tornar a base de dados inconsistente. A normalização ajuda a reduzir as referidas anormalidades através de várias formas normais.

No nosso caso de estudo, iremos abordar as 3 primeiras formas, fundamentando as decisões tomadas.

3.2.1. 1ª Forma Normal

No modelo conceptual existem duas entidades que possuem um atributo multi-valor: contacto. A primeira forma normal verifica-se quando todos os atributos de todas as relações possuem valores atómicos. Na transição para o modelo lógico, o grupo teve em conta que os atributos multi-valor dão origem a uma tabela adicional pelo que de imediato, também se confirma que é respeitada a primeira forma normal.

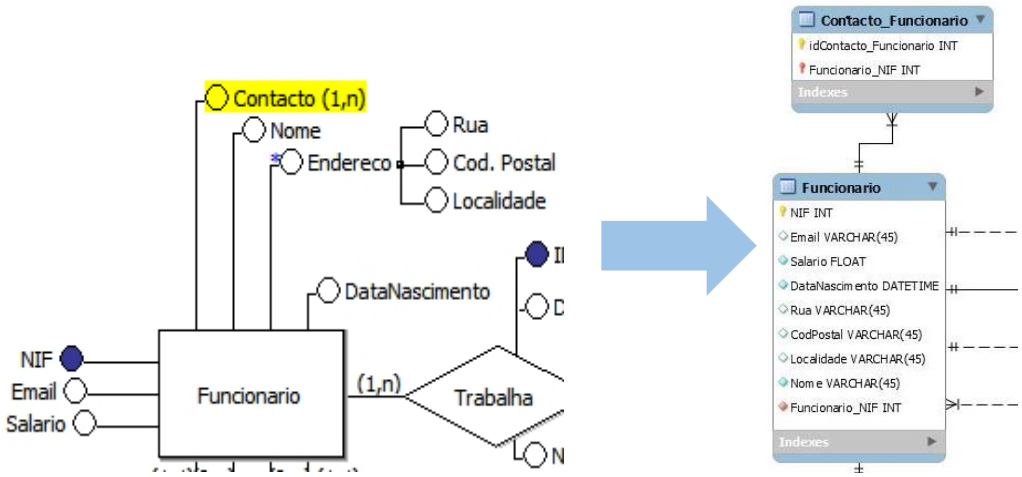


Figura 8. 1ª Forma normal (funcionário)

E da mesma forma, para a entidade fornecedor.

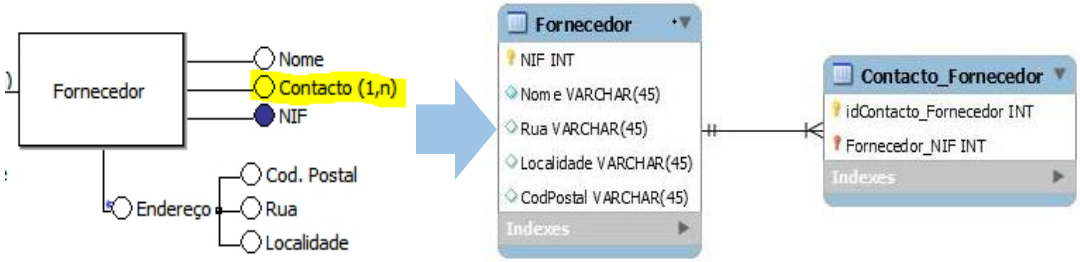
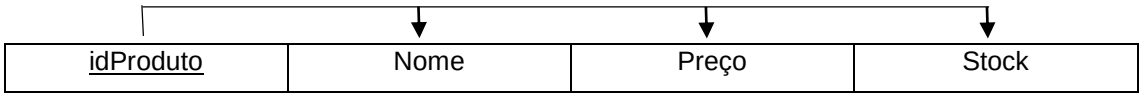


Figura 9. 1ª Forma normal (fornecedor)

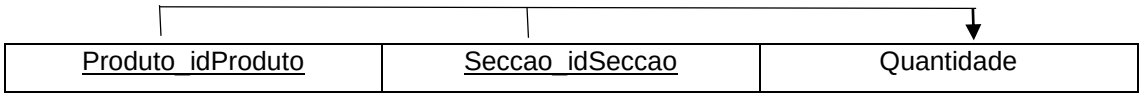
Note que a entidade cliente também possui um atributo contacto, no entanto é um atributo simples pois na nossa perspetiva, assumimos que seria apenas um contacto e não vários.

3.2.2. 2ª Forma Normal

Para melhor justificarmos a validade da segunda forma normal vamos expor alguns exemplos que demonstram, na totalidade, as dependências dos atributos. A segunda forma é respeitada quando um atributo que não pertence à chave primária é totalmente dependente desta.
Ou seja, no caso de uma chave primária composta, um atributo que não pertença à chave tem de ser funcionalmente dependente de todos os atributos que formam a chave. Por questões de simplicidade, iremos apresentar apenas dois exemplos ilustrativos, um para cada caso.
Por exemplo, na tabela produto:



Vemos que a chave primária determina funcionalmente os restantes atributos. Olhando agora para a tabela Produto_Seccao, temos uma chave primária composta por dois atributos. Contudo, o atributo quantidade é determinado por ambos os atributos que formam a chave. Assim se verifica que se respeita a segunda forma normal.



3.2.3. 3ª Forma Normal

A terceira forma normal trata de eliminar as dependências transitivas, isto é, trata de remover os atributos que são funcionalmente dependentes de outros atributos que não pertencem à chave. Na primeira versão do modelo lógico, haviam duas tabelas que continham um atributo que era calculado a partir de dois atributos da mesma tabela.

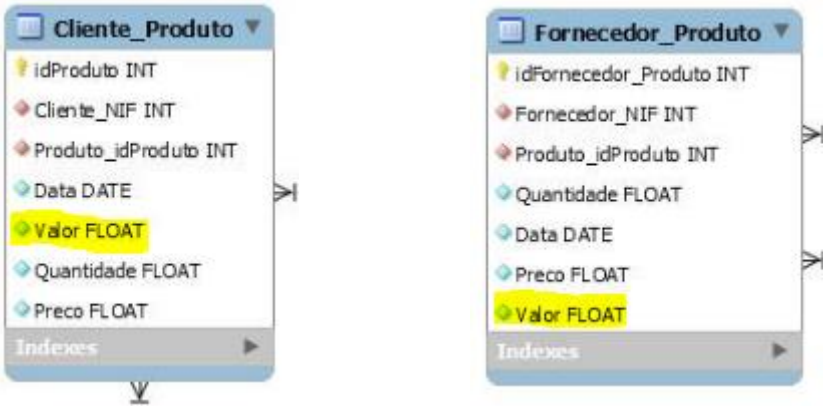
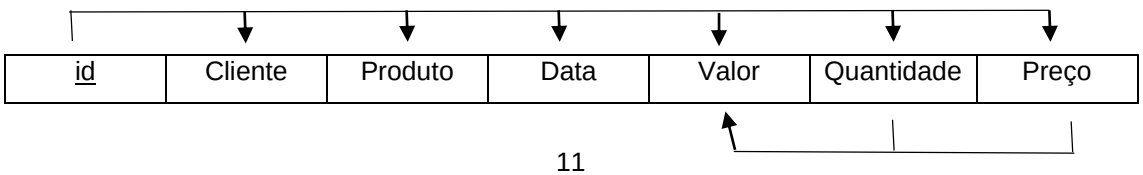
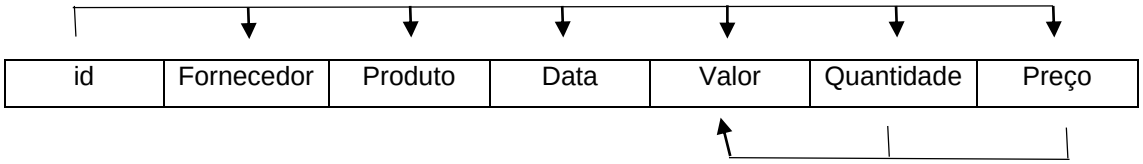


Figura 10. 3ª Forma normal

Portanto a nível de dependências funcionais tínhamos, respetivamente, algo do género:





Logo, a solução encontrada foi remover o atributo Valor de ambas as tabelas conseguindo assim, respeitar a 3ª Forma normal.

