SISTEMA DE GESTÃO DE UMA CADEIA DE STANDS

BASE DE DADOS

PROF. JOAQUIM SOUSA PINTO
PROF. CARLOS COSTA

P8G4
ROBERTO CASTRO 107133
TIAGO GOMES 108307

Índice

1. Introdução

No âmbito da unidade curricular de Base de Dados, foi-nos proposta a realização deste trabalho prático, cujo tema recai sobre a gestão de uma cadeia de Stands de Automóveis e possui diversas concessionárias espalhadas pelo país.

A base de dados criada incide, portanto, na gestão do stock do stand nas suas concecionárias, clientes, funcionários, compras e vendas, maioritariamente.

O desenvolvimento deste trabalho prático tem como objetivo a aplicação prática dos conhecimentos adquiridos nas aulas teóricas e práticas desta unidade curricular, desde o pensamento e desenho do modelo da base de dados à sua gestão e manipulação por sistemas de software.

2. Análise de Requisitos

Tendo em conta o sistema proposto efetuou-se uma análise de requisitos, de modo a levar todas as parte chaves necessárias para uma implementação eficaz, clara e sucinta.

2.1. Entidades

Entidade – Representa qualquer tipo de pessoa registada no sistema, sendo caracterizada por um NIF, nome, endereço, telefone e e-mail. São entidades o **Cliente**, o **Fornecedor** e o **Funcionário**.

Cliente e Fornecedor – São entidades e caracterizam-se apenas pelo seu NIF.

Funcionário – Um funcionário é uma entidade, caracterizando-se pelo seu NIF, a sua função e o a concessionária onde trabalha.

Stand – Representa as concessionárias que fazem parte da cadeia, possuindo as seguintes informações id, nome, endereço, telefone e email.

Veículo – Um veículo é a representação dos carros reunindo a matrícula, marca, modelo, ano, cilindrada, cor, quilometragem, categoria, preço anunciado, observações e o stand onde se encontra.

Cor – Representação de um dicionário das cores que um veículo pode conter, sendo representada por um id e pela respetiva cor.

Marca – Representação de um dicionário das marcas a que um veículo pode ter, sendo representada por um id e pela respetiva marca.

Função Stand – Representação de um dicionário das funções que um funcionário pode desempenhar, sendo representada por um id e pela respetiva função.

Categoria – Representação de um dicionário das categorias disponíveis para o veículo, sendo representada por um id e pela respetiva categoria.

Combustível – Representação de um dicionário dos combustíveis dos veículos, sendo representada por um id e pelo combustível.

Compra – Representa o ato de adicionar um veículo ao stock de uma concessionária, ou seja a compra de um veículo por parte do stand, tendo um id, o NIF de quem está a vender o carro, o id da concessionária que está a comprar, a matrícula, a respetiva data do registo e o preço de compra.

Venda – Representa o ato de vender um veículo a um cliente, tendo um id, o NIF de quem está a vender o carro (funcionário), o NIF do cliente, o id da concessionária que está a comprar, a matrícula, a respetiva data do registo e o preço de venda.

2.2. Funcionalidades

Convém denotar que este sistema consegue realizar as seguintes operações:

- O sistema permite o registo de stands, carros, clientes, fornecedores, funcionários, marcas, cores, combustíveis, categorias e funções.
- O sistema permite o registo de vendas e compras de carros.
- O sistema permite o gerenciamento de estoque de carros em cada stand.
- O sistema permite filtrar os carros em stock face a determinadas características.
- O sistema permite, de igual forma, as retoma de veículos, sendo esta efetuada de forma direta se for um veículo anteriormente vendido, contudo caso seja um veículo que nunca pertenceu à cadeia de stands a retoma é realizada como um compra.
- O sistema também permite a visualização de todo isto, bem com a atualização de qualquer dado.

3. Conceptualização da base de dados

Durante a elaboração da base de dados e das suas funcionalidades o desenho da mesma sofreu algumas alterações, quer por particularidades que fariam mais sentido, quer também para tornar todo o sistema mais robusto e mais próximo da realidade.

Desta forma, anexamos de seguidas a versão final de todos os diagramas e esquemas contruídos.

