universidade de aveiro deti departamento de electrónica, telecomunicações e informática

Licenciatura em Engenharia Informática
Bases de Dados

Miguel Miragaia 108317 (50%)

Gonçalo Lopes 107572 (50%)

P8-G8

2022/2023

Projeto- Aplicação de gestão de stock e vendas de produtos de Higiene

Índice

Introdução

- No contexto da disciplina de Base de Dados foi desenvolvido um sistema para gerir o stock e as vendas de uma Empresa (Higiliquidos).

- A construção deste sistema foi realizada em SQL(inserção, remoção e manipulação de dados), e uma interface, em C#.

Todo o código implementado encontra-se organizado por pastas.

Na pasta SQL estão disponíveis todos os scripts SQL:

- DDL
- DML
- Procedure
- Indexes
- Triggers
- UDF
- Views

- Na pasta Interface está disponível todo o código C#.

- O ficheiro demo.mp4 corresponde a uma demonstração da interface.

Análise de Requisitos

- Para uma boa criação da base de dados, foi necessário começar por elaborar uma análise de requisitos, sendo para isso realizadas várias pesquisas e perguntas diretamente ao nosso cliente (Higiliquidos), fazendo assim um levantamento detalhado de toda a informação que se considerou ser essencial para o desenvolvimento desta aplicação.

- Passámos, então, para o desenho conceptual, onde foram criadas as entidades, relacionamentos e restrições. Visto que, este processo não é determinístico, houve alguns aspetos que deixaram dúvidas, sendo posteriormente alterados.

Entidades:

- Empresa
- Armazém
- Carrinha
- Carrinha_Distribuidor
- Compra
- Venda
- Produto

- Empresa tem N carrinhas e N armazéns e N pessoas, sendo estes dependentes da empresa para a sua existência.

- Carrinha_Distribuidor é entidade intermédia entre Carrinha e Distribuidor.

- Venda e Compra representam as Encomendas feitas à nossa empresa pelos clientes, e as compras que nós fazemos a um Fornecedor, respetivamente.

- Produto é gerido no Armazém tendo em conta a modulação destas 2 entidades (compra e venda), sendo para adicionar ou retirar uma certa quantidade.

- Venda está ligada com o Produto, Armazém, Distribuidor, Vendedor e Cliente.

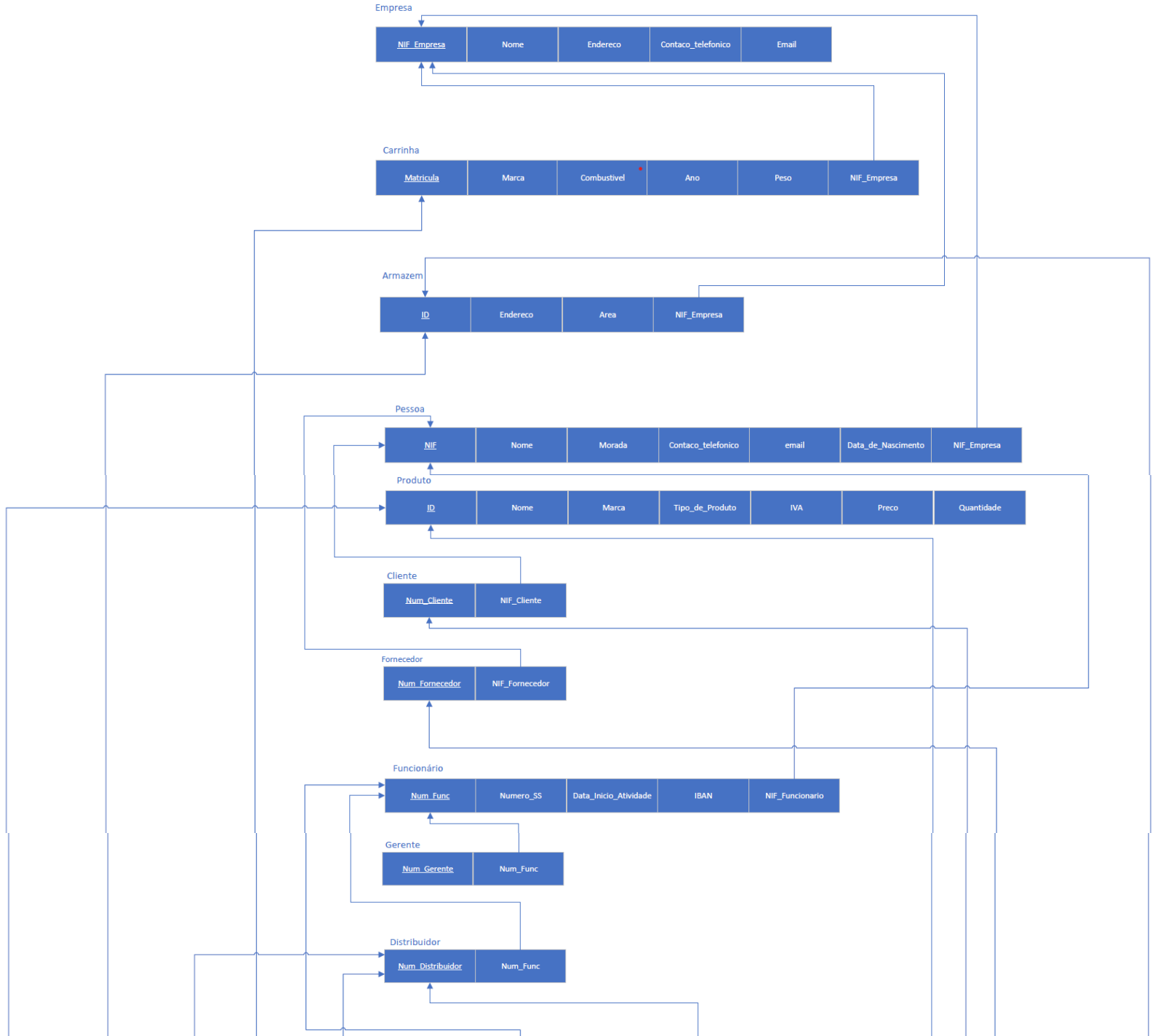
- Compra está ligada ao Produto, Armazém e ao Distribuidor.

