M&T's - Sports Line

Bases de Dados

Martim Neves, Tiago Dias

VERSÃO FINAL

M&T's - Sports Line

DEPARTAMENTO DE ELECTRÓNICA,
TELECOMUNICAÇÕES E INFORMÁTICA

Bases de Dados

Martim Neves, Tiago Dias
(88904) martimfneves@ua.pt, (88896) tiago.adonis@ua.pt

9 Junho 2020

Conteúdo

Capítulo 1

Introdução

No âmbito da unidade curricular de Bases de Dados, foi nos proposto uma criação de uma base de dados e seu respetivo modelo para a gestão de informação de uma loja de desporto da empresa "M&T's - Sports Line". Tem-se como objetivo criar também uma aplicação de interação com a respetiva base de dados onde deverá ser possível manipular e visualizar dados de forma simples e rápida.

Tendo em conta uma prévia análise de requisitos acerca das necessidades e capacidades do modelo de dados, efetuaram-se os primeiros diagramas relativos às relações entre as entidades existentes no sistema.

Seria necessário gerir não só qualquer artigo envolvido em todos os processos da loja, mas também todos os clientes e funcionários da mesma. Seria ainda necessário também administrar as diferentes lojas, armazéns, e por fim emitir novas vendas, devoluções e transportes para o sistema.

Capítulo 2

Análise de Requisitos

O nosso projeto é sobre uma marca de desporto chamada "M&T's Sports Line". Atualmente a marca possui várias lojas em várias cidades do nosso país como em Aveiro, Porto e Lisboa, querendo ainda aumentar o seu número. Cada loja é caracterizada por um número que a identifica, um nome e uma localização.

Estas lojas poderão ter um ou mais armazéns, dependendo do seu tamanho e afluência por parte dos clientes. É nos armazéns que são guardados a quantidade de produtos que mais tarde irão ser disponibilizados para as lojas correspondentes. Estes são caracterizados por um ID, a sua capacidade total e a loja á qual disponibilizam os produtos neles contidos.

No que toca aos produtos disponíveis nas várias lojas, estes são identificados por um código, preço, nome, categoria e quantidade. Os seus clientes têm sempre a possibilidade de poder efetuar a compra numa loja física ou se preferirem podem realizar uma encomenda para uma morada á sua escolha através de um transporte feito pela empresa que se caracteriza por um ID, uma data de entrega, um local de destino, código dos produtos transportados e a quantidade de produtos transportados. Caso fiquem descontentes com os produtos ou caso exista algum defeito poderão sempre fazer uma devolução á loja. Devolução esta que necessita de ser efetuada entre o cliente e um funcionário, registando o ID da devolução, a data, o montante devolvido, o código dos produtos devolvidos e a quantidade de produtos devolvidos.

No que diz respeito á possibilidade de realizar entregas para uma morada á escolha do cliente, este pode, se assim desejar, mudar a morada de entrega, data ou a quantidade dos produtos encomendados.

Os dados dos clientes, como o nome, NIF, morada e número de telemóvel, irão ficar guardados numa base de dados correspondente á empresa, tendo estes a escolha de divulgar a sua morada ou não.

As compras na loja são sempre efetuadas entre um cliente e um funcionário da loja em questão, sendo esta compra caracterizada pelo seu número identificador, montante, data, código dos artigos e quantidade de artigos comprados.

As lojas possuem vários funcionários que são identificados pelo seu número de funcionário, nome, morada e o número de telemóvel.