3.1. Diagrama ER

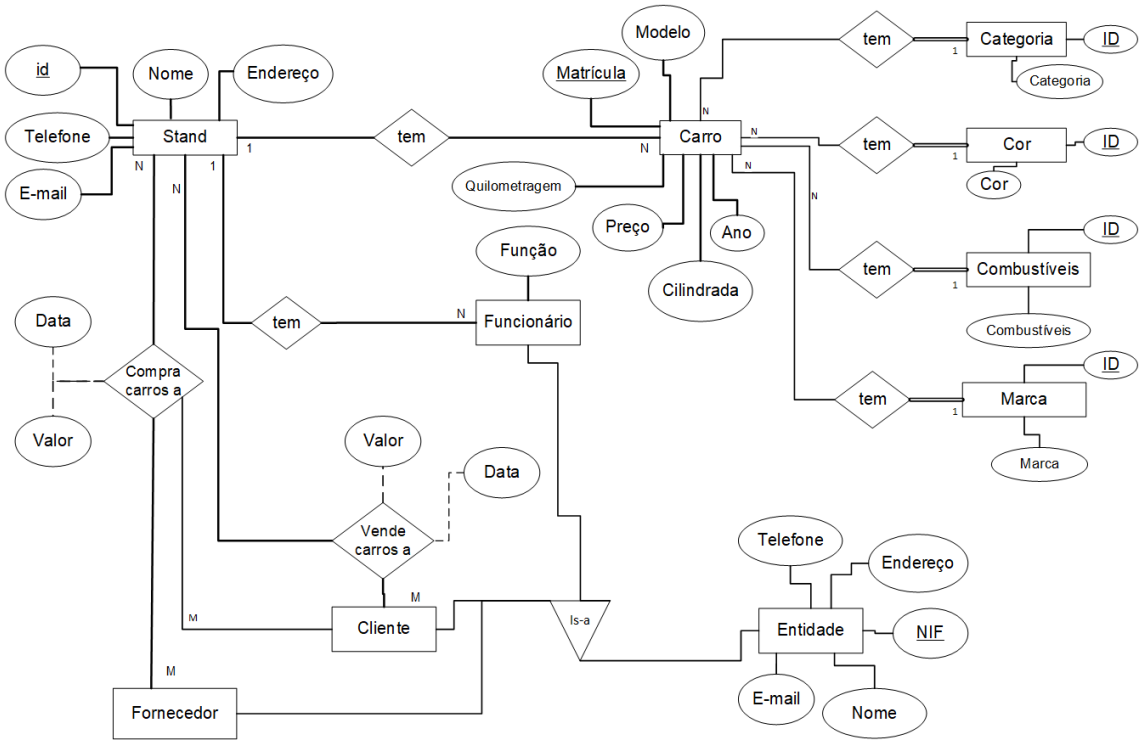

Figura 1 - Diagrama DER

3.2. Esquema Entidade – Relação

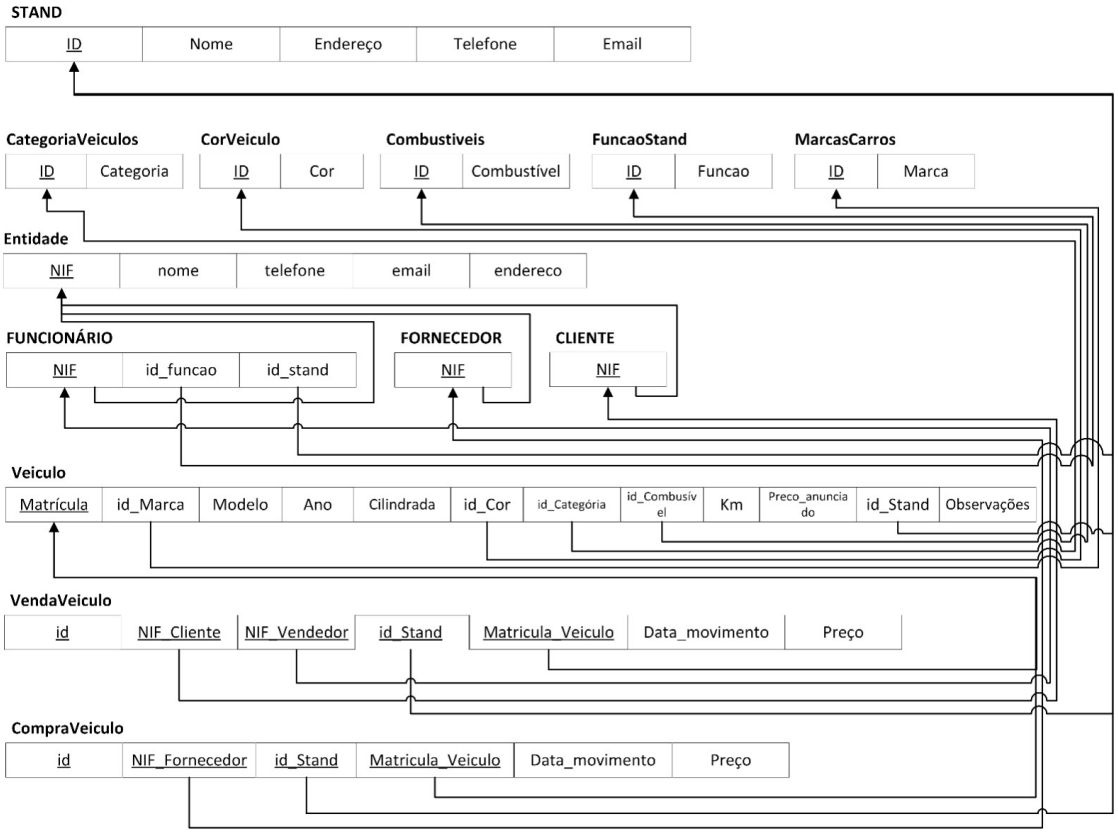

Figura 2 -Esquema ER

3.3. Modelo Relacional

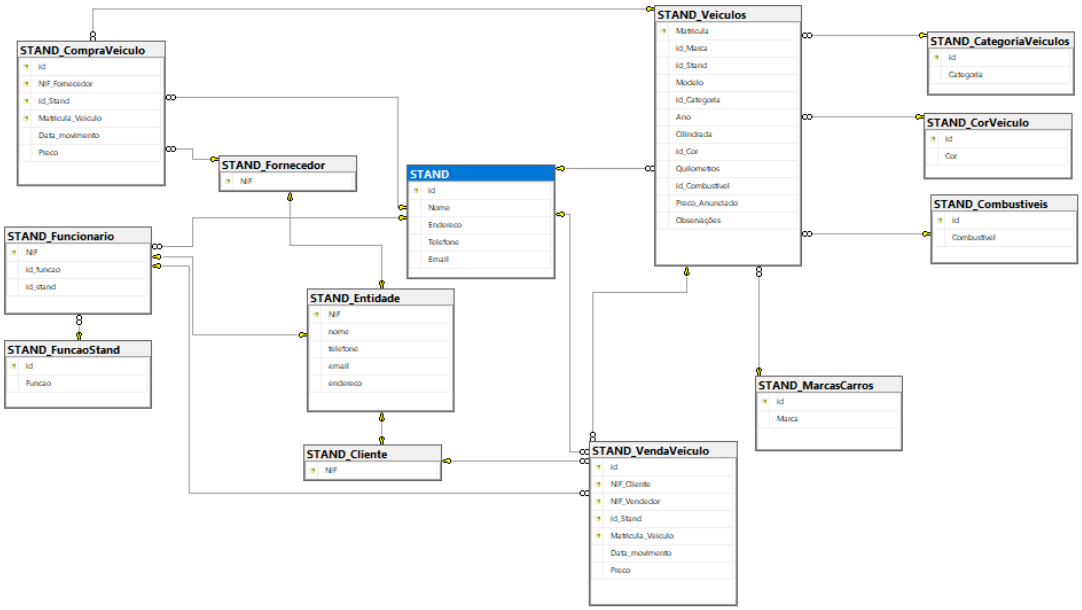

Figura 3 - Diagrama de Base de Dados

Este diagrama representa as ligações entre as entidades, sendo o mesmo gerado pelo SQL Server, após a criação das tabelas na base de dados.

4. Construção da Base de Dados

De modo implementar todo o que foi proposto até agora, a base de dados foi implementada em SQL, linguagem utilizada durante as aulas. O processo de criação da base de dados pode ser dividido em duas etapas distintas. A primeira etapa envolve a criação das tabelas utilizando a Linguagem de Definição de Dados (Data Definition Language). Os detalhes das tabelas podem ser encontrados no arquivo DDL.sql. A segunda etapa é a inserção de dados nas tabelas, utilizando a Linguagem de Manipulação de Dados (Data Manipulation Language). As inserções podem ser encontradas no arquivo DML.sql.

4.1. Criação das Tabelas

As tabelas foram criadas utilizando a Linguagem de Definição de Dados (DDL) em SQL. Cada tabela foi projetada considerando os requisitos e necessidades do sistema, definindo os nomes, tipos de dados, restrições de integridade e relações entre as tabelas. A criação das tabelas estabeleceu uma estrutura adequada para armazenar os dados relevantes de forma organizada e eficiente.

