universidade de aveiro

Departamento de Eletrónica, Telecomunicações e Informática

Mestrado Integrado em Engenharia de Computadores e Telemática

Projeto de Base de Dados

2018/2019

Gestão de Stands e Oficinas

P1G10

João Coelho, 80335

Marco Silva, 84770

Introdução

No âmbito da unidade curricular de base de dados, foi proposta a realização de um projeto no qual se integrassem os conhecimentos adquiridos ao longo da mesma com uma aplicação em C#/WPF. Desta forma, desenvolvemos um gestor de stands e oficinas que visa facilitar o processo de gestão de contas de clientes, processos de venda e reparação e a organização de stock em C#.

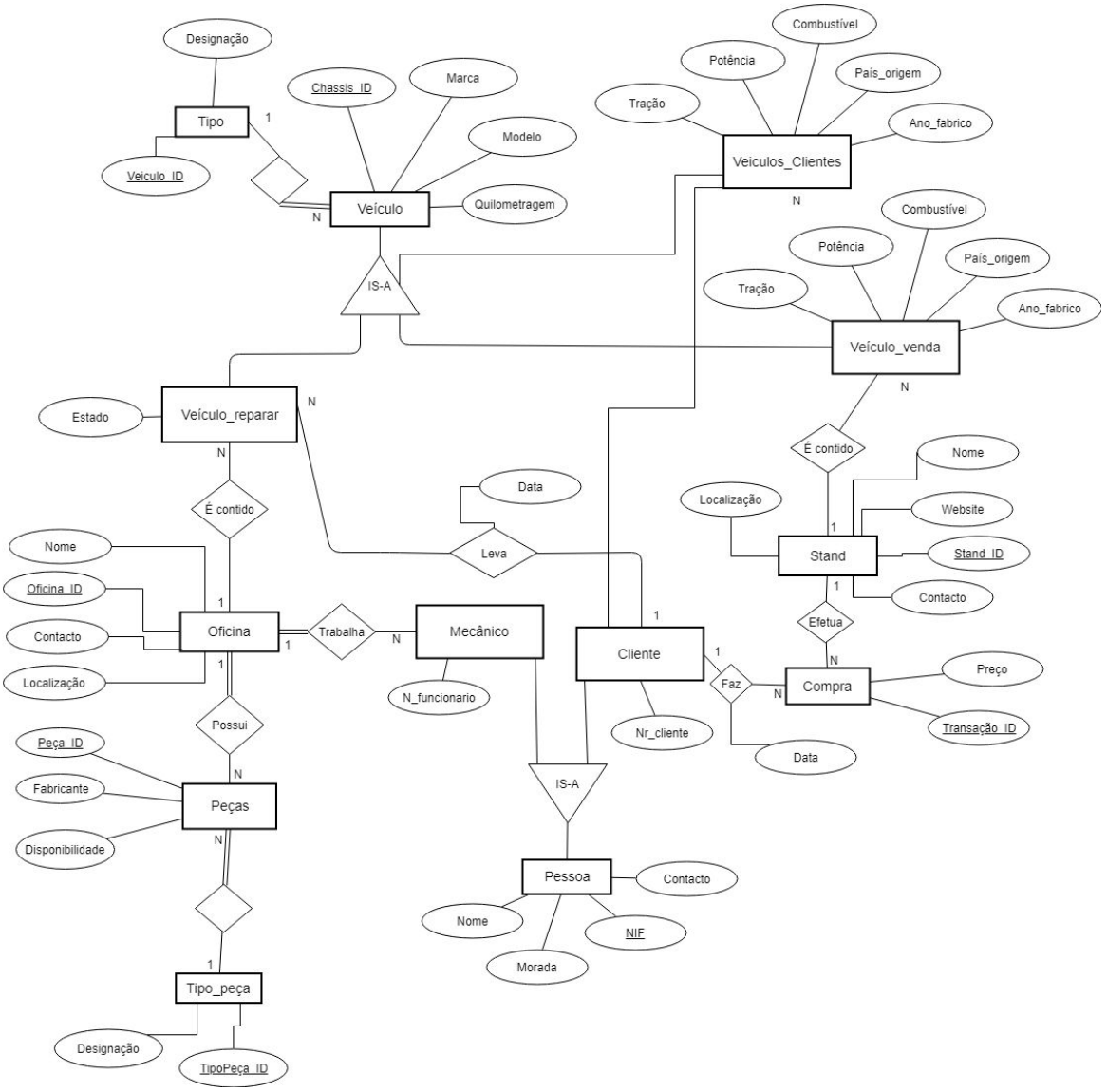
Análise de Requisitos

Sendo este projeto a modulação de um sistema de gestão de um stand automóvel e também de uma oficina que possui várias peças em stock e carros prontos a ser reparados e entregues.