Diagramas

Esquema Relacional

- Com o desenvolvimento da base de dados apercebemo-nos de que algumas entidades e relação estariam mal implementadas e com a ajuda do professor realizamos as mudanças por forma a obtermos uma boa solução, bem modulada para a nossa aplicação.

- Sendo a maior mudança evidente, a modulação das "Encomendas" deixou de ser feita a através de uma encomenda em si e passou a ser modulada por uma "Compra" e "Venda" que envolvem várias entidades entre si.



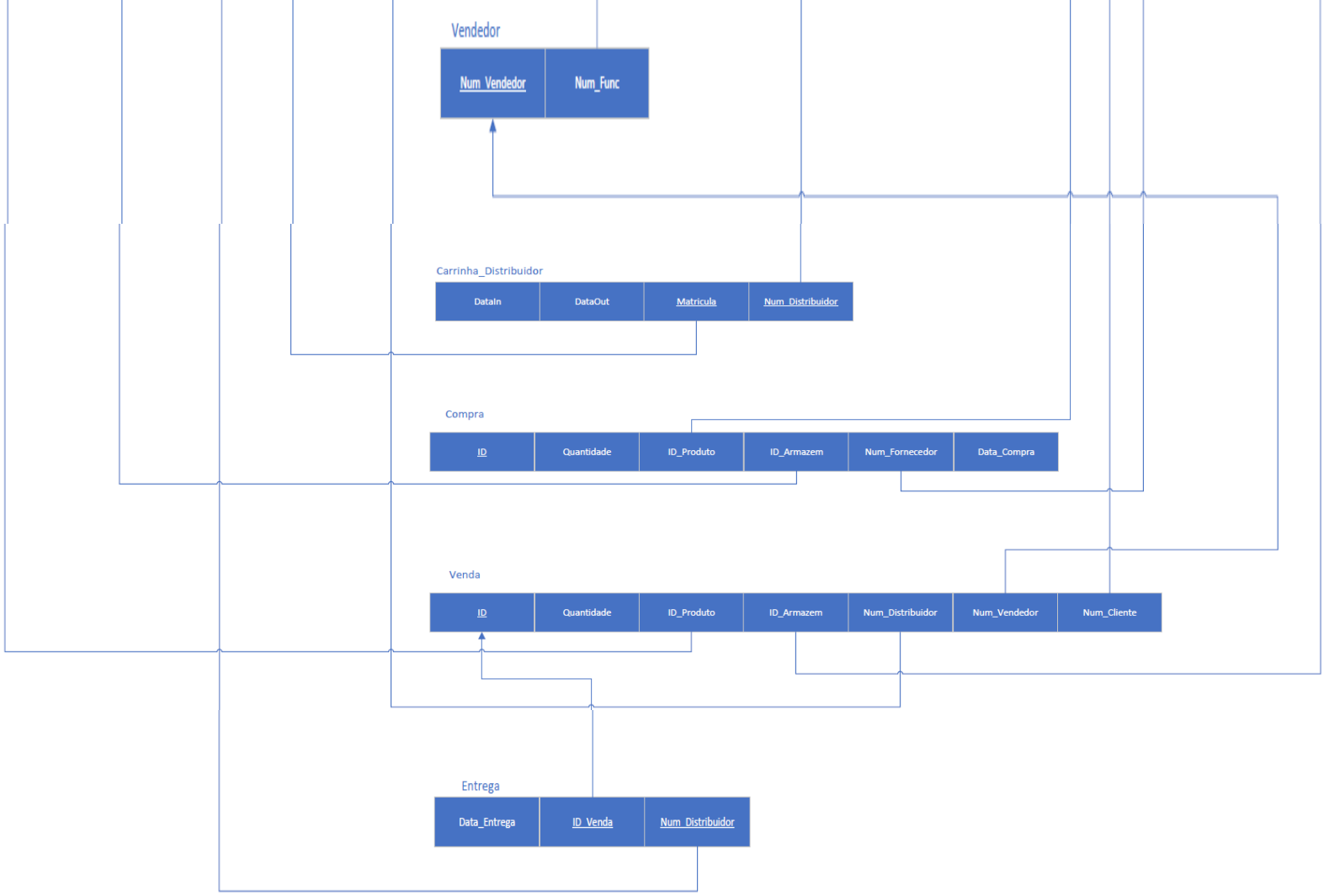
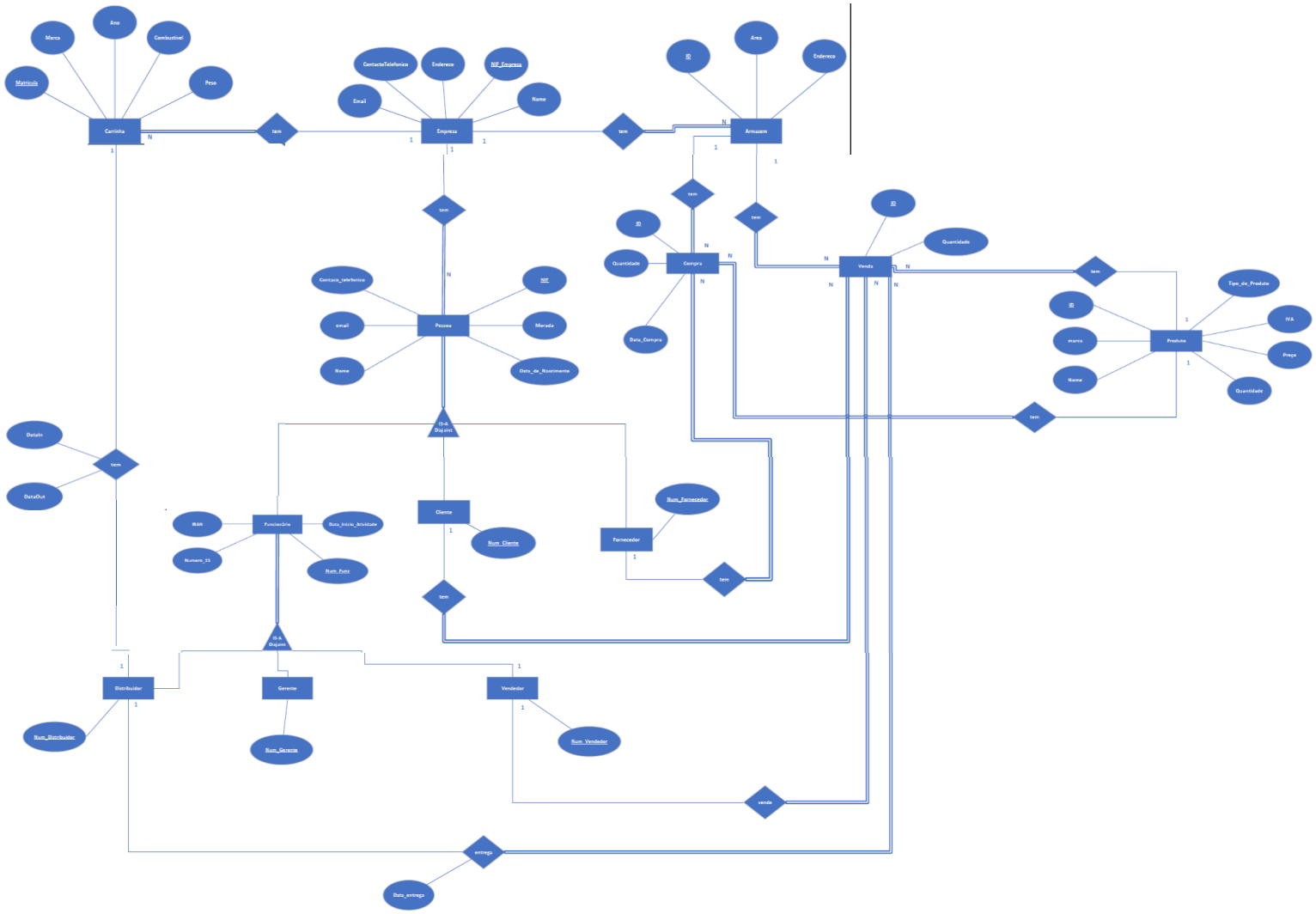