As seguintes figuras são exemplos de tabelas implementadas.

```
CREATE TABLE STAND_Veiculos (
    Matricula VARCHAR(10) PRIMARY KEY,
    id_Marca INT FOREIGN KEY REFERENCES STAND_MarcasCarros(id),
    id_Stand INT FOREIGN KEY REFERENCES STAND(id),
    Modelo VARCHAR(50),
    id_Categoria INT FOREIGN KEY REFERENCES STAND_CategoriaVeiculos(id),
    Ano INT,
    Cilindrada INT,
    id_Cor INT FOREIGN KEY REFERENCES STAND_CorVeiculo(id),
    Quilometros INT,
    id_Combustivel INT FOREIGN KEY REFERENCES STAND_Combustiveis(id),
    Preco_Anunciado DECIMAL(10,2),
    Observações VARCHAR(255)
);
```

Figura 4 - Tabela "Stand_Veiculos"

```
CREATE TABLE STAND_CompraVeiculo (
    id          int             identity(1,1),
    NIF_Fornecedor INT FOREIGN KEY REFERENCES STAND_Entidade(NIF),
    id_Stand INT FOREIGN KEY REFERENCES STAND(id),
    Matricula_Veiculo VARCHAR(10) FOREIGN KEY REFERENCES STAND_Veiculos(Matricula),
    Data_movimento DATE NOT NULL,
    Preco int NOT NULL,
    PRIMARY KEY (id, NIF_Fornecedor,id_Stand, Matricula_Veiculo)

);
```

Figura 5 - Tabela "Stand_CompraVeiculo"

4.2. Inserção de dados

Os dados foram inseridos nas tabelas utilizando a Linguagem de Manipulação de Dados (DML) em SQL. Registos foram adicionados em cada tabela, respeitando a estrutura definida anteriormente. Esses dados são essenciais para o funcionamento e utilização da base de dados e contribuiu para a perceção de como a inserção funcionava.

```
INSERT INTO STAND_FuncaoStand (Funcao)
VALUES
('Gerente de Stand'),
('Vendedor de Automóveis'),
('Consultor Financeiro'),
('Técnico de Manutenção/Serviço'),
('Recepcionista'),
('Gestor de Peças'),
('Gerente de Marketing e Publicidade'),
('Limpeza e Preparação de Veículos');
```

Figura 6 – Preenchimento com dados da Tabela "Stand_FuncaoStand"

```
INSERT INTO STAND_CategoriaVeiculos( Categoria)
VALUES
('Ligeiro'),
('Comercial'),
('Pesado'),
('Motociclo'),
('Quadriciclo Pesado'),
('SUV/4x4'),
('VAN/Minivan')
```

Figura 7 - Preenchimento com dados da Tabela "Stand_CategoriaVeiculos"

5. SQL Programming

5.1. Stored Procedures

As Stored Procedures são rotinas pré-compiladas que podem conter uma ou mais instruções SQL e têm um nome associado. Elas permitem agrupar operações complexas, receber parâmetros de entrada e retornar resultados. São úteis para reutilização de código, melhoria de desempenho e segurança, além de simplificar a lógica de programação no banco de dados.

```
ALTER PROCEDURE STAND_AdicionarCarro ( @matricula VARCHAR(10), @marca VARCHAR(255), @stand VARCHAR(255), @modelo VARCHAR(50), @categoria VARCHAR(255), @ano INT, @cilindrada INT, @cor VARCHAR(255), @km INT,
                    @combustivel VARCHAR(255), @preco_anunciado DECIMAL(10,2), @observações VARCHAR(255), @NIF_fornecedor INT, @data DATE, @preco_compra DECIMAL(10,2) )
AS
    BEGIN
        DECLARE @count INT;
        DECLARE @count2 INT;
        DECLARE @erro VARCHAR(100);
        DECLARE @id_stand INT;
        DECLARE @id_marca INT;
        DECLARE @id_cor INT;
        DECLARE @id_categoria INT;
        DECLARE @id_combustivel INT;

        SET @count = (SELECT dbo.STAND_checkIfFornecedorExists(@NIF_fornecedor))
        SET @count2 += (SELECT dbo.STAND_checkIfMatriculaExists(@matricula))

        IF(@count<1)
            RAISERROR ('NIF introduzido não Existe', 16,1);
            IF(@count>=1)
                RAISERROR ('Matrícula introduzida já Existe', 16,1);
        ELSE
            BEGIN
                BEGIN TRY
                    BEGIN TRAN

                                SET @id_stand = (SELECT dbo.STAND_getIDStandFROMNome(@stand))
                                SET @id_marca = (SELECT dbo.STAND_getIDMarcaFROMNome(@marca))
                                SET @id_cor = (SELECT dbo.STAND_getIDCorFROMNome(@cor))
                                SET @id_categoria = (SELECT dbo.STAND_getIDCategoriaFROMNome(@categoria))
                                SET @id_combustivel = (SELECT dbo.STAND_getIDCombustivelFROMNome(@combustivel))
                                INSERT INTO STAND_Veiculos(Matricula, id_Marca, id_Stand, Modelo, id_Categoria, Ano, Cilindrada, id_Cor, Quilometros, id_Combustivel, Preco_Anunciado, Observações)
                                            VALUES (@matricula, @id_marca, @id_stand, @modelo, @id_categoria, @ano, @cilindrada, @id_cor, @km,@id_combustivel, @preco_anunciado, @observações);
                                INSERT INTO STAND_CompraVeiculo(NIF_Fornecedor,id_Stand, Matricula_Veiculo, Data_movimento, Preco ) VALUES (@NIF_fornecedor,@id_stand,@matricula,@data,@preco_compra)

                    COMMIT TRAN
                END TRY
                BEGIN CATCH
                    Rollback TRAN
                    SELECT @erro = ERROR_MESSAGE();
                    SET @erro =  'O Carro não foi inserido, algum valor inserido incorretamente'
                    RAISERROR (@erro, 16,1);
                END CATCH
            END
    End
GO
```