Desta forma, definimos que os requisitos necessários para a nossa aplicação seriam:

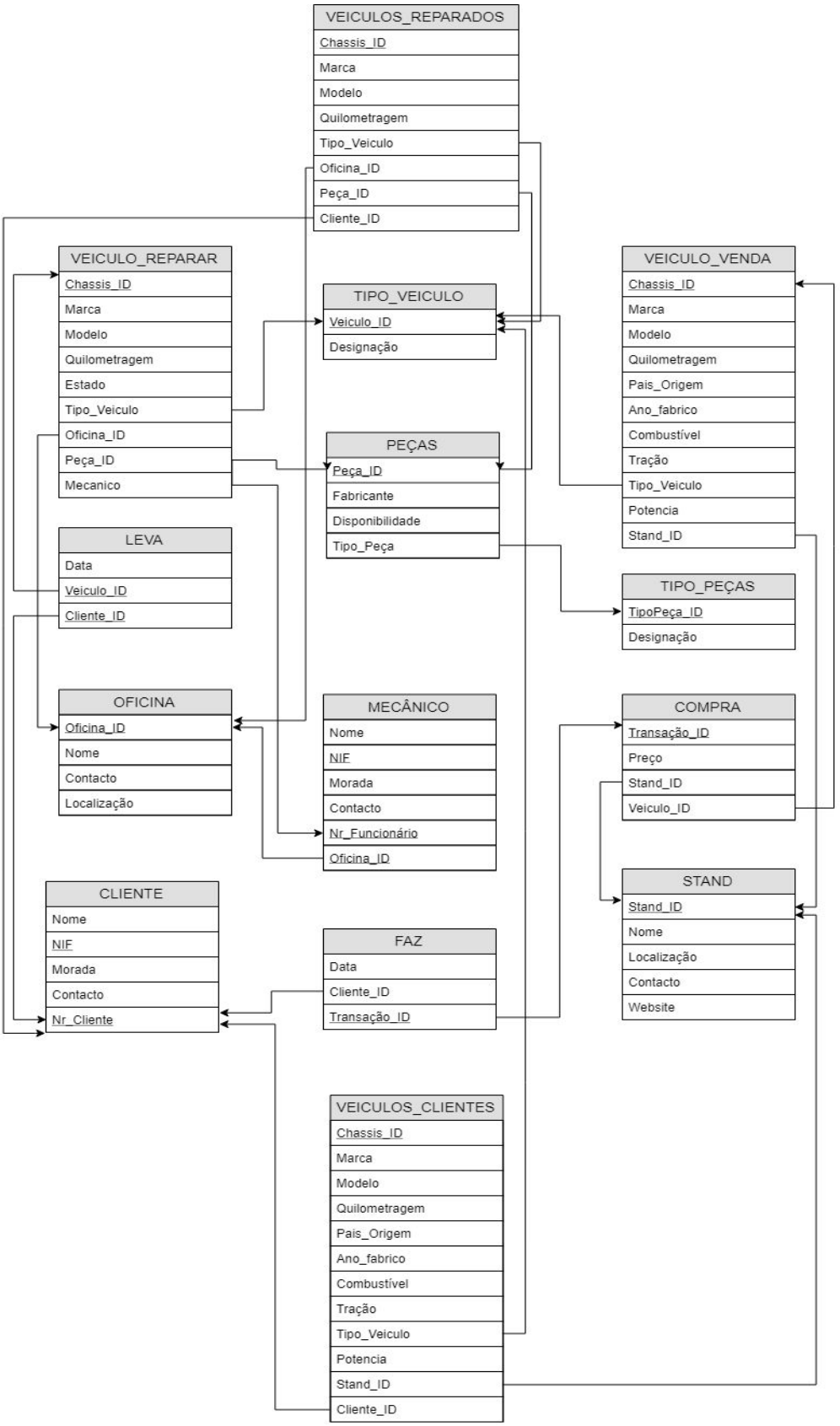
- Adição/Remoção/Edição de mecânicos;
- Adição/Remoção/Edição de clientes;
- Adição/Remoção/Edição/Consulta de veículos a reparar;
- Adição/Remoção/Edição/Consulta de veículos à venda;
- Adição/Remoção/Edição/Consulta de peças;
- “Login” como cliente podendo comprar carros ou reparar os seus próprios carros;
- “Login” como mecânico podendo reparar ou entregar carros da oficina.