Capítulo 3

Diagrama e Esquema

3.1 Diagrama Entidade-Relação

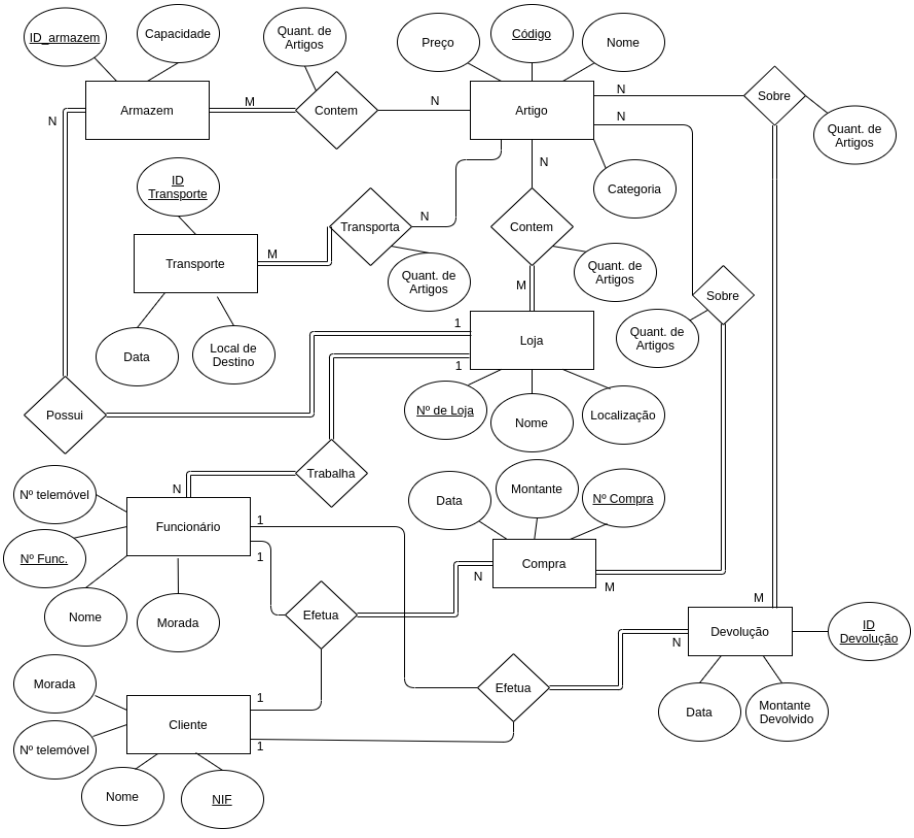

Figura 3.1: Diagrama Entidade-Relação

Falando sobre as diferenças entre este novo diagrama entidade-relação e o que foi entregue na primera fase, as entidades Receção e Venda foram retiradas, uma vez que eram exatamente iguais ás entidades Devolução e Compra, que já existiam. As obrigatoriedades nas relações entre as entidades Devolução, Artigo, Compra, Cliente e Funcionário foram revistas, uma vez que seguem o pressuposto que todas as compras necessitam de um artigo, mas um artigo não necessita de uma compra, por exemplo. A cardinalidade das entidades ditas anteriormente também foi mudada, juntamente com a das entidades Artigo, Armazém e Transporte. Para finalizar todas as alterações efetuadas, foram criadas duas novas relações. Uma entre a entidade Armazém e Artigo e outra entre a entidade Funcionário e Loja, de forma a poder garantir um cenário mais realista.

Tendo já esclarecido o contexto principal do nosso problema, bem como as alterações efetuadas no DER para esta segunda fase, mostramos agora as principais entidades e relações que as envolvem no novo DER que se encontra em cima referenciado.

A nível de entidades podemos considerar a entidade **Artigo** como uma das mais importantes do sistema. Maior parte das relações que ocorrem no sistema incluem esta entidade. Existem ainda as duas entidades **Clientes** e **Funcionários** que são as únicas entidades ditas pessoais que existem no sistema.