Figura 8 - Stored Procedure para Adicionar um Carro (Compra por parte do Stand)

```
ALTER PROCEDURE STAND_AdicionarStand ( @nome VARCHAR(255), @email VARCHAR(255),@telefone VARCHAR(9), @endereco VARCHAR(255))
AS
    BEGIN
        DECLARE @count INT;
        DECLARE @erro VARCHAR(100);
        SET @count = (SELECT dbo.STAND_checkIfStandExists(@nome))
        IF(@count>=1)
            RAISERROR ('O Stand introuduzido, já existe', 16,1);
        ELSE
            BEGIN
                BEGIN TRY
                    INSERT INTO STAND(Nome, Endereco, Telefone, EMAIL) VALUES ( @nome, @endereco, @telefone, @email);

                END TRY
                BEGIN CATCH
                    SELECT @erro = ERROR_MESSAGE();
                    SET @erro =  'O Stand não foi inserido, algum valor inserido incorretamente'
                    RAISERROR (@erro, 16,1);
                END CATCH
            END
    End
GO
```

Figura 9 - Stored Procedure para Adicionar uma Concessionária

5.2. UDF's

As UDF's (User-Defined Functions) são funções personalizadas e apresentam os mesmos benefícios que os Stored Procedures, em termos de compilação e otimização de execução.

De um modo geral, as UDF's foram utilizadas para verificar se algum atributo já estava definido em uma tabela específica ou para retornar um valor de uma entidade específica. Elas foram úteis para realizar essas verificações e obter os resultados necessários, principalmente durante a utilização de Stored Procedures.

```
CREATE FUNCTION STAND_getIDFuncaoFROMNome(@funcao VARCHAR(255)) RETURNS INT
AS
    BEGIN
        DECLARE @id INT
        SELECT @id = id FROM STAND_FuncaoSTAND WHERE Funcao = @funcao
        RETURN @id
    END
GO
```

Figura 10 - UDF que retorna o ID da Concessionária (Stand) pelo seu nome

```
CREATE FUNCTION STAND_checkIfFuncionarioExists (@NIF INT) RETURNS INT
AS
    BEGIN
        DECLARE @counter INT
        SELECT @counter=COUNT(1) FROM STAND_Funcionario WHERE NIF=@NIF
        RETURN @counter
    END
GO
```

Figura 11 - UDF que verifica a existência do Funcionário pelo seu NIF

5.3. Triggers

Os triggers são um tipo especial de Stored procedured que é executado automaticamente quando acontece algum evento, como por exemplo, uma inserção, atualização ou eliminação de dados de uma tabela. Ou seja, evento relacionados com à manipulação de dados. Quando ocorre uma das ações definidas no trigger é executado.