3.3. Validação das relações com as transações

Apresentamos, de seguida, duas transações que considerámos essenciais no contexto deste problema. Tratam-se de ações frequentes e requerem um especial cuidado, nomeadamente, quando acontecem várias dessas ações em simultâneo. Assim, as transações definidas são:

- Um cliente compra um produto
- Um fornecedor abastece um produto

Os mapas de transações podem ser consultados no anexo 3.

3.3.1. Compra de um produto

A primeira transação definida foi a da compra de um produto por parte de um determinado cliente. É, muito provavelmente, umas das operações mais básicas num hipermercado. No entanto, se não existir rigor a nível de transações podem ocorrer inconsistências não só a nível das quantias armazenadas e/ou vendidas de cada produto como também acerca do valor faturado. Esta transação é descrita da seguinte forma:

Inicia-se com uma inserção na tabela Cliente_Produto, com o respetivo ID da compra, do cliente e do produto, data e quantidade vendida. De seguida, na tabela Produto_Seccao remove-se a quantidade vendida. Por último, na tabela Produto atualiza-se a quantidade de produto que foi vendido.

3.3.2. Abastecimento de um produto

Relativamente à segunda transação, é bastante análoga à anterior. A diferença está que em vez de se reduzir a quantidade de produto, aumenta-se. Portanto, esta transação segue as seguintes fases:

Inicia com uma inserção do registo na tabela Fornecedor_Produto. Depois, adiciona-se a quantia respetiva na tabela Produto terminando com a atualização da quantidade do produto referido na tabela Produto_Seccao.

3.4. Regras de integridade

As restrições de integridade garantem uma base de dados completa, precisa e consistente. Para atingir tal objetivo é importante que se respeitem as regras que se seguem:

- **Necessidade de valores** – No desenvolvimento do dicionário de dados, alguns atributos têm, obrigatoriamente, que possuir um valor válido, ou seja, não nulo.
- **Restrições do domínio do atributo** – Todos os atributos têm de ter associado uma gama de valores os quais podem tomar. O domínio dos atributos pode ser consultado no anexo 1 (Tabela 3), referente aos dicionários de dados.
- **Multiplicidade** – Na passagem do modelo conceptual para o modelo lógico surgiram novos relacionamentos, como é o caso dos atributos multi-valor, onde passa a existir um relacionamento de cardinalidade 1:N com a tabela adicional. A nível dos relacionamentos N:M os relacionamentos com a nova tabela passam a ser de 1:N para os dois "sentidos".
- **Integridade da entidade** – Esta regra diz que a chave primária de uma entidade nunca pode assumir valores nulos. Pela análise do dicionário de dados conseguimos ver que esta regra está bem explícita.
- **Integridade referencial** – O valor de uma chave estrangeira numa relação "filho" deverá ser e existir na chave primária da relação "pai" a ela associada. Portanto, sempre qua uma relação "pai" sofre uma alteração ou remoção, essa alteração tem que se refletir nas chaves estrangeiras às quais está ligada. Eis alguns exemplos ilustrativos (antes disso, assuma que a entidade "filho" é a que contem a chave estrangeira e que a entidade "pai" contem a chave primária):
    - Inserir um registo na relação filho – Apenas se verifica se a chave estrangeira na relação filho têm alguma correspondência com o valor da chave primária na relação pai.
    - Remover um registo da relação filho – Uma remoção na relação filho não afeta a integridade em qualquer aspeto.
    - Atualização de um registo na relação filho – O procedimento é totalmente igual ao de inserir um registo na relação filho.
    - Inserir um registo na relação pai – Uma inserção na relação pai não interfere com a integridade referencial. Contudo de modo manter a informação atualizada e consistente é necessário proceder à atualização nas respetivas relações filho.
    - Remover registo na relação pai – No caso da remoção de um registo na relação pai a integridade referencial pode ser posta em causa, na medida em que pode existir um registo numa relação filho sem uma entidade pai. Neste sentido, foi decidido que iriamos utilizar a funcionalidade *SET NULL* que apesar da remoção da chave estrangeira, mantem na mesma o registo. A razão desta escolha foi baseada no facto que querermos manter todo o histórico de registos, quer seja de horas de trabalho, quer seja de produtos comprados e/ou vendidos.

- Atualizar a chave primária num registo pai – A nível da atualização da chave primária num registo pai decidimos que usaríamos a funcionalidade *CASCADE* uma vez que trata de atualizar automaticamente todos os registos onde esta chave primária aparece como chave estrangeira. Isto é, se numa determinada situação atualizarmos uma chave primária, uma outra tabela que tenha esta chave como chave estrangeira fica devidamente atualizada graças a este mecanismo de integridade.

3.5. Análise do crescimento futuro

O tamanho inicial da nossa Base de Dados, ou seja, a nossa *script* de povoamento inicial para testes, contem as seguintes quantidades de registos:

- Funções: 6
- Funcionários: 9
- Contactos dos funcionários: 14
- Funções dos funcionários: 18
- Secções: 5
- Produtos: 25
- Funcionários em secção: 18
- Produto em secção: 25
- Fornecedores: 8
- Contactos dos fornecedores: 8
- Clientes: 8
- Registos de compras dos clientes: 8
- Registos dos abastecimentos: 9

Em termos de crescimento futuro torna-se complicado estimar o aumento do tamanho da Base de Dados visto que não possuímos quaisquer elementos informativos sobre o dia-a-dia de um hipermercado, nomeadamente, acerca das vendas, funcionários que estiveram de serviço assim como os próprios abastecimentos, entre outras coisas. Posto isto, não nos é possível analisar, com a devida precisão, o crescimento futuro da base de dados.

4. Modelo Físico

Nesta secção realizamos a tradução do modelo lógico anteriormente apresentado para o modelo físico a ser implementado no SGBD. Para além disso, esta etapa tem como objetivos otimizar a forma como os dados são armazenados e acedidos. Destacam-se ainda os seguintes passos:

- Tradução do modelo lógico para o SGBD
- Organização dos ficheiros e índices
- Desenho das vistas dos utilizadores
- Mecanismos de segurança
- Monitorizar e afinar o sistema operativo

4.1. Tradução do modelo lógico para o SGBD

A primeira fase da modelação física corresponde à tradução do modelo lógico para o SGBD que pretendemos implementar. Nela iniciamos com o desenho das relações base.

4.1.1. Desenho das relações base

Para cada relação apresentamos a informação referente ao domínio, valores por defeito, valores nulos e possíveis restrições.