3.2 Esquema Relacional

O esquema relacional obtido no final da criação das tabelas na base de dados foi o seguinte:

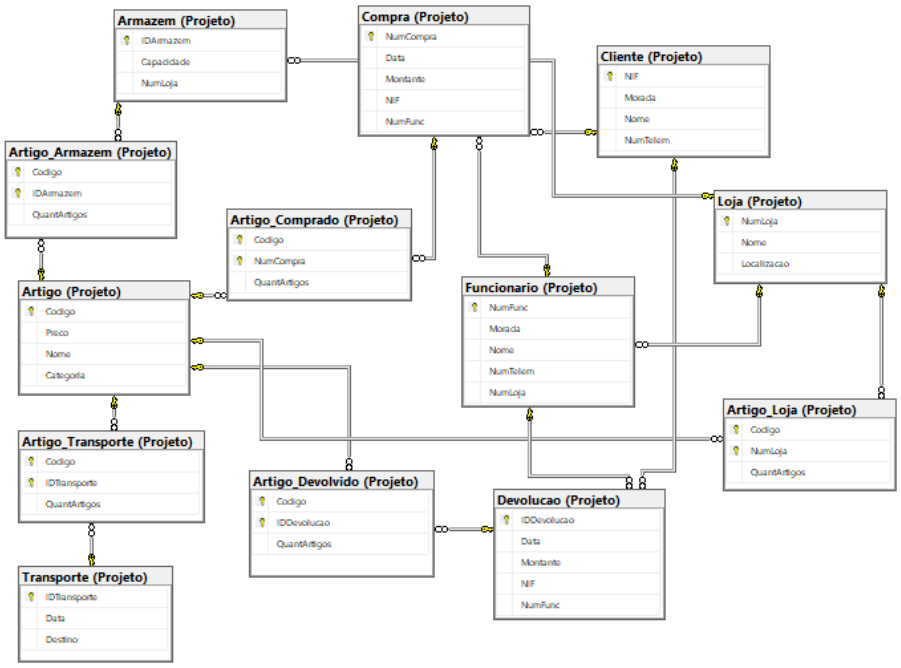

Figura 3.2: Esquema Relacional

Fazendo uma análise comparativa com o nosso primeiro esquema relacional, é necessário dar principal ênfase á criação das tabelas **Artigo_Armazem**, **Artigo_Loja**, **Artigo_Transporte**, **Artigo_Comprado** e **Artigo_Devolvido** que resultaram da cardinalidade de N:M das relações que envolviam a entidade **Artigo**. Estas tabelas permitem assim diferenciar quando um artigo se encontra em armazém, em loja, em transporte, comprado ou devolvido, respetivamente. Ainda devido a esta mudança efetuada sobre a cardinalidade de algumas entidades, a tabela **Artigo** ficou assim com menos atributos.

A tabela **Armazem** contém agora o atributo NumLoja, garantindo assim a ligação direta com a tabela **Loja**, que também teve um novo atributo, Nome. A última mudança efetuada foi relativamente á tabela **Transporte** que já não possui o atributo LocalOrigem, uma vez que este local será uma loja.

Capítulo 4

Implementação

Passando agora para a implementação propriamente dita, e explicando todas as ferramentas utilizadas para o desenvolvimento da Base de Dados, que passa pela utilização de features como as Views, Stored Procedures, User Defined Functions ou Triggers, iremos explicar a sua aplicação em alguns dos casos mais específicos do sistema.

4.1 Base de Dados

4.1.1 Normalização

Não foi necessário efetuar um processo de normalização sobre a nossa base de dados uma vez que não existia qulquer tipo de redundância.

4.1.2 Views

No caso das Views, estas são utilizadas como um primeiro nível de abstração ao modelo de dados. Por exemplo, no caso da exibição dos produtos que existem num determinado armazém ou loja, é necessário fazer uma junção de diversas tabelas para que possamos obter todos os atributos necessários na sua exibição. As Views tornam esta junção possível garantindo assim um acesso total á informação necessária dos produtos em questão, sem ter que juntar constantemente todas as tabelas necessárias em cada query efetuado.