Na nossa implementação, utilizamos triggers do tipo "after insert", que vão ser ativados sempre que alguma linha seja adicionada à tabela associada ao trigger.

```
USE [p8g4]
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TRIGGER [dbo].[trg_check_insert_on_STAND]
ON [dbo].[STAND]
AFTER INSERT
AS
BEGIN
  BEGIN TRANSACTION;
  IF EXISTS (SELECT 1 FROM inserted WHERE Nome = '')
  BEGIN
    RAISERROR ('O Nome não pode estar vazio.', 16, 1);
    ROLLBACK TRANSACTION;
    RETURN;
  END
  IF EXISTS (SELECT 1 FROM inserted WHERE Endereco = '')
  BEGIN
    RAISERROR ('O Endereço não pode estar vazio.', 16, 1);
    ROLLBACK TRANSACTION;
    RETURN;
  END
  IF EXISTS (SELECT 1 FROM inserted WHERE LEN(Telefone) <> 9)
  BEGIN
    RAISERROR ('O Telefone deve ter 9 dígitos.', 16, 1);
    ROLLBACK TRANSACTION;
    RETURN;
  END
  IF EXISTS (SELECT 1 FROM inserted WHERE Email NOT LIKE '%_@_%._%')
  BEGIN
    RAISERROR ('O Email deve ter um formato válido.', 16, 1);
    ROLLBACK TRANSACTION;
    RETURN;
  END
  COMMIT TRANSACTION;
END
GO

ALTER TABLE [dbo].[STAND] ENABLE TRIGGER [trg_check_insert_on_STAND]
GO
```

Figura 12 – Trigger que verifica se os parâmetros introduzidos são válidos para serem introduzidos na Tabela Stand

```
USE [p8g4]
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TRIGGER [dbo].[trg_check_nif_exists]
ON [dbo].[STAND_Cliente]
AFTER INSERT
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted i
        LEFT JOIN [dbo].[STAND_Entidade] e ON i.NIF = e.NIF
        WHERE e.NIF IS NULL
    )
    BEGIN
        RAISERROR ('O NIF introduzido não existe na tabela STAND_Entidade.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END
END;
GO

ALTER TABLE [dbo].[STAND_Cliente] ENABLE TRIGGER [trg_check_nif_exists]
GO
```

Figura 13 - Trigger que verifica se o NIF a introduzir na tabela Cliente existe na tabela Entidade

5.4. Views

Uma view é uma representação visual de uma tabela ou de várias tabelas, através de operações em SQL.

No nosso sistema todas as consultas, que a interface faz, são feitas utilizando views. Isso evita que a interface tenha de fazer um query muito extensa.

Uma query mais extensa é a de ver os carros que foram retomados pelo stand, ou seja, os veículos que foram comprados à clientes que tinham comprado esse mesmo veículo no stand. Por isso resolvemos optamos por fazer com que a interface fizesse essa mesma consulta através de uma view.

```
CREATE VIEW [dbo].[STAND_ViewVeiculosRetomados] AS
SELECT c.*
FROM STAND_ViewVeiculosComprados c
LEFT JOIN STAND_Fornecedor f ON c.NIF_Fornecedor = f.NIF
LEFT JOIN STAND_ViewVeiculosVendidos v ON c.Matricula = v.Matricula
WHERE f.nif IS NULL AND v.Matricula IS NULL;
GO
```

Figura 14 - View dos Veículos Retomados pelo Stand

```
CREATE VIEW STAND_ViewTodosVeiculos AS
SELECT
STAND_Veiculos.Matricula, STAND_MarcasCarros.Marca, STAND_Veiculos.Modelo, STAND_CorVeiculo.Cor ,
STAND_Veiculos.Cilindrada, STAND_Veiculos.Ano, STAND_Veiculos.Quilometros, STAND_Combustiveis.Combustivel,
STAND_CategoriaVeiculos.Categoria, STAND_Veiculos.Preco_Anunciado,STAND.Nome, STAND_Veiculos.Observações
FROM STAND_Veiculos
INNER JOIN STAND_MarcasCarros ON STAND_MarcasCarros.id = STAND_Veiculos.id_Marca
INNER JOIN STAND ON STAND.id = STAND_Veiculos.id_Stand
INNER JOIN STAND_CategoriaVeiculos ON STAND_CategoriaVeiculos.id = STAND_Veiculos.id_Categoria
INNER JOIN STAND_CorVeiculo ON STAND_CorVeiculo.id = STAND_Veiculos.id_Cor
INNER JOIN STAND_Combustiveis ON STAND_Combustiveis.id = STAND_Veiculos.id_Combustivel
```

Figura 15 - View dos todos os Veículos do Stand

5.5. Indexes

Optamos pela utilização de indexes na nossa base de dados, mesma ela sendo pequenas dimensões. Fizemos o seu uso nas tabelas *STAND* quando se pesquisa pelo *id* do stand, *STAND_Entidade* quando se pesquisa pelo *NIF* de um cliente, funcionário ou fornecedor, e *STAND_Veiuclos* quando se pesquisa pela matrícula do veículo. Os indexes foram implementados nestas tabelas devido a serem as tabelas de maior dimensão.

```
CREATE INDEX idx_Stand_id ON STAND (id)
GO

CREATE INDEX idx_Entidade_NIF ON STAND_Entidade (NIF)
GO

CREATE INDEX idx_Veiculos_Matricula ON STAND_Veiculos (Matricula)
GO
```

Figura 16 - Indexes criados para a nossa base de dados

6. Interface Gráfica

A interface gráfica do aplicativo foi desenvolvida no Visual Studio, utilizando a plataforma Windows Forms. Todo o código foi escrito em C# e uso do Windows Forms permitiu criar uma interface visual interativa e amigável para os utilizadores.