Relação Funcionário

```
CREATE TABLE IF NOT EXISTS `trabalho`.`Funcionario` (
  `NIF` INT NOT NULL,
  `Email` VARCHAR(45) NULL unique,
  `Salario` DECIMAL(10,4) NOT NULL check(Salario>200),
  `DataNascimento` DATETIME NOT NULL,
  `Rua` VARCHAR(45) NULL,
  `CodPostal` VARCHAR(45) NULL,
  `Localidade` VARCHAR(45) NULL,
  `Nome` VARCHAR(45) NOT NULL,
  `Chefe` INT NULL,
  PRIMARY KEY (`NIF`),
  INDEX `fk_Funcionario_Funcionario1_idx` (`Chefe` ASC),
  CONSTRAINT `fk_Funcionario_Funcionario1`
    FOREIGN KEY (`Chefe`)
    REFERENCES `trabalho`.`Funcionario` (`NIF`)
    ON DELETE set null
    ON UPDATE cascade

    )
ENGINE = InnoDB;
```

Figura 11. Relação funcionário

Relação Secção

```
CREATE TABLE IF NOT EXISTS `trabalho`.`Seccao` (
  `idSeccao` INT NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(45) NOT NULL unique,
  `Funcionario_NIF` INT NULL,
  PRIMARY KEY (`idSeccao`),
  INDEX `fk_Seccao_Funcionario1_idx` (`Funcionario_NIF` ASC),
  CONSTRAINT `fk_Seccao_Funcionario1`
    FOREIGN KEY (`Funcionario_NIF`)
    REFERENCES `trabalho`.`Funcionario` (`NIF`)
    ON DELETE set null
    ON UPDATE cascade)
ENGINE = InnoDB;
```

Figura 12. Relação secção

Relação Cliente

```
CREATE TABLE IF NOT EXISTS `trabalho`.`Cliente` (
  `NIF` INT NOT NULL,
  `DataNascimento` DATETIME NOT NULL ,
  `Contacto` INT NOT NULL,
  `Email` VARCHAR(45) NULL unique,
  `Nome` VARCHAR(45) NOT NULL,
  `Rua` VARCHAR(45) NULL,
  `Localidade` VARCHAR(45) NULL,
  `CodPostal` VARCHAR(45) NULL,
  PRIMARY KEY (`NIF`)
  )
ENGINE = InnoDB;
```

Figura 13. Relação cliente

Relação Função

```
CREATE TABLE IF NOT EXISTS `trabalho`.`Funcao` (
  `idFuncao` INT NOT NULL AUTO_INCREMENT,
  `Descricao` VARCHAR(45) NOT NULL unique,
  PRIMARY KEY (`idFuncao`))
ENGINE = InnoDB;
```

Figura 14. Relação função

Relação Produto

```
CREATE TABLE IF NOT EXISTS `trabalho`.`Produto` (
  `idProduto` INT NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(45) NOT NULL unique,
  `PrecoVenda` DECIMAL(10,4) NOT NULL check(PrecoVenda>0),
  `Stock` INT NOT NULL check(Stock>0),
  PRIMARY KEY (`idProduto`))
ENGINE = InnoDB;
```

Figura 15. Relação produto

Relação Fornecedor

```
CREATE TABLE IF NOT EXISTS `trabalho`.`Fornecedor` (
  `NIF` INT NOT NULL ,
  `Nome` VARCHAR(45) NOT NULL,
  `Rua` VARCHAR(45) NULL,
  `Localidade` VARCHAR(45) NULL,
  `CodPostal` VARCHAR(45) NULL,
  PRIMARY KEY (`NIF`))
ENGINE = InnoDB;
```

Figura 16. Relação fornecedor

Relação Contacto_Funcionario

```
CREATE TABLE IF NOT EXISTS `trabalho`.`Contacto_Funcionario` (
  `idContacto_Funcionario` INT NOT NULL,
  `Funcionario_NIF` INT NOT NULL,
  PRIMARY KEY (`idContacto_Funcionario`, `Funcionario_NIF`),
    FOREIGN KEY (`Funcionario_NIF`)
    REFERENCES `trabalho`.`Funcionario` (`NIF`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

Figura 17. Relação contacto funcionário

Relação Funcionario_Seccao

```
CREATE TABLE IF NOT EXISTS `trabalho`.`Funcionario_Seccao` (
  `idFuncionario_Seccao` INT NOT NULL AUTO_INCREMENT,
  `Funcionario_NIF` INT NULL,
  `Seccao_idSeccao` INT NULL,
  `Data` DATETIME NOT NULL,
  `Nr_Horas` DECIMAL(10,4) NOT NULL,
  PRIMARY KEY (`idFuncionario_Seccao`),
  INDEX `fk_Funcionario_has_Seccao_Seccao1_idx` (`Seccao_idSeccao` ASC),
  INDEX `fk_Funcionario_has_Seccao_Funcionario_idx` (`Funcionario_NIF` ASC),
  CONSTRAINT `fk_Funcionario_has_Seccao_Funcionario`
    FOREIGN KEY (`Funcionario_NIF`)
    REFERENCES `trabalho`.`Funcionario` (`NIF`)
    ON DELETE set null
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Funcionario_has_Seccao_Seccao1`
    FOREIGN KEY (`Seccao_idSeccao`)
    REFERENCES `trabalho`.`Seccao` (`idSeccao`)
    ON DELETE set null
    ON UPDATE cascade)
ENGINE = InnoDB;
```

Figura 18. Relação funcionário secção

Relação Produto_Seccao

```
CREATE TABLE IF NOT EXISTS `trabalho`.`Produto_Seccao` (
  `Produto_idProduto` INT NOT NULL,
  `Seccao_idSeccao` INT NOT NULL,
  `Quantidade` INT NOT NULL check(Quantidade>0),
  PRIMARY KEY (`Produto_idProduto`, `Seccao_idSeccao`),
  INDEX `fk_Produto_has_Seccao_Seccao1_idx` (`Seccao_idSeccao` ASC),
  INDEX `fk_Produto_has_Seccao_Produto1_idx` (`Produto_idProduto` ASC),
  CONSTRAINT `fk_Produto_has_Seccao_Produto1`
    FOREIGN KEY (`Produto_idProduto`)
    REFERENCES `trabalho`.`Produto` (`idProduto`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Produto_has_Seccao_Seccao1`
    FOREIGN KEY (`Seccao_idSeccao`)
    REFERENCES `trabalho`.`Seccao` (`idSeccao`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

Figura 19. Relação produto secção

Relação Fornecedor_Produto

```
CREATE TABLE IF NOT EXISTS `trabalho`.`Fornecedor_Produto` (
  `idFornecedor_Produto` INT NOT NULL AUTO_INCREMENT,
  `Fornecedor_NIF` INT  NULL,
  `Produto_idProduto` INT  NULL,
  `Quantidade` INT NOT NULL check(Quantidade>0),
  `Data` DATE NOT NULL,
  `PrecoCompra` DECIMAL(10,4) NOT NULL check (PrecoCompra>0),
  INDEX `fk_Fornecedor_has_Produto_Produto1_idx` (`Produto_idProduto` ASC),
  INDEX `fk_Fornecedor_has_Produto_Fornecedor1_idx` (`Fornecedor_NIF` ASC),
  PRIMARY KEY (`idFornecedor_Produto`),
  CONSTRAINT `fk_Fornecedor_has_Produto_Fornecedor1`
    FOREIGN KEY (`Fornecedor_NIF`)
    REFERENCES `trabalho`.`Fornecedor` (`NIF`)
    ON DELETE set null
    ON UPDATE cascade,
  CONSTRAINT `fk_Fornecedor_has_Produto_Produto1`
    FOREIGN KEY (`Produto_idProduto`)
    REFERENCES `trabalho`.`Produto` (`idProduto`)
    ON DELETE set null
    ON UPDATE cascade)
ENGINE = InnoDB;
```

Figura 20. Relação fornecedor produto

Relação Cliente_Produto

```
CREATE TABLE IF NOT EXISTS `trabalho`.`Cliente_Produto` (
  `idCompra` INT NOT NULL AUTO_INCREMENT,
  `Cliente_NIF` INT  NULL,
  `Produto_idProduto` INT  NULL,
  `Data` DATE NOT NULL,
  `Quantidade` INT NOT NULL check(Quantidade>0),
  `PrecoCompra` DECIMAL(10,4) NULL check (PrecoCompra>0),
  PRIMARY KEY (`idCompra`),
  INDEX `fk_Cliente_has_Produto_Produto1_idx` (`Produto_idProduto` ASC),
  INDEX `fk_Cliente_has_Produto_Cliente1_idx` (`Cliente_NIF` ASC),
  CONSTRAINT `fk_Cliente_has_Produto_Cliente1`
    FOREIGN KEY (`Cliente_NIF`)
    REFERENCES `trabalho`.`Cliente` (`NIF`)
    ON DELETE set null
    ON UPDATE cascade,
  CONSTRAINT `fk_Cliente_has_Produto_Produto1`
    FOREIGN KEY (`Produto_idProduto`)
    REFERENCES `trabalho`.`Produto` (`idProduto`)
    ON DELETE set null
    ON UPDATE cascade)
ENGINE = InnoDB;
```