Vendedor
Num_Vendedor
Num_Func
Carrinha_Distribuidor
DataIn
DataOut
Matricula
Num_Distribuidor
Compra
ID
Quantidade
ID_Produto
ID_Armazem
Num_Fornecedor
Data_Compra
Venda
ID
Quantidade
ID_Produto
ID_Armazem
Num_Distribuidor
Num_Vendedor
Num_Cliente
Entrega
Data_Entrega
ID_Venda
Num_Distribuidor

Diagrama Entidade Relacionamento



Linguagem SQL DDL

- Todas as tabelas foram criadas da mesma forma, seguindo, então, a regra Higiliquidos.[tablename].

```
CREATE TABLE Higiliquidos.Empresa (
  NIF_Empresa INT NOT NULL,
  Nome varchar(64) NOT NULL,
  Email varchar(64) NOT NULL,
  Endereco varchar(64) NOT NULL,
  ContactoTelefonico INT NOT NULL,


  PRIMARY KEY (NIF_Empresa)
);
GO
```

Linguagem SQL DML (Data Manipulation Language)

- Todos os comandos associados à manipulação e alteração de dados foram inseridos dentro de estruturas de dados como *Stored Procedures, UDF, Triggers*, Indexes, Views.

Stored Procedures

- A maioria das chamadas feitas à base de dados são através de Stored Procedures, devido a todas as vantagens fornecidas com o uso destas estruturas de dados. Foram criadas Stored Procedures para:

- Adicionar elementos a uma tabela, em alguns casos usando **transações** (exemplo addClient)

```
DROP PROCEDURE addClient
GO
CREATE PROCEDURE addClient
(
    @nif INT = NULL,
    @nome VARCHAR(256) = NULL,
        @num_cliente INT = NULL,
        @dataNasc Date = NULL,
        @email VARCHAR(64) = NULL,
    @morada VARCHAR(64) = NULL,
        @telemovel INT = NULL


)
AS
        BEGIN
                DECLARE @count INT;
                DECLARE @erro VARCHAR(100);
                SET @count = (SELECT Higiliquidos.checkIfNIFExists(@nif))
                IF(@count>=1)
                        RAISERROR ('O NIF introduzido já existe, não é possível adicionar o Cliente', 16,1);
                ELSE
                        BEGIN
                                BEGIN TRY
                                        BEGIN TRAN
                                                INSERT INTO Higiliquidos.Pessoa(NIF, Nome, Data_de_Nascimento, Email, Morada, ContactoTelefonico, NIF_Empresa)
                                                VALUES(@nif, @nome, @dataNasc, @email, @morada, @telemovel, 203551783);

                                                INSERT INTO Higiliquidos.Cliente(NIF_Cliente, Num_Cliente)
                                                VALUES(@nif, @num_cliente);
                                        COMMIT TRAN
                                END TRY
                                BEGIN CATCH
                                        Rollback TRAN
                                                SELECT @erro = ERROR_MESSAGE();
                                                SET @erro =  'O Cliente não foi inserido, algum valor inserido incorretamento'
                                                RAISERROR (@erro, 16,1);
                                END CATCH
                        END
        END
GO
```