Em baixo encontra-se o código de criação da View que permite listar os produtos de uma determinada loja e o código que permite a sua interação com a interface gráfica.

```
CREATE VIEW StoreProducts AS
SELECT Artigo.Nome AS Name, Preco AS Price, QuantArtigos AS Units,
Loja.NumLoja AS Store
FROM ((Projeto.Loja JOIN Projeto.Artigo_Loja ON
        Loja.NumLoja=Artigo_Loja.NumLoja)
        JOIN Projeto.Artigo ON Artigo_Loja.Codigo=Artigo.Codigo)
```

Listing 4.1: Código de criação da view StoreProducts

```
CMD = New SqlCommand
CMD.Connection = CN
CMD.CommandText = "SELECT Name, Price, Units FROM StoreProducts
    WHERE Store = @store"
CMD.Parameters.Add("@store", SqlDbType.VarChar, 1)
CMD.Parameters("@store").Value = numStore
```

Listing 4.2: Código de interação da view StoreProducts com a interface gráfica

4.1.3 Stored Procedures

Em relação às Stored Procedures, estas são usadas para parametrizar e abstrair todas as operações de escrita - Insert ou Update - bem como operações de remoção de dados. Para além de algumas verificações na inserção ou atualização de alguns dados, como por exemplo na atualização do número total de produtos disponíveis em loja após uma devolução, as SPs ajudam também a melhorar a performance da própria base de dados.

Segue-se excertos do código de criação da SP relativa ao exemplo enteriormente dado e o código da sua interação com a interface gráfica.

```
CREATE PROCEDURE ReturnProduct (@ReturnID INT, @Date Date,
@Value DECIMAL(5,2), @NIF BIGINT, @WorkersCode INT,
@StoreNum INT, @Code INT, @Quant INT)
(...)
IF EXISTS(SELECT * FROM Artigo_Loja WHERE NumLoja=@StoreNum AND
Codigo=@Code)
(...)
    DECLARE @Units INT;
    SELECT @Units=QuantArtigos FROM Artigo_Loja
    WHERE Codigo=@Code AND NumLoja=@StoreNum;

    UPDATE Artigo_Loja SET QuantArtigos=@Units + @Quant
    WHERE Codigo=@Code AND NumLoja=@StoreNum;

    INSERT Devolucao(IDDevolucao, Data, Montante, NIF, NumFunc)
    VALUES (@ReturnID, @Date, @Value, @NIF, @WorkersCode);

    INSERT Artigo_Devolvido(Codigo, IDDevolucao, QuantArtigos)
    VALUES (@Code, @ReturnID, @Quant);
ELSE
    INSERT Artigo_Loja(Codigo, NumLoja, QuantArtigos)
    VALUES (@Code, @StoreNum, @Quant);
```

```
    INSERT Devolucao(IDDevolucao, Data, Montante, NIF, NumFunc)
    VALUES (@ReturnID, @Date, @Value, @NIF, @WorkersCode);

    INSERT Artigo_Devolvido(Codigo, IDDevolucao, QuantArtigos)
    VALUES (@Code, @ReturnID, @Quant);
(...)
```

Listing 4.3: Código de criação da SP ReturnProduct

```
CMD = New SqlCommand()
CMD.Connection = CN
CMD.CommandText = "EXEC Projeto.ReturnProduct @ReturnID,
@Date, @Value, @NIF, @WorkersCode, @StoreNum, @Code, @Quant"
CMD.Parameters.Add("@ReturnID", SqlDbType.Int)
CMD.Parameters.Add("@Date", SqlDbType.Date)
CMD.Parameters.Add("@Value", SqlDbType.Decimal)
(...)
CMD.Parameters("@StoreNum").Value = numStore
CMD.Parameters("@Code").Value = productCode
CMD.Parameters("@Quant").Value = units
```

Listing 4.4: Código de interação da SP ReturnProduct com a interface gráfica

4.1.4 User Defined Functions

Em contraste com as Stored Procedures, as UDFs presentes no sistema final são utilizadas apenas para a leitura de dados da Base de Dados visto terem o benefício de poderem ser utilizadas em sub-queries. Neste caso em particular, são utilizadas para parametrizar algumas leituras, por exemplo para verificar as compras e devoluções que um determinado cliente efetuou.