Figura 21. Relação cliente produto

Relação Funcao_Funcionario

```
CREATE TABLE IF NOT EXISTS `trabalho`.`Funcao_Funcionario` (
  `Funcao_idFuncao` INT NOT NULL,
  `Funcionario_NIF` INT NOT NULL,
  PRIMARY KEY (`Funcao_idFuncao`, `Funcionario_NIF`),
  INDEX `fk_Funcao_has_Funcionario_Funcionario1_idx` (`Funcionario_NIF` ASC),
  INDEX `fk_Funcao_has_Funcionario_Funcao1_idx` (`Funcao_idFuncao` ASC),
  CONSTRAINT `fk_Funcao_has_Funcionario_Funcao1`
    FOREIGN KEY (`Funcao_idFuncao`)
    REFERENCES `trabalho`.`Funcao` (`idFuncao`)
    ON DELETE cascade
    ON UPDATE cascade,
  CONSTRAINT `fk_Funcao_has_Funcionario_Funcionario1`
    FOREIGN KEY (`Funcionario_NIF`)
    REFERENCES `trabalho`.`Funcionario` (`NIF`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

Figura 22. Relação função funcionário

Relação Contacto_Fornecedor

```
CREATE TABLE IF NOT EXISTS `trabalho`.`Contacto_Fornecedor` (
  `idContacto_Fornecedor` INT NOT NULL,
  `Fornecedor_NIF` INT NOT NULL,
  PRIMARY KEY (`idContacto_Fornecedor`, `Fornecedor_NIF`),
  INDEX `fk_Contacto_Fornecedor_Fornecedor1_idx` (`Fornecedor_NIF` ASC),
  CONSTRAINT `fk_Contacto_Fornecedor_Fornecedor1`
    FOREIGN KEY (`Fornecedor_NIF`)
    REFERENCES `trabalho`.`Fornecedor` (`NIF`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

Figura 23. Relação contacto fornecedor

4.1.2. Desenho das representações dos dados derivados

Na passagem do modelo conceptual para o modelo lógico existia um atributo denominado por "Valor" nas tabelas Cliente_Produto e Forncedor_Produto que correspondia à multiplicação dos atributos "Quantidade" e "Preço".

Contudo, depois da normalização através da terceira forma normal, esse atributo "Valor" acabou por ser removido e desta forma deixamos de ter atributos derivados. Podia, no entanto, haver um atributo derivado que seria o salário do funcionário. O salario podia, por exemplo, ser um atributo que fosse calculado segundo a função do funcionário e das horas que trabalhou em cada umas secções onde está inserido.

Por questões de simplicidade o grupo considerou que o salário do funcionário seria um valor constante e atribuído no momento da inserção do registo. Com essa decisão não existe qualquer atributo que seja calculado tendo em conta os valores de outros atributos.

4.1.3. Desenho das restrições gerais

A nível de restrições gerais, o nosso grupo, quis demonstrar apenas alguns casos ilustrativos desta funcionalidade. Portanto, como se pode ver pelas figuras que apresentamos abaixo existem algumas restrições, do género:

- O salário de um funcionário tem de ser superior a 200€
- Um funcionário precisa de ser maior de idade (pelo menos 18 anos) para poder trabalhar no hipermercado

```
CREATE TABLE IF NOT EXISTS `trabalho`.`Funcionario` (
  `NIF` INT NOT NULL,
  `Email` VARCHAR(45) NULL unique,
  `Salario` DECIMAL(10,4) NOT NULL check(Salario>200),
  `DataNascimento` DATETIME NOT NULL,
  `Rua` VARCHAR(45) NULL,
  `CodPostal` VARCHAR(45) NULL,
  `Localidade` VARCHAR(45) NULL,
  `Nome` VARCHAR(45) NOT NULL,
  `Chefe` INT NULL,
  PRIMARY KEY (`NIF`),
  INDEX `fk_Funcionario_Funcionario1_idx` (`Chefe` ASC),
  CONSTRAINT `fk_Funcionario_Funcionario1`
    FOREIGN KEY (`Chefe`)
    REFERENCES `trabalho`.`Funcionario` (`NIF`)
    ON DELETE set null
    ON UPDATE cascade

    )
ENGINE = InnoDB;
```

Figura 24. Restrição do salário de um funcionário

Relativamente à restrição da idade do funcionário procedemos de uma forma diferente, visto que não conseguimos implementar tal e qual como o salário recorrendo á funcionalidade *CHECK*. Assim sendo, a alternativa encontrada foi a criação de um *trigger* que verifica, antes de cada inserção na tabela Funcionário, se o novo registo contém a idade mínima para trabalhar.

```
delimiter $$
create trigger anoFuncionario
before insert on funcionario
for each row
begin
dECLARE msg varchar(255);

if((datediff(now(),new.DataNascimento))/365<18)
Then
 set msg= 'Não tem idade minima para trabalhar';
          SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;

end if;

end $$
```

Figura 25. *Trigger* que garante que um funcionário tem 18 anos

Para além das restrições que expomos, existem muitas outras que podem ser implementadas, tais como:

- A quantidade de um produto tem de ser igual ou superior a 0
- Não podem existir produtos com um preço negativo
- Um cliente não pode comprar mais quantidade do que aquela que está exposta
- Não podem existir secções com ID diferentes mas com nomes iguais
- Repor, automaticamente, um produto que tenha atingido um limite mínimo de quantidade exposta
- Impor uma capacidade de armazenamento de modo a que não seja infinito

4.2. Organização dos ficheiros e índices

Esta secção aborda a forma como estão guardados os ficheiros que constituem o sistema de base de dados. Falaremos também acerca dos índices que foram criados no sentido de aumentar a *performance* do sistema.

4.2.1. Análise de transações

Nesta secção iremos analisar mais detalhadamente as transações já apresentadas no capítulo 3, na parte da modelação lógica. Considere as transações em baixo:

- Compra de um produto
- Abastecimento de um produto

No anexo 3 encontram-se disponíveis os mapas de transações correspondentes a cada uma das transações anteriormente apresentadas.

Compra de um produto por parte do cliente

A entidade produto é uma das entidades chaves neste universo. Deste modo, sempre que é registada uma compra é inserido um registo na tabela Cliente_Produto. Como é óbvio, a compra resulta numa diminuição da quantidade de produtos expostos na secção onde está colocado, logo, é necessário que se atualize a tabela Produto_Seccao removendo a quantidade do produto comprada. Por último, na tabela Produto atualiza-se a quantidade de produto que foi vendido.
A figura 26 (apresentada abaixo) corresponde à transação em questão, contudo, não traduz totalmente o procedimento exposto. Para esta situação decidimos criar um *trigger* que ajudasse a verificar uma questão que já foi referida na parte do desenho de restrições gerais. Trata-se da questão de um cliente querer comprar uma quantidade de produto superior à que está à venda. A figura 27 é correspondente a esse *trigger* que efetua tal verificação antes da inserção do novo registo na tabela Cliente_Produto.

```
DELIMITER $$
create procedure inserirCliente(in NIFCliente INT, in dataNascimentoC datetime, in
                                ContactoCliente INT, in emailCliente VARCHAR(45), in nomeCliente
                                varchar(45), IN ruaCliente varchar(45), IN localidadeCliente varchar(45),
                                 in codPostalCliente varchar(45), in quantidadeComprada int,
                                 in produtoComprado INT)

begin

declare erro bool default 0;

declare continue handler for sqlexception set erro=1;

start transaction;

INSERT INTO Cliente
    (NIF, DataNascimento,Contacto,Email,Nome,Rua,Localidade,CodPostal )
    VALUES
    (NIFCliente,dataNascimentoC,ContactoCliente,emailCliente,nomeCliente,
    ruaCliente,localidadeCliente,codPostalCliente);

INSERT INTO Cliente_Produto
    (idCompra, Cliente_NIF, Produto_idProduto, Data, Quantidade,PrecoCompra)
    Values
    (idCompra,NIFCliente,produtoComprado,now(),quantidadeComprada,null);


if erro
then rollback;
else commit;
end if;
```