- Remover elementos de uma tabela

```
DROP PROCEDURE deleteClient
GO
CREATE PROCEDURE deleteClient
(
    @nif INT
)
AS
BEGIN
        SET NOCOUNT ON;
        DELETE FROM Higiliquidos.Cliente WHERE NIF_Cliente=@nif
        DELETE FROM Higiliquidos.Pessoa WHERE NIF=@nif
END
GO
```

- Aplicar Filtros

```
DROP PROCEDURE filterClients
GO
CREATE PROCEDURE filterClients
(
    @nif INT = NULL,
    @nome VARCHAR(256) = NULL,
        @num_cliente INT = NULL,
        @dataNasc Date = NULL,
        @email VARCHAR(64) = NULL,
    @morada VARCHAR(64) = NULL,
        @telemovel INT = NULL
)
AS
BEGIN
        SET NOCOUNT ON;
    SELECT DISTINCT *
    FROM view_clients
    WHERE (@nif IS NULL OR NIF = @nif)
        AND (@nome IS NULL OR Nome LIKE ISNULL(@nome, '') + '%')
        AND (@num_cliente IS NULL OR Num_Cliente = @num_cliente)
        AND (@dataNasc IS NULL OR Data_de_Nascimento >= @dataNasc)
        AND (@email IS NULL OR Email LIKE ISNULL(@email, '') + '%')
        AND (@morada IS NULL OR Morada LIKE ISNULL(@morada, '') + '%')
        AND (@telemovel IS NULL OR ContactoTelefonico = @telemovel)
END
GO
```

- Adicionar elementos a uma tabela, em alguns casos usando **cursor** (exemplo addVenda)

```
GO
CREATE PROCEDURE addVenda
(
	@id_venda INT = NULL,
	@id_prod INT = NULL,
	@Quantidade INT = NULL,
	@data_venda DATE = NULL,
	@nome_vendedor VARCHAR(256) = NULL,
	@id_arm INT = NULL,
	@nif_cliente  INT = NULL,
	@nome_dist VARCHAR(256) = NULL
)
AS
	BEGIN
		DECLARE @countVenda INT;
		DECLARE @counterProd INT;
		DECLARE @counterArm INT;
		DECLARE @num_dist INT;
		DECLARE @num_vendedor INT;
		DECLARE @num_cliente INT;

		DECLARE @erro VARCHAR(100);
		DECLARE @quantity INT;
		DECLARE @IDProduto VARCHAR(5);
		SET @countVenda = Higiliquidos.VerificarIDVendaExistente(@id_venda)
		SET @counterProd = Higiliquidos.checkifProdutoIDExists(@id_prod);
		SET @counterArm = Higiliquidos.checkifArmazemIDExists(@id_arm);
		SET @num_dist = Higiliquidos.getnumDistBYname(@nome_dist);
		SET @num_vendedor = Higiliquidos.getnumVendedorBYname(@nome_vendedor);
		SET @num_cliente = Higiliquidos.getnumClienteByNif(@nif_cliente);
		;
```

```
	IF (@counterProd = 0)
		RAISERROR('O produto com o ID %d não existe.', 16, 1, @id_prod);
	ELSE IF (@counterArm < 1)
		RAISERROR('O armazém com o ID %d não existe.', 16, 1, @id_arm);
	ELSE IF (@num_dist < 1)
		 RAISERROR('O distribuidor com o nome %s não existe.', 16, 1, @nome_dist);
	ELSE IF (@num_vendedor < 1)
		RAISERROR('O vendedor com o nome %s não existe.', 16, 1, @nome_vendedor);
	ELSE IF (@num_cliente < 1)
		RAISERROR('O cliente com o NIF %d não existe.', 16, 1, @nif_cliente);
	ELSE IF (@countVenda >= 1)
		RAISERROR('O ID de venda %d já existe.', 16, 1, @id_venda);
	ELSE
		BEGIN
			DECLARE productCursor CURSOR FAST_FORWARD
				FOR SELECT ID , Quantidade FROM Higiliquidos.Produto WHERE ID = @id_prod;
			OPEN productCursor;
			FETCH productCursor INTO @IDProduto , @quantity;
				WHILE @@FETCH_STATUS = 0
					BEGIN
						IF @quantity >= @Quantidade
							BEGIN
								UPDATE Higiliquidos.Produto SET Quantidade = Quantidade - @Quantidade WHERE     ID = @id_prod
								INSERT INTO Higiliquidos.Venda(ID, Quantidade, ID_Produto, ID_Armazem, Num_Vendedor, Num_Distribuidor, Num_Cliente)
								VALUES (@id_venda, @Quantidade, @id_prod, @id_arm, @num_vendedor, @num_dist, @num_cliente)
								INSERT INTO Higiliquidos.Entrega (Data_Entrega, ID_Venda, Num_Distribuidor) VALUES (@data_venda, @id_venda, @num_dist)
							END
						ELSE
							BEGIN
								RAISERROR ('Produto nao disponivel', 16,1);
							END
						FETCH productCursor INTO @IDProduto , @quantity
					END
			CLOSE productCursor;
			DEALLOCATE productCursor;
			RETURN ;
		END
	END
GO
```