É deixado em baixo o código de criação da UDF e o código da interação com a interface gráfica deste exemplo anteriormente dado.

```
CREATE FUNCTION Projeto.PurchasedProductsPerClient(@NIF BIGINT)

RETURNS @PurchasedProducts TABLE (Number INT,
Product_Name VARCHAR(40), NUnits INT, Date DATE,
Purchase_Price DECIMAL(5,2))

AS
BEGIN
    INSERT @PurchasedProducts

    SELECT Compra.NumCompra AS Number, Artigo.Nome AS Product_Name,
    Artigo_Comprado.QuantArtigos AS NUnits, Compra.Data AS Date,
    Compra.Montante AS Purchase_Price
    FROM (((Projeto.Artigo_Comprado JOIN Projeto.Compra ON
    Artigo_Comprado.NumCompra=Compra.NumCompra)
    JOIN Projeto.Cliente ON Compra.NIF=Cliente.NIF)
    JOIN Projeto.Artigo ON Artigo_Comprado.Codigo=Artigo.Codigo)
    WHERE Cliente.NIF = @NIF
```

```
    RETURN;
END
```

Listing 4.5: Código de criação da UDF PurchasedProductsPerClient

```
CMD = New SqlCommand()
CMD.Connection = CN
CMD.CommandText = "SELECT * FROM Projeto.PurchasedProductsPerClient
    (@NIF);"
CMD.Parameters.Add("@NIF", SqlDbType.Int)
CMD.Parameters("@NIF").Value = NIF
```

Listing 4.6: Código de interação da UDF PurchasedProductsPerClient com a interface gráfica

4.1.5 Triggers

Por fim, alguns dos Triggers existentes no modelo final garantem a existência de produtos suficientes em loja para que uma compra possa ser efetuada, ou então a possibilidade de adicionar ou não mais unidades ao armazém em questão. Já outros Triggers verificam se numa inserção de um novo cliente ou funcionário o número de telemóvel associado já existe na base de dados.

Para uma melhor compreensão e análise destes Triggers, fornecemos a seguir um exemplo.

```
CREATE TRIGGER Check_UnitsWarehouse ON Artigo_Armazem
AFTER INSERT
AS
    BEGIN
    DECLARE @Warehouse INT;
    DECLARE @Units INT;
    DECLARE @Occupied INT;
    DECLARE @Storage INT;

    SELECT @Warehouse = IDArmazem FROM INSERTED;
    SELECT @Units = QuantArtigos FROM INSERTED;

    SELECT @Storage=Capacidade FROM Armazem WHERE
    IDArmazem=@Warehouse;

    SELECT @Occupied=SUM(QuantArtigos) FROM Artigo_Armazem
    WHERE IDArmazem=@Warehouse;

    IF(@Occupied + @Units > @Storage)
    BEGIN
        RAISERROR('Warehouse storage reached!', 16, 1);
        ROLLBACK TRAN;
    END
END
```

Listing 4.7: Código de criação do Trigger Check_UnitsWarehouse

4.2 Aplicação

Em relação à aplicação, e tal como já mencionado anteriormente, esta é utilizada não só como uma maneira de visualizar e alterar dados da base de dados mas como um ponto de venda de uma loja de produtos de desporto. A ideia principal é não só servir como uma interface de gestão para com a base de dados mas também com os serviços da própria loja. Assim sendo, podemos contar com a possibilidade de realizar compras e devoluções através da aplicação(Figura 4.1) mas também com features como a gestão de funcionários e clientes (Figura 4.2).