DER



- Uma oficina pode ter uma ou mais peças, um ou mais mecânicos e um ou mais veículos para reparar, daí as várias relações (1:N).
- Um stand pode ter mais que uma Compra, esta sendo constituída obrigatoriamente por um cliente. O stand pode também ter um ou mais veículos à venda (1:N).
- Um cliente pode ter associado um ou mais veículos, podendo estes não ser nem para reparar nem para venda.

Modelo Relacional



- Uma pessoa pode ser um cliente do stand que pode comprar um ou mais veículos, ou pode ser um mecânico que trabalha na oficina. Cada pessoa é caracterizada pelo nome, morada, NIF e contacto. O cliente tem ainda um número de cliente e o mecânico tem um número de funcionário.
- Um veículo é caracterizado por um Chassis_ID, marca, modelo, quilometragem, combustível e um tipo que se divide em três categorias: ligeiro, motociclo e pesado (de passageiros ou mercadorias)
- Um veículo para venda é uma entidade fraca de veículo que, para além dos atributos do veículo, tem um país de origem, ano de fabrico, potência, tração e um ID do stand em que se insere.
- Um veículo para reparar, tal como o veículo para venda, é uma entidade fraca de veículo e tem um ID para o identificar na oficina, um ID da peça necessária e um estado (por reparar ou reparado)
- O stand é composto por veículos para venda e é onde são efetuadas as compras, caracterizadas por um ID de transação, um preço e um número de cliente.
- A oficina é composta por veículos para reparar e por peças existentes em stock.
- As peças são compostas por um ID, nome, fabricante, disponibilidade e um tipo que as divide em três categorias: motor, suspensão e eletrónica.

Estrutura da implementação

Optámos por implementar o nosso projeto baseado no uso de Stored Procedures e User Defined Functions. A utilização de Índices não foi necessária visto que tratamos todos os elementos das tabelas através das suas primary keys.

Em termos mais específicos, para todas as ações diretas de inserção, remoção e edição de dados utilizamos Stored Procedures por questões não só de segurança e prevenção a ataques, mas também por serem parametrizáveis. Por outro lado, para todas as ações de visualização de dados utilizamos User Defined Functions. Num primeiro desenvolvimento, tínhamos criado views, mas depois, decidimos optar pelas UDFs devido à sua performance e por aceitarem parâmetros.

Utilizamos ainda dois triggers para executar algumas modificações.

Deployment

Para além dos ficheiros com todos os Stored Procedures, User Defined Functions, Triggers e Views criadas, temos também os ficheiros com a criação da base de dados e alguns inserts.

Índices

Visto que não temos uma quantidade excessiva de dados na nossa base de dados, não priorizamos a utilização de índices. Após o desenvolvimento dos SPs e das UDFs ainda ponderamos a utilização de índices, no entanto chegamos a conclusão que seriam desnecessários.

User Defined Functions

```
-- Retorna todos os veículos do cliente correspondente ao ID dado
CREATE FUNCTION STAND.ClientVehicles(@Cliente INT) RETURNS TABLE AS
    RETURN(
        SELECT Chassis_ID, Marca, Modelo, Tipo_Veiculo
        FROM STAND.VEICULOS_CLIENTES WHERE Cliente_ID = @Cliente
    );
GO

-- Retorna todos os veículos de todos os clientes
CREATE FUNCTION STAND.AllClientVehicles() RETURNS TABLE AS
    RETURN(
        SELECT Chassis_ID, Marca, Modelo, Tipo_Veiculo, Cliente_ID
        FROM STAND.VEICULOS_CLIENTES
    );
GO

-- Retorna todos os veículos de um certo tipo (Recebe o ID do tipo como argumento)
CREATE FUNCTION STAND.VehicleByType(@Veiculo INT) RETURNS TABLE AS
    RETURN(
        SELECT 'Para Reparar' AS Type, Chassis_ID,Marca,Modelo FROM STAND.VEICULO_REPARAR WHERE Tipo_Veiculo = @Veiculo
        UNION
        SELECT 'Para Venda' AS Type, Chassis_ID,Marca,Modelo FROM STAND.VEICULO_VENDA WHERE Tipo_Veiculo = @Veiculo
        UNION
        SELECT 'Dos Clientes' AS Type, Chassis_ID,Marca,Modelo FROM STAND.VEICULOS_CLIENTES WHERE Tipo_Veiculo = @Veiculo
        UNION
        SELECT 'Repararados' AS Type, Chassis_ID,Marca,Modelo FROM STAND.VEICULOS_REPARADOS WHERE Tipo_Veiculo = @Veiculo
    );
GO
```