Figura 26. Transação da compra de um produto por parte de um cliente

```
delimiter $$
create trigger inserirPreco
before insert on cliente_produto
for each row

begin

declare QuantiadeStock int;
declare preco decimal (10,4);
declare msg varchar(255);

      select Stock into QuantiadeStock
      from produto
      where new.Produto_idProduto=idProduto;

if(new.Quantidade>QuantiadeStock)
   then set msg= concat_ws(',','Só tens ',QuantiadeStock,' unidades para comprares');
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;
else
    select PrecoVenda INTO preco
    from produto
    where produto.idProduto=new.Produto_idProduto;

    set New.PrecoCompra=preco;

    update produto
    set Stock=Stock-new.Quantidade
    where idProduto=new.Produto_idProduto;

 end if;
end $$
```

Figura 27. *Trigger* que garante uma quantidade positiva do produto

Abastecimento de um produto

Muito analogamente à transação anterior, temos agora um aumento na quantidade de um produto, ou seja, o abastecimento de um produto. O abastecimento inicia-se com a inserção de um registo na tabela Fornecedor_Produto contendo o ID do abastecimento, do fornecedor, do produto, quantidade, data e o preço por unidade. De seguida, atualiza-se a tabela produto, adicionando a quantidade de abastecimento (veja o *trigger* da figura 29). Esta transação termina com a atualização da tabela Produto_Seccao através da adição da quantidade de produto fornecida.

```
delimiter $$
CREATE PROCEDURE insereProduto (in nomeProduto varchar(45),in precoVenda decimal(10,4),
                                in stockProduto int, in seccaoProduto int, in quantidadeSeccao int,
                                in fornecedor int, in precoComprado decimal(10,4))

BEGIN

DECLARE Erro BOOL DEFAULT 0;

DECLARE produtoID INT;
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET Erro = 1;

start transaction;

INSERT INTO produto (idProduto,Nome,PrecoVenda,Stock) VALUES
                    (idProduto,nomeProduto,precoVenda,0);

SELECT idProduto INTO produtoID
FROM produto
ORDER BY idProduto DESC
LIMIT 1;

INSERT INTO fornecedor_produto (idFornecedor_Produto,Fornecedor_NIF,Produto_idProduto,Quantidade, Data, PrecoCompra) VALUES
                               (idFornecedor_Produto,fornecedor,produtoID, stockProduto,now(),precoComprado);

INSERT INTO Produto_Seccao (Produto_idProduto,Seccao_idSeccao,Quantidade) VALUES
                           (produtoID, seccaoProduto,quantidadeSeccao);

 IF Erro THEN
        ROLLBACK;
    ELSE
        COMMIT;
    END IF;
end $$
```

Figura 28. Transação de abastecimento de um produto

```
delimiter $$
create trigger incrementarStock
after insert on fornecedor_produto
for each row
begin

update produto
set Stock=Stock+new.Quantidade
where new.Produto_idProduto=idProduto;

end $$
```

Figura 29. *Trigger* que incrementa o *stock* de produto

Pelo que podemos espectar, as tabelas mais acedidas serão as tabelas que derivam na transição do modelo conceptual para o modelo lógico, em especial, as dos relacionamentos de muitos para muitos, tais como:

- Funcionario_Seccao

- Produto_Seccao
- Cliente_Produto
- Fornecedor_Produto

Estas tabelas estarão em constante atualização devido ao facto de estarem associadas a um DATETIME e terem um registo para cada um deles. No que diz respeito aos produtos, estes podem também sofrer atualizações como acontece, por exemplo, nas promoções.

4.2.2. Escolha da organização os ficheiros

A organização dos ficheiros que constituem a base de dados em armazenamento secundário determina significativamente a eficiência no acesso aos dados. A Base de Dados foi criada em MySQL pelo que é o motor da mesma que decide a forma de organizar a informação.
O motor usado foi o InnoDB que mantém a sua área de *buffer* para armazenar dados e índices em memória principal. A forma como os ficheiros estão organizados acaba por ser ainda mais eficiente do que mapeamento (*hashing*). Para além disso, o motor disponibiliza meios de suporte às transações (ACID – Atomicidade, Consistência, Isolamento e Durabilidade) e para chaves estrangeiras.

4.2.3. Escolha de índices

O motor InnoDB coloca em todas as tabelas um índice onde guarda a informação dos registos. Habitualmente este índice é sinónimo da chave primária da tabela. Assim sendo, apresentamos apenas dois índices ilustrativos como exemplo de aplicação.

```
create index precoProduto on produto(Nome,PrecoVenda ASC);
create index funcionarioSalario on funcionario(Nome,Salario ASC);
```

Figura 30. Índices

4.2.4. Estimativa de espaço em disco necessário

A estimativa de espaço em disco necessário foi realizada tendo em conta os seguintes tamanhos de cada domínio:

- Integer – 4 bytes
- Decimal (10,4) – 6 bytes
- Date – 3 bytes
- Datetime – 8 bytes
- Varchar (N) - 1 + N bytes

Depois de termos calculado manualmente, o espaço necessário para cada tabela é o seguinte:

- Contacto_Funcionario – 8 bytes (14 registos)
- Funcao_Funcionario – 8 bytes (18 registos)
- Função – 50 bytes (6 registos)

- Funcionário – 252 bytes (9 registos)
- Funcionario_Seccao – 26 bytes (18 registos)
- Cliente – 246 bytes (8 registos)
- Cliente_Produto – 25 bytes (8 registos)
- Produto – 60 bytes (25 registos)
- Fornecedor – 188 bytes (8 registos)
- Contacto_Fornecedor – 64 bytes (8 registos)
- Fornecedor_Produto – 25 bytes (9 registos)
- Produto_Seccao – 12 bytes (25 registos)
- Secção – 54 bytes (5 registos)

Na secção 3.5 está apresentado o número de registos para o povoamento inicial e pelo que podemos calcular que o espaço inicial necessário para a base de dados é de, aproximadamente, 8.98 KB. Este resultado foi obtido através da multiplicação do número de registos de cada tabela pelo número de bytes ocupado por registo.

4.3. Desenho das vistas dos utilizadores

Uma das questões importantes no desenvolvimento de um Sistema de Gestão de Base de Dados é a restrição das vistas de cada utilizador. A nossa implementação tem em conta que existirão N tipos de utilizadores:

- Administrador – Tem acesso a toda a Base de Dados e pode executar todas as ações que entender.
- Chefe de secção – Tem acesso a toda a informação exceto os dados pessoais dos clientes.
- Funcionário – Tem os mesmos privilégios do que o chefe de secção, no entanto, com a exceção de não conhecer os fornecedores.
- Cliente – Pode consultar os produtos que estão expostos, ver as compras que lhe dizem respeito e os seus dados pessoais.