Para exibir os dados armazenados de forma clara e organizada, utilizamos a estrutura DataGridView. Este componente permitiu-nos apresentar as informações em colunas, facilitando a visualização e análise dos dados. Além disso, implementamos recursos de ordenação, o que significa que os utilizadores podem classificar as informações por atributos específicos, tornando mais fácil encontrar dados relevantes ou realizar pesquisas específicas.

Cada formulário possui um layout semelhante. No topo do formulário, adicionamos uma caixa de pesquisa que permite aos usuários filtrar os dados com base em uma determinada categoria. Isso foi implementado usando uma ComboBox, onde os utilizadores podem selecionar a categoria desejada para a pesquisa. Logo abaixo da caixa de pesquisa, encontra-se a DataGridView que exibe as informações relacionadas ao formulário em questão.

Na parte inferior do formulário, incluímos campos onde se pode inserir os novos dados ou efetuar alterações das informações existentes. Estes campos variam dependendo do tipo de dados que estão sendo manipulados.

Para melhorar a experiência do utilizador, implementamos uma verificação de erros durante as operações de inserção e atualização de dados na interface gráfica. Se ocorrer algum problema ou anomalia durante essas ações, uma MessageBox será exibida, informando sobre o ocorrido e fornecendo detalhes relevantes. Por outro lado, se a ação for executada com sucesso, como a inserção bem-sucedida de um novo veículo, uma MessageBox será exibida para advertir o utilizador sobre o sucesso da operação.

As imagens a seguir fornecem uma visão geral da interface gráfica, permitindo aos utilizadores visualizar e interagir com os dados de maneira intuitiva e eficiente.