O exemplo dado mostra três das udfs que desenvolvemos. Na primeira retornamos os veículos associados a um cliente dado o seu número de cliente, na segunda é devolvida uma tabela com todos os veículos de todos os clientes e, por último, retorna-se uma tabela com todos os veículo de um dado tipo. Além destas ainda temos as seguintes udfs:

- STAND.AllClients() -> Retorna todos os clientes dos stands ou oficinas
- STAND.AllGarageVehicles(); -> Retorna todos os veículos das Oficinas
- STAND.AllNewVehicles(); -> Retorna todos os veículos novos nos Stands
- STAND.AllParts(); -> Retorna todas as peças (disponíveis/indisponíveis)
- STAND.AllStandVehicles(); -> Retorna todos os veículos dos Stands (novos/usados)
- STAND.AllUsedVehicles(); -> Retorna todos os veículos usados nos Stands
- STAND.AllVehicles(); -> Retorna todos os veículos nos Stands
- STAND.AvailableParts(); -> Retorna todas as peças disponíveis
- STAND.ClientVehicles(Nr_Cliente); -> Retorna todos os veículos de um cliente
- STAND.AllMechanics(); -> Retorna todos os mecânicos

- STAND.OccupiedMechanics(); -> Retorna todos os mecânicos ocupados
- STAND.AvailableMechanics(); -> Retorna todos os mecânicos disponíveis
- STAND.ToRepairVehicles(); -> Retorna todos os veículos por reparar
- STAND.RepairedVehicles(); -> Retorna todos os veículos que já foram reparados
- STAND.UnavailableParts(); -> Retorna todas as peças indisponíveis
- STAND.VehicleByType(Tipo_Veiculo); -> Retorna todos os veículos de uma categoria
- STAND.AllClientVehicles(); -> Retorna todos os veículos de todos os clientes
- STAND.ToRepairNoMechanic(); -> Retorna os veículos por reparar sem mecanico
- STAND.ToRepairByMec(Nr_Funcionario); -> Retorna os veículos que um mecânico tem associados
- STAND.RepairedOfCliente(Nr_Cliente); -> Retorna os veículos que já foram reparados de um cliente

Stored Procedures

```
--Adicionar cliente
CREATE PROCEDURE sp_addClient @Nome VARCHAR(40), @NIF CHAR(9), @Morada VARCHAR(30), @Contacto CHAR(9) AS
    IF @NIF is null
    BEGIN
        PRINT 'O NIF não pode ser vazio.'
        RETURN
    END

    DECLARE @nifClient AS TINYINT
    SET @nifClient = (SELECT COUNT(STAND.CLIENTE.NIF) AS nifClient FROM STAND.CLIENTE WHERE STAND.CLIENTE.NIF=@NIF)

    DECLARE @nifMechanic AS TINYINT
    SET @nifMechanic = (SELECT COUNT(STAND.MECANICO.NIF) AS nifMechanic FROM STAND.MECANICO WHERE STAND.MECANICO.NIF=@NIF)

    IF (@nifMechanic != 0)
    BEGIN
        PRINT 'Este NIF esta associado a um funcionário'
        RETURN
    END

    IF (@nifClient = 0)
    BEGIN
        IF @Nome is null
        BEGIN
            PRINT 'O Nome não pode estar vazio'
            RETURN
        END

        IF @Contacto is null
        BEGIN
            PRINT 'O Contacto não pode estar vazio'
            RETURN
        END

        INSERT INTO STAND.CLIENTE(Nome, NIF, Morada, Contacto)
        VALUES (@Nome, @NIF, @Morada, @Contacto);
    END
    ELSE
        PRINT 'O cliente associado a este NIF ja existe'
GO
```

O exemplo dado é o código ao qual está associada a inserção de um cliente. Inicialmente é verificada se o NIF (primary key) é NULL, caso seja, não se pode inserir esse cliente. Para além disto, é feita uma verificação dum conjunto de fatores antes da sua criação, sendo alguns deles verificar se o NIF já está associado a um funcionário, e se o nome ou o contacto estão vazios. Se estas condições não se verificarem, a inserção é executada.