Em baixo apresentamos, para as várias vistas possíveis, o código SQL respetivo.

```
CREATE VIEW informacaoFuncionarios as
        SELECT f.NIF AS 'NIF',
               F.Nome AS 'Nome',
               F.Email AS 'Email',
               CF.idContacto_Funcionario AS 'Telefone'
        FROM funcionario AS F
        inner JOIN contacto_funcionario AS CF on F.NIF=CF.Funcionario_NIF;
```

Figura 31. Vista sobre informação sobre os funcionários

```
create view trabalhoFuncionaio as
        select f.salario as 'Salário',
               fu.descricao as 'Função',
               s.nome as 'Secção',
               f.nome as 'Nome'
        from funcionario as f
        inner join funcao_funcionario as ff on ff.Funcionario_NIF = f.nif
        inner join funcao as fu on fu.idFuncao=ff.Funcao_idFuncao
        inner join funcionario_seccao as fs on fs.Funcionario_NIF = f.nif
        inner join seccao as s on s.idSeccao = fs.Seccao_idSeccao;
```

Figura 32. Vista sobre a função de um funcionário

```
create view funcionarioResponsavel as
          SELECT F.NIF AS 'NIF',
                 F.Nome as 'Nome',
                 S.Nome AS 'Nome Secção'
          from funcionario AS F
          inner join seccao AS S on F.NIF= S.Funcionario_NIF;
```

Figura 33. Vista sobre os responsáveis de secção

```
create view informacaoCliente as
        select NIF AS 'NIF',
               Nome AS 'Nome',
               Email AS 'Email',
               Contacto AS 'Telefone'
        from cliente;
```

Figura 34. Vista sobre a informação dos clientes

```
create view compraCliente as
        select c.Nome as 'Nome cliente',
               cp.quantidade as 'Quantidade',
               p.Nome as 'Nome do produto',
               p.precovenda as 'Preço Produto'
         from cliente as c
         inner join cliente_produto as cp on cp.Cliente_NIF= c.nif
         inner join produto as p on p.idProduto = cp.Produto_idProduto;
```

Figura 35. Vista sobre as compras dos clientes

```
create view informacaoFornecedor as
        select idContacto_Fornecedor as 'Contacto Fornecedor',
               f.Nome as 'Nome fornecedor',
               f.NIF as 'Nif'
        from fornecedor as f
        inner join contacto_fornecedor as cf on f.nif = cf.Fornecedor_NIF;
```

Figura 36. Vista sobre os dados dos fornecedores

```
create view produtosFornecidos as
        select p.Nome as 'Produto',
               s.Nome as 'Secção',
               p.precovenda as 'Preço',
               fp.quantidade as 'Quantidade do produto',
               f.nome as 'Nome fornecedor'
        from fornecedor as f
        inner join fornecedor_produto as fp on fp.Fornecedor_NIF=f.nif
        inner join produto as p on p.idProduto = fp.Produto_idProduto
        inner join produto_seccao as ps on ps.Produto_idProduto=p.idProduto
        inner join seccao as s on s.idSeccao = ps.Seccao_idSeccao;
```

Figura 37. Informação sobre os abastecimentos de produtos

4.4. Mecanismos de segurança

Relativamente aos mecanismos de segurança, podemos dizer que vão de encontro com o tema abordado na secção precedente. Ou seja, cada tipo de utilizador tem determinados privilégios que lhe permite ver apenas a parte da Base de Dados que lhe diz respeito. Por exemplo, não faz sentido que um cliente possa ver os dados pessoais de um funcionário. Ou por exemplo, no caso do administrador, este tem de ser capaz de gerir o quer que seja, logo tem de ser capaz de executar qualquer ação, seja ela qual for (inserção, remoção e atualização).

Assim sendo, a nível de ações os vários utilizadores podem fazer o seguinte:

- Administrador – Como já foi referido, é o único utilizador com permissão para adicionar, remover ou atualizar quaisquer tabela.
- Chefe de secção – Complementando a informação sobre as vistas, os chefes de secção podem ainda adicionar ou atualizar todas as tabelas, à exceção dos dados dos clientes.
- Funcionário – As mesmas permissões do que os chefes de secção com a exceção de não poderem alterar as tabelas relativas aos fornecedores.
- Clientes - Para além de poderem ver os seus dados pessoais, os clientes também podem altera-los e ver as compras que fizeram.

4.5. Monitorizar e afinar o sistema operativo

Os objetivos principais de uma Base de Dados resumem-se em armazenar a informação de forma estruturada de modo a que o seu acesso seja feita da forma mais fácil e eficiente. A eficiência depende de imensos fatores tais como o espaço do armazenamento, as especificações do sistema relativamente a tempos de resposta ou até mesmo a capacidade de processamento de várias transações em simultâneo.

Isto é, a conjugação de todos esses fatores permite afinar o sistema de modo a obter o melhor equilíbrio possível. Contudo temos de estar conscientes de que, por vezes, a melhoria num fator pode estar a prejudicar outro. Nesses casos, o melhor será fazer um balanço e calcular os custos decidindo de acordo com o que é mais favorável para os utilizadores.

Resumindo, a monitorização e afinação do sistema operativo é um processo contínuo e subjetivo, na medida que pode variar consoante as necessidades de cada utilizador, e deve ser feito, preferencialmente, em ocasiões que o sistema não esteja a ser utilizado.

5. Ferramentas utilizadas

Eis as ferramentas usadas no desenvolvimento deste projeto:

- Microsoft Word 2013 – Elaboração do relatório
- brModelo 3.0 – Desenho do modelo conceptual
- Github – Controlo de versões
- MySQL Workbench – Criação e manipulação da base de dados
- Microsoft SQL Server - Sistema de gestão de base de dados relacional

6. Conclusões e Trabalho Futuro

Através da realização deste trabalho prático foi possível adquirir e consolidar um conjunto de competências relacionadas com o processo de criação e desenvolvimento de SGBD.

Desde o início do trabalho, ou seja, desde a tomada de decisão sobre o tema do problema, da análise do mesmo e da sua contextualização, que começamos a definir quais seriam os objetivos aos quais nos íamos propor assim como seria estruturado este processo de evolução.

Na fase seguinte, começamos com a primeira parte da modelação que corresponde à modelação conceptual onde efetuamos o levantamento de requisitos do sistema, a identificação e descrição das entidades envolventes bem como a descrição dos relacionamentos entre as entidades. Esta fase terminou com a criação do modelo conceptual que serviu de base na representação do sistema que foi implementado.

Na parte da modelação lógica procedemos à tradução do modelo concetual para o modelo lógico. No sentido de aumentar a consistência do sistema foram verificadas as três primeiras regras de normalização e várias regras de integridade. Ainda nesta fase, foi feita uma análise a duas possíveis transações, nosso ver, as mais relevantes, e um estudo sobre o crescimento futuro da base de dados.

No que diz respeito à modelação física fizemos a transição do modelo lógico para SGBD. Esta transição incluiu o desenho das relações e das restrições gerais do sistema. Para além disso, foi também abordada a organização dos ficheiros, explicando mais detalhadamente as transações propostas na fase anterior. Foi possível perceber quais as tabelas que seriam mais acedidas e estimar o espaço em disco ocupado de acordo com a nossa *script* de povoamento. Depois vieram as questões relacionadas com os utilizadores do SGBD, nomeadamente, sobre os privilégios e as permissões de cada um deles. Para terminar, foi feita uma pequena abordagem à temática da monitorização e afinação do sistema, onde podemos concluir que se trata de um processo contínuo e muito relativo consoante, não só as especificações físicas do sistema, mas também com as necessidades do utilizador final.

Em termos de trabalho futuro existem imensos pontos que podem ser adicionados ou até mesmo melhorados. Por exemplo, adicionar mais registos à Base de Dados uma vez que contêm apenas registos suficientes para a parte experimental das funcionalidades implementadas. Outro aspeto seria a criação de mais restrições gerais, tais como as que foram sugeridas na secção 4.1.3. Um caso mais subjetivo mas também importante seria a alteração do nome de alteração do nome de alguns atributos para um nome mais sugestivo e mais intuitivo. Por último, talvez a alteração do domínio de alguns atributos

Temos consciência que o trabalho desenvolvido não está preparado, nem perto, de ser aplicado em ambiente real, no entanto, consideramos que foi suficiente para aplicar as noções lecionadas durante o semestre nesta unidade curricular.

Bibliografia

[1] Connolly, T. M. & Begg, C.E., 2005. Database System - A Practical Approach to Design, Implementation and Management 4th Edition

Lista de Siglas e Acrónimos

Esta secção é dedicada, como o título indica, à listagem de siglas e acrónimos.