Views

- Foram criadas views, visto que, como o próprio nome indica, ajudam a ter uma visão melhor sobre os dados que estão a ser modelados. Atuam como uma tabela virtual, que não existe fisicamente, pelo que podem ajudar a diminuir a complexidade de certas queries.

- Exemplo de uma view utilizada para ver os produtos:

```
DROP VIEW view_produtos
GO

CREATE VIEW view_produtos AS
SELECT DISTINCT
    p.ID,
        p.Nome,
        p.Marca,
        p.Tipo_de_Produto,
        p.IVA,
        p.Preco,
        p.Quantidade,
        v.ID_Armazem

FROM
    Higiliquidos.Venda v
    JOIN Higiliquidos.Produto p ON v.ID_Produto = p.ID
    JOIN Higiliquidos.Armazem a ON v.ID_Armazem = a.ID
GO
```

- Exemplo de uma view para ver uma venda:

```
DROP VIEW view_vendas
GO

CREATE VIEW view_vendas AS
SELECT DISTINCT
        ent.Data_Entrega,
    v.ID AS ID_Venda,
    v.ID_Produto,
    p.Nome AS Nome_Produto,
    p.Tipo_de_Produto,
    p.Preco,
    p.IVA,
    v.Quantidade,
    v.ID_Armazem,
    pc.Nome AS Nome_Cliente,
        c.NIF_Cliente,
    pv.Nome AS Nome_Vendedor,
    pd.Nome AS Nome_Distribuidor

FROM
    Higiliquidos.Venda v
    JOIN Higiliquidos.Produto p ON v.ID_Produto = p.ID
    JOIN Higiliquidos.Armazem a ON v.ID_Armazem = a.ID
    JOIN Higiliquidos.Cliente c ON v.Num_Cliente = c.Num_Cliente
    JOIN Higiliquidos.Vendedor ve ON v.Num_Vendedor = ve.Num_Vendedor
    JOIN Higiliquidos.Distribuidor d ON v.Num_Distribuidor = d.Num_Distribuidor
    JOIN Higiliquidos.Funcionario fv ON fv.Num_Func = ve.Num_Func
    JOIN Higiliquidos.Funcionario fd ON fd.Num_Func = d.Num_Func
    JOIN Higiliquidos.Pessoa pv ON fv.NIF_Funcionario = pv.NIF
    JOIN Higiliquidos.Pessoa pd ON fd.NIF_Funcionario = pd.NIF
        JOIN Higiliquidos.Pessoa pc ON c.NIF_Cliente = pc.NIF
        JOIN Higiliquidos.Entrega ent ON v.ID = ent.ID_Venda;
GO
```