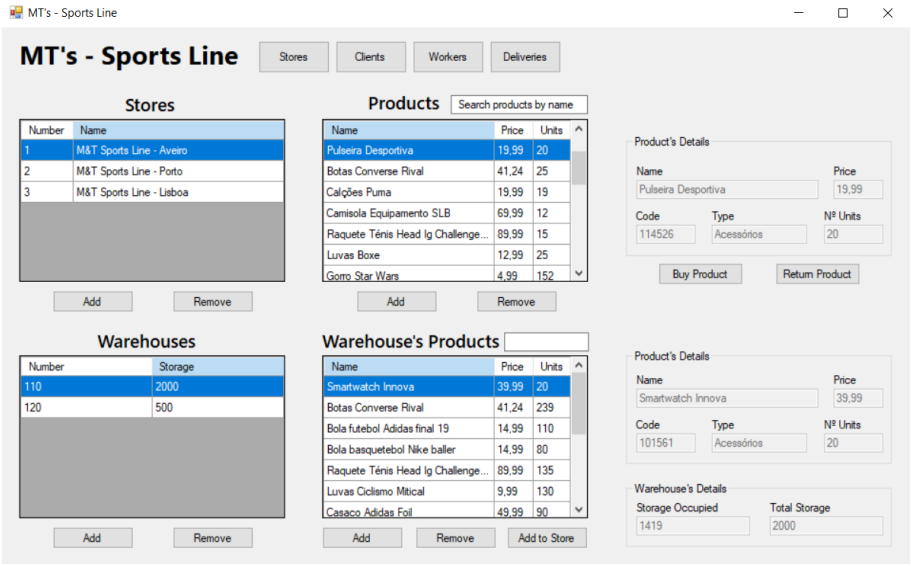

Figura 4.1: Página das lojas

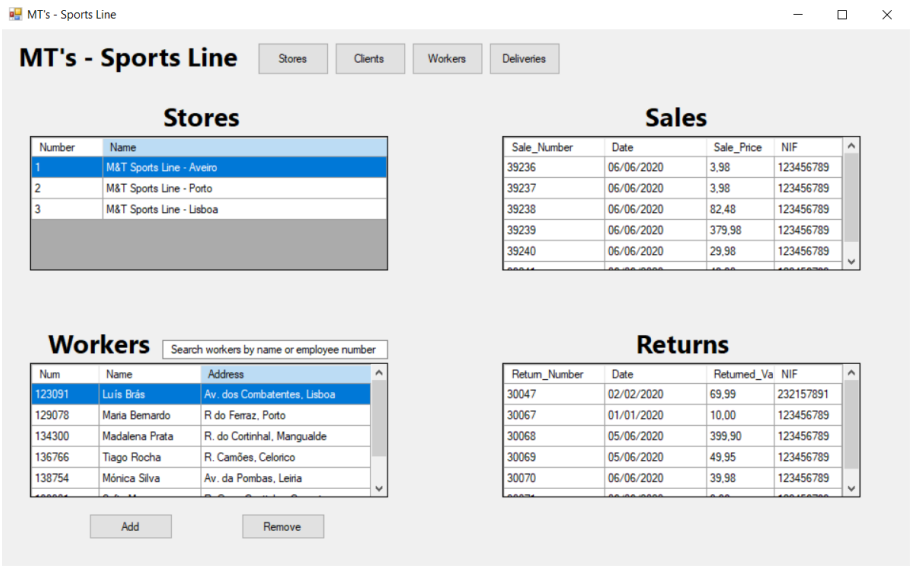

Figura 4.2: Página de gestão dos funcionários

Capítulo 5

Conclusão

No decorrer do projeto vários desafios surgiram. Sendo esta a primeira vez a trabalhar com um modelo de dados maior e com mais rigor, podemos dizer que a versão final do sistema se adequa àquilo que havia sido projetado nas primeiras iterações do projeto. Os requisitos foram cumpridos e grande parte do conteúdo lecionado durante o semestre foi estudado e adaptado ao nosso problema em particular. Assim, é de valor referir que apesar do produto final estar de acordo com as especificações iniciais, existem ainda alguns detalhes que seriam importantes completar e que não foram finalizados no âmbito deste projeto. Seriam estes, por exemplo, uma utilização de mais conteúdos lecionados.