BD Base de Dados

SGBD Sistema de Gestão de Bases de Dados

N Nulo

MV Multi-valor

Anexos

Os anexos são utilizados no sentido de incluir informação adicional necessária e indispensável para uma melhor compreensão do relatório assim como para complementar alguns tópicos, secções ou assuntos abordados.

Assim sendo, os anexos estão divididos em duas partes:

- Anexo 1 – Referente aos dicionários de dados
- Anexo 2 – Alusiva às versões do modelo conceptual
- Anexo 3 – Relativo aos mapas de transações

I. Anexo 1 – Dicionários de dados

Entidade	Multiplicidade	Relacionamento	Multiplicidade	Entidade
Funcionário	(1,1)	Gere	(0,n)	Funcionário
Funcionário	(1,n)	Tem	(1,n)	Função
Funcionário	(1,1)	Responsável	(1,n)	Secção
Funcionário	(0,n)	Trabalha	(0,n)	Secção
Secção	(1,n)	Contém	(1,n)	Produto
Cliente	(0,n)	Compra	(0,n)	Produto
Produto	(0,n)	É fornecido	(0,n)	Fornecedor

Tabela 1. Dicionário de dados de relacionamentos

Entidade	Descrição	Sinónimos	Ocorrência
Funcionário	Funcionários do hipermercado	Trabalhador	Funcionário trabalha numa ou várias secções; Funcionário gere funcionários; Funcionário responsável por secção; Funcionário tem uma ou várias funções;
Função	Funções dos funcionários	Cargo	Vários funcionários exercem várias funções;
Secção	Secções do hipermercado	Área	Funcionário trabalha numa ou várias secções; Funcionário responsável por secção; Uma secção contém vários produtos;
Produto	Produtos armazenados / adquiridos	Artigo	Uma secção contém vários produtos;

			Vários clientes compram vários produtos; Vários produtos são fornecidos por vários fornecedores;
Cliente	Clientes do hipermercado	Comprador	Vários clientes compram vários produtos;
Fornecedor	Fornecedores de produtos	Abastecedor	Vários produtos são fornecidos por vários fornecedores;

Tabela 2. Dicionário de dados das entidades

Entidade	Atributos	Descrição	Domínio	N	MV	Tipo	Tamanho
Funcionário	Nome	Nome do funcionário	Caracteres variáveis	Não	Não	VARCHAR	45
	NIF	Número de identificação fiscal	Inteiro	Não	Não	INTEGER	9
	Salario	Salário mensal	Vírgula flutuante	Não	Não	DECIMAL	(10,4)
	Email	Correio eletrónico	Caracteres variáveis	Sim	Não	VARCHAR	45
	Contacto	Contacto telefónico	Inteiro	Não	Sim	INTEGER	9
	Data de nascimento	Data de nascimento	Data	Não	Não	DATE	-
	Rua	Rua	Caracteres variáveis	Sim	Não	VARCHAR	45
	Localidade	Localidade	Caracteres variáveis	Sim	Não	VARCHAR	45
	Código Postal	Código Postal	Caracteres variáveis	Sim	Não	VARCHAR	45
Função	ID	Identificador da função	Inteiro	Não	Não	INTEGER	9
	Descrição	Breve descrição da função	Caracteres variáveis	Sim	Não	VARCHAR	45

Secção	ID	Identificador da secção	Inteiro	Não	Não	INTEGER	4
	Nome	Nome da secção	Caracteres variáveis	Não	Não	VARCHAR	45
Produto	ID	Identificador do produto	Inteiro	Não	Não	INTEGER	9
	Nome	Nome	Caracteres variáveis	Não	Não	VARCHAR	45
	Preço	Preço de venda	Vírgula flutuante	Não	Não	DECIMAL	(10,4)
	Stock	Quantidade armazenada	Inteiro	Não	Não	INTEGER	9
Cliente	NIF	Número de identificação fiscal	Inteiro	Não	Não	INTEGER	9
	Nome	Nome do cliente	Caracteres variáveis	Não	Não	VARCHAR	45
	Email	Correio eletrónico	Caracteres variáveis	Sim	Não	VARCHAR	45
	Contacto	Contacto telefónico	Inteiro	Não	Não	INTEGER	9
	Data de nascimento	Data de nascimento	Data	Não	Não	DATE	-
	Rua	Rua	Caracteres variáveis	Sim	Não	VARCHAR	45
	Localidade	Localidade	Caracteres variáveis	Sim	Não	VARCHAR	45
	Código postal	Código Postal	Caracteres variáveis	Sim	Não	VARCHAR	45
Fornecedor	NIF	Número de identificação fiscal	Inteiro	Não	Não	INTEGER	9
	Nome	Nome do fornecedor	Caracteres variáveis	Não	Não	VARCHAR	45
	Contacto	Contacto telefónico	Inteiro	Não	Sim	INTEGER	9
	Rua	Rua	Caracteres variáveis	Sim	Não	VARCHAR	45
	Localidade	Localidade	Caracteres variáveis	Sim	Não	VARCHAR	45