Figura 17 14 - Menu inicial

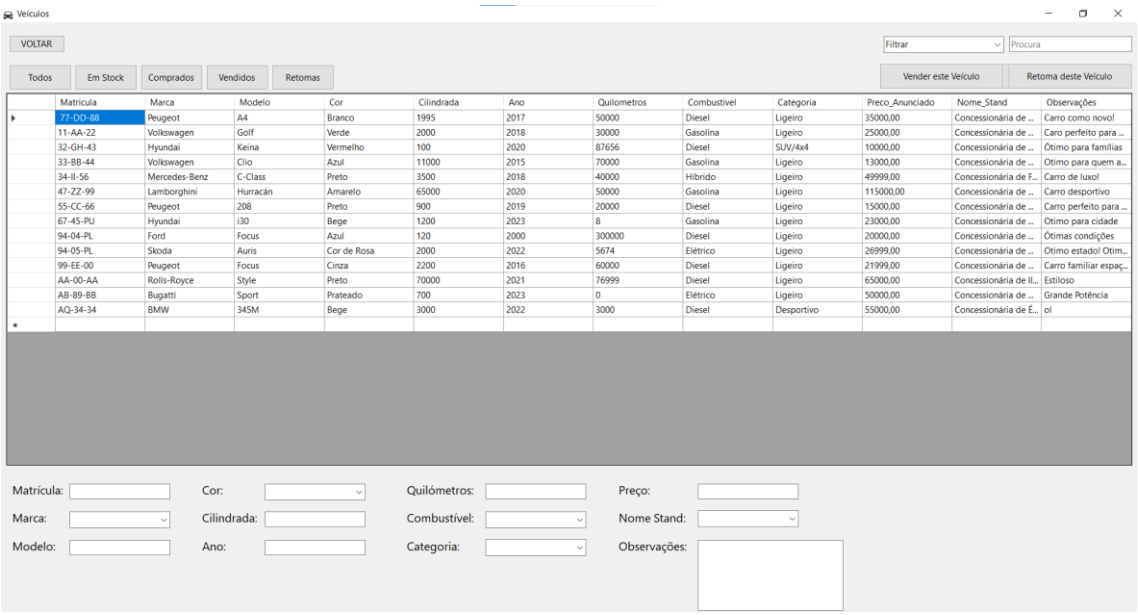

Figura 18 15 - Form dos Veículos

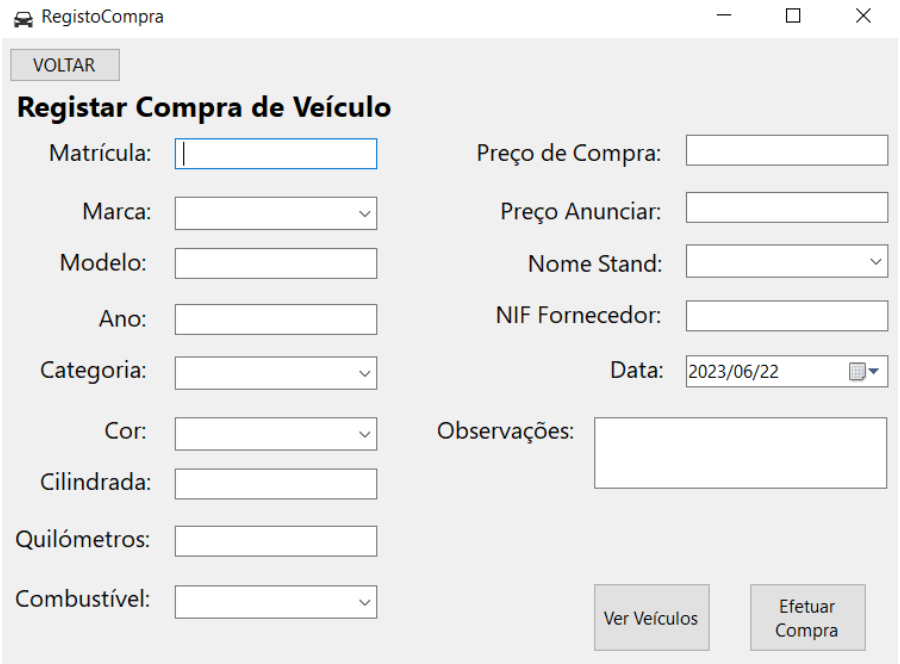

Figura 19 - 16Form da Compra de Veículos (Adicionar um Veículo)

7. Segurança

Para minimizar possíveis vulnerabilidades da base de dados a ataques de SQL Injection, na implementação foram considerados os seguintes pontos de precaução:

1. Verificações dos dados inseridos pelos utilizadores: Implementamos verificações através de triggers para validar os dados inseridos. Isto ajuda a garantir que apenas dados válidos e seguros sejam aceites e armazenados.

2. Utilização de SQL Parametrizado ou Stored Procedures: Priorizamos o uso de SQL Parametrizado ou Stored Procedures em vez de recorrer ao SQL Dinâmico. Essas abordagens ajudam a evitar a inserção direta de valores inválidos, reduzindo o risco de SQL Injection.

3. Mensagens de erro compreensíveis: Fornecemos mensagens de erro adequadas aos, de modo que se entenda claramente quando introduzem valores ou informações incorretas. Essas mensagens auxiliam os utilizadores a corrigir possíveis erros e garantir a integridade dos dados.

Ao considerar esses pontos, procuramos fortalecer a segurança da base de dados e reduzir as possibilidades de exploração de vulnerabilidades.

8. Demonstração

No Link em anexo, é possível observar uma pequena demonstração das principais funcionalidades da interface gráfica, que foram descritas anteriormente.

Link:

https://drive.google.com/file/d/1eP1B-62MWpv3pRfCGBgAtfOByWayNFHF/view?usp=sharing

9. Conclusão

Em suma, o trabalho de desenvolvimento da base de dados para a gestão de uma cadeia de Stands de Automóveis foi realizado com sucesso, aplicando os conhecimentos adquiridos na unidade curricular de Base de Dados.

A análise de requisitos permitiu identificar as entidades e funcionalidades essenciais para o sistema. A construção da base de dados envolveu a criação das tabelas, seguida pela inserção de dados relevantes. Durante o processo, foram utilizadas Stored Procedures e UDF's para facilitar operações complexas e obter resultados específicos.

Medidas de segurança foram implementadas para prevenir ataques de SQL Injection, como a verificação dos dados inseridos pelos utilizadores e a utilização de SQL Parametrizado ou Stored Procedures. Além disso, foram fornecidas mensagens de erro compreensíveis para ajudar os utilizadores a corrigirem informações incorretas.

A base de dados foi integrada a uma interface gráfica para demonstrar as funcionalidades do sistema.

No geral, o trabalho permitiu a aplicação prática dos conhecimentos teóricos e práticos adquiridos, além de contribuir para a compreensão da importância da eficiência na gestão de bases de dados.

10. Bibliografia

- Slides disponibilizados na página do eLearning@UA da Unidade Curricular.
- https://www.w3schools.com/sql/
- https://docs.microsoft.com/en-us/dotnet/csharp/