Triggers

- Os Triggers são representativos de um tipo especial de Stored Procedure, que são apenas executados em determinados eventos associados à manipulação de dados – isto é, quando uma das ações previstas ocorre, os Triggers são ativados. Na nossa base de dados, apenas utilizámos triggers do tipo “after insert, update”, ou seja, eram ativadas, caso ocorresse algum “insert” ou “update” na tabela associada ao trigger.

```
--INSERT PERSON IN PESSOATABLE ---------------------------------
DROP TRIGGER Higiliquidos.addPersona
GO
CREATE TRIGGER Higiliquidos.addPersona ON Higiliquidos.Pessoa
AFTER INSERT, UPDATE
AS
        SET NOCOUNT ON;
        DECLARE @total AS int;
        DECLARE @totalTelemovel AS int;
        DECLARE @NIFPerson AS int;
        DECLARE @numTele AS int;
        DECLARE @dataNasc DATE;
        SELECT @dataNasc = Data_de_Nascimento FROM inserted;
        SELECT @NIFPerson = NIF FROM INSERTED;
        SELECT @numTele = ContactoTelefonico FROM INSERTED;

        IF (@dataNasc IS NOT NULL) AND (TRY_CONVERT(DATE, @dataNasc, 103) IS NULL)
        BEGIN
                RAISERROR('A data de nascimento deve estar no formato dd/mm/aaaa.', 20, 1);
                ROLLBACK;
        END

        IF LEN(@NIFPerson) <> 9
        BEGIN
                RAISERROR('NIF tem de ter 9 números!' , 20, 1);
                ROLLBACK TRAN;
        END

        IF LEN(@numTele) <> 9
        BEGIN
                RAISERROR('Número de Telemovel tem de ter 9 números!' , 20, 1);
                ROLLBACK TRAN;
        END

        SELECT @totalTelemovel = count(*) FROM Higiliquidos.Pessoa WHERE ContactoTelefonico = @numTele
        IF @totalTelemovel > 1
        BEGIN
                RAISERROR('Número de telemovel repetido na base de dados!' , 20, 1);
                ROLLBACK TRAN;
        END

        SELECT @total = count(*) FROM Higiliquidos.Pessoa where NIF = @NIFPerson;
        IF @total > 1
        BEGIN
                RAISERROR('NIF repetido na base de dados!' , 20, 1);
                ROLLBACK TRAN;
        END
GO
```

Índices

- Para melhorar a performance da pesquisa na base de dados foram adicionados 3 índices.

```
-- produtos index
DROP INDEX idx_product_name ON Higiliquidos.Produto;
GO
CREATE INDEX idx_product_name ON Higiliquidos.Produto(ID, Nome);

-- clientes index
DROP INDEX idx_client ON Higiliquidos.Cliente;
GO
CREATE INDEX idx_client ON Higiliquidos.Cliente(Num_Cliente);

-- vendas index
DROP INDEX idx_vendas ON Higiliquidos.Venda;
GO
CREATE INDEX idx_vendas ON Higiliquidos.Venda(ID);
```

UDF

- Para adicionar novas funcionalidades e verificações adicionamos algumas UDFs como por exemplo para:

- Verificação de um certo parâmetro

```
GO
CREATE FUNCTION Higiliquidos.checkIfNIFExists (@nif INT) RETURNS INT
AS
        BEGIN
                DECLARE @counter INT
                SELECT @counter=COUNT(1) FROM Higiliquidos.Pessoa WHERE NIF = @nif
                RETURN @counter
        END
GO
```

- Obtenção de um dado através de outro

```
DROP FUNCTION Higiliquidos.getnumVendedorBYname
GO
CREATE FUNCTION Higiliquidos.getnumVendedorBYname (@nomeVendedor varchar(256))
RETURNS INT
AS
BEGIN
  DECLARE @numVendedor INT;

  SELECT @numVendedor = Num_Vendedor
  FROM Higiliquidos.Vendedor v
  INNER JOIN Higiliquidos.Funcionario f ON v.Num_Func = f.Num_Func
  INNER JOIN Higiliquidos.Pessoa p ON f.NIF_Funcionario = p.NIF
  WHERE p.Nome = @nomeVendedor;

  IF @numVendedor IS NULL
    SET @numVendedor = 0;

  RETURN @numVendedor;
END;
GO
```

Demo

- É possível encontrar na pasta um ficheiro demo.mp4, onde se pode ver de um modo geral o funcionamento de toda a interface. Assim como também existem alguns prints que de uma forma geral demonstram a nossa interface.

Notas

- Em relação ao trabalho apresentado na última aula prática houve uma grande evolução na interface assim como adição de algumas Stored Procedures, Triggers, UDFs e Índices.

- Existe um link para uma pasta do onedrive que contém a submissão do projeto na sua totalidade visto no elearning estar limitado a 50MB.

Conclusão

- Durante a realização deste trabalho tentamos aplicar todos os conhecimentos adquiridos ao longo do semestre quanto à implementação da base de dados, modelação de dados, etc… .

- A ideia inicial teve de sofrer algumas alterações, tendo por fim chegado a uma solução que consideramos correta.

- Em suma, acreditamos que a maioria dos objetivos foi cumprida, e por isso foi possível criar uma base de dados que permite gerir de forma correta o stock e vendas de uma Empresa de Produtos e Equipamentos de Higiene (Higiliquidos).