	Código Postal	Código Postal	Caracteres variáveis	Sim	Não	VARCHAR	45

Tabela 3. Dicionário de dados dos atributos das entidades

II. Anexo 2 – Versões do modelo conceptual

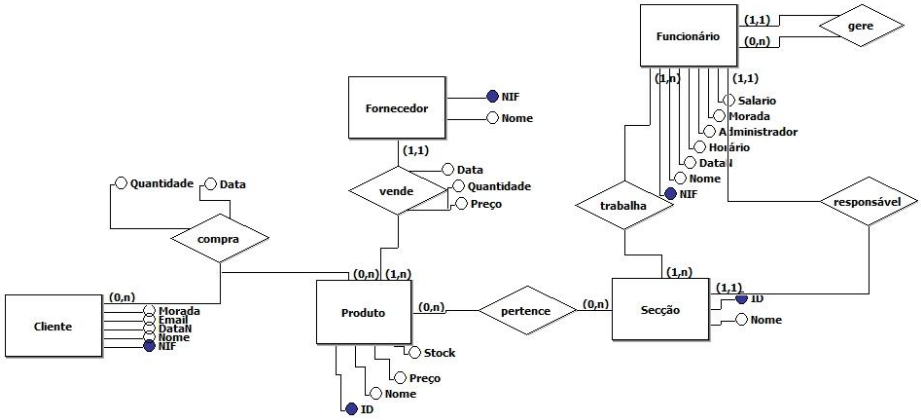


Figura 38. Modelo conceptual - versão 1

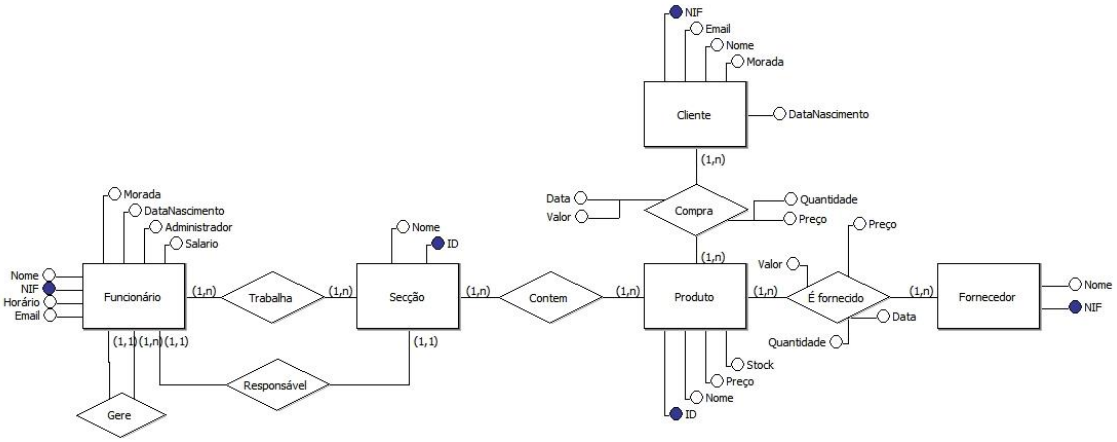


Figura 39. Modelo conceptual - versão 2

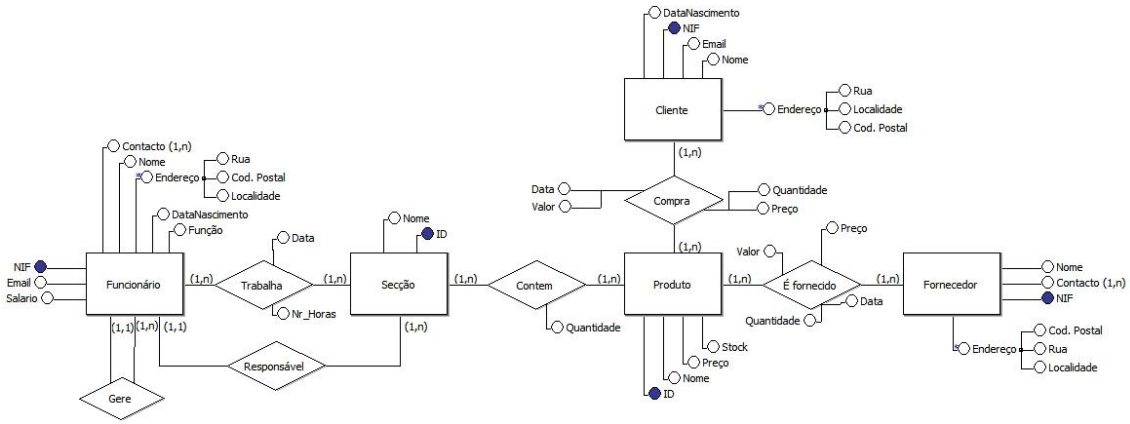


Figura 40. Modelo conceptual - versão 3

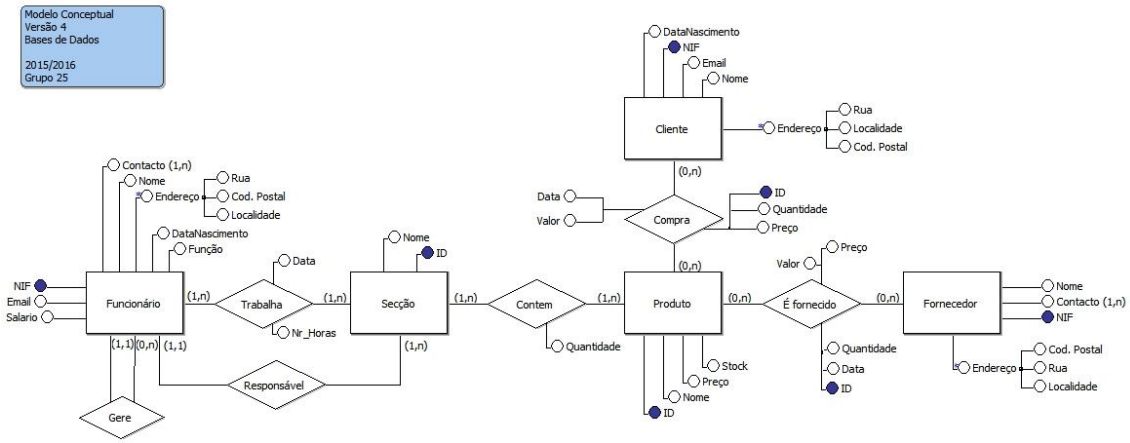


Figura 41. Modelo conceptual - versão 4

III. Anexo 3 – Mapa de transações

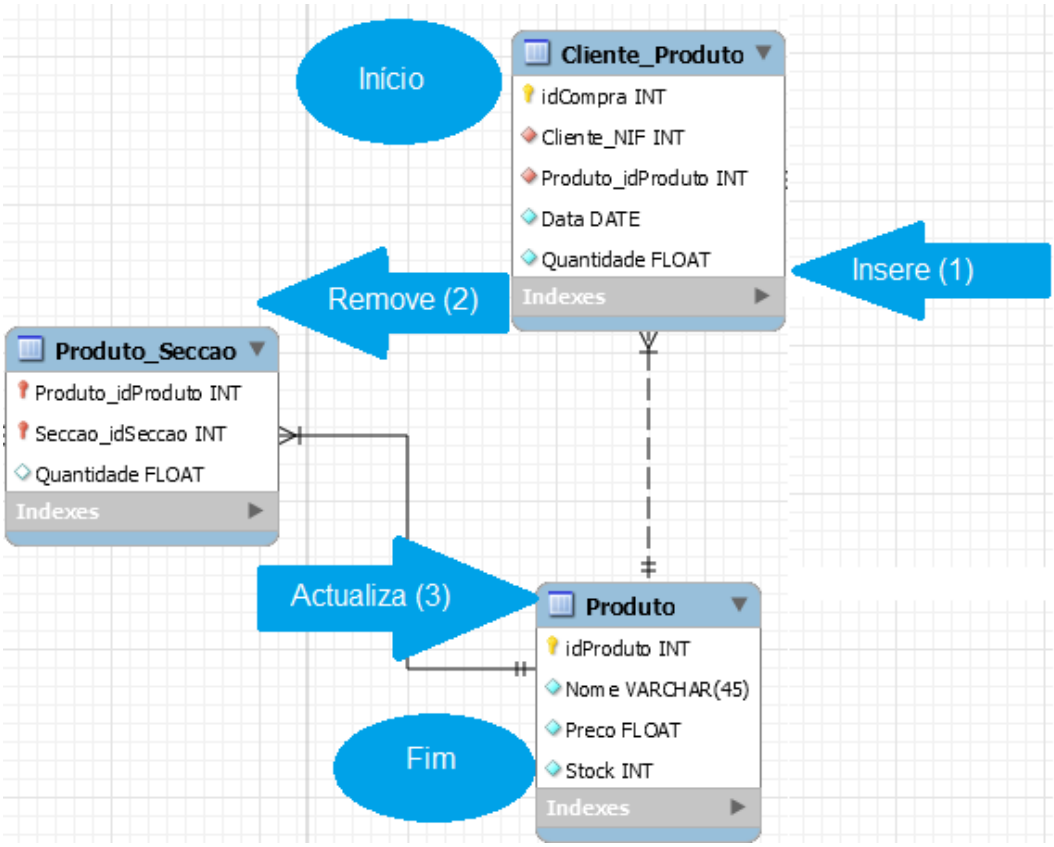


Figura 42. Mapa da transação da compra de um produto

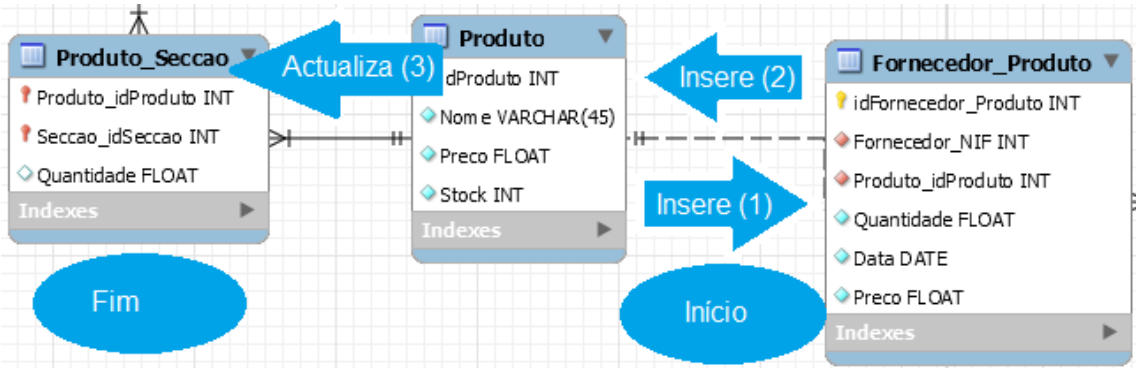


Figura 43. Mapa da transação de abastecimento de stock