sp_addClient - Adicionar cliente.

sp_addGarage - Adicionar uma oficina
sp_addMechanic - Adicionar mecânico
sp_addPart - Adicionar uma peça ao stock
sp_addPartType - Adicionar tipo de peça
sp_addStand - Adicionar um stand
sp_addVehicleToRepair - Adicionar veiculo para reparar
sp_addVehicleToSell - Adicionar veiculo para vender
sp_addVehicleType - Adicionar tipo de veiculo
sp_buyVehicle - Cliente compra veiculo
sp_deleteClient - Apagar cliente
sp_deleteGarage - Apagar oficina
sp_deleteMechanic - Apagar mecanico
sp_deletePart - Apagar Peça
sp_deleteStand -Apagar stand
sp_deleteRepaired - Apagar veiculo reparado
sp_deleteToRepair - Apagar veiculo para reparar
sp_deleteToSell - Apagar veiculo para venda
sp_deleteVeiculoCliente - Apagar veiculo do cliente
sp_deliverVehicle - Entregar o veiculo ao cliente (isto é, mudar para VEICULOS_REPARADOS)
sp_MechanicToGarage - Atribuir um mecanico a uma garagem (Oficina_ID é atribuido a um tuplo da tabela MECANICO)
sp_repairVehicle - Reparar um veiculo
sp_VehicleToMechanic - Atribuir um mecanico a um veiculo (Mecanico e Oficina_ID atribuidos a um tuplo da tabela VEICULO_REPARAR)
sp_updateStock - Atualizar o stock de peças
sp_updateClient - Atualizar a info de um cliente
sp_updateMechanic - Atualizar a info de um mecanico

Triggers

```
CREATE TRIGGER trg_updateTransaction on STAND.FAZ AFTER INSERT AS
BEGIN
    UPDATE STAND.FAZ SET Data_Compra = GETDATE() FROM STAND.FAZ f JOIN Inserted i ON f.Transacao = i.Transacao;
END;
GO

CREATE TRIGGER trg_updateVehicleToMechanic on STAND.LEVA AFTER INSERT AS
BEGIN
    UPDATE STAND.LEVA SET Data_Entrega = GETDATE() FROM STAND.LEVA l JOIN Inserted i ON l.Veiculo_ID = i.Veiculo_ID;
END;
```

Na imagem acima é possível ver os triggers que usamos no nosso projeto. Apenas criámos estes dois pois não achamos necessidade em criar mais nenhum. Ambos são executados após uma inserção, na tabela STAND.FAZ no caso do primeiro e na tabela STAND.LEVA no caso do segundo. O objetivo destes triggers é preencher o campo da data, em ambas as tabelas, que é inserida a “NULL” no momento da inserção dos restantes valores e deste modo evitar que o utilizador da aplicação tenha que adicionar a mesma.

SQL DDL

Todas as tabelas foram criadas com código SQL DDL presente no ficheiro "Create_Stand.sql". Criámos 14 tabelas e de seguida é apresentado um trecho de código que serviu para a criação da tabela STAND.VEICULOS_CLIENTES correspondente aos veículos associados a todos os clientes.

```
CREATE TABLE STAND.VEICULOS_CLIENTES(
    Chassis_ID      INT NOT NULL,
    Marca           VARCHAR(15) NOT NULL,
    Modelo          VARCHAR(30) NOT NULL,
    Quilometragem   INT,
    Pais_Origem     VARCHAR(30),
    Ano_Fabrico     CHAR(4),
    Combustivel     VARCHAR(10) NOT NULL,
    Tracao          VARCHAR(10),
    Tipo_Veiculo    INT NOT NULL,
    Potencia        INT,
    Stand_ID        INT NOT NULL,
    Cliente_ID      INT NOT NULL,
    PRIMARY KEY(Chassis_ID),
    FOREIGN KEY(Tipo_Veiculo) REFERENCES STAND.TIPO_VEICULO(Veiculo_ID)
    ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY(Stand_ID) REFERENCES STAND.STAND(Stand_ID)
    ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY(Cliente_ID) REFERENCES STAND.CLIENTE(Nr_Cliente)
    ON DELETE CASCADE ON UPDATE CASCADE
);
```

SQL DML

Os inserts, deletes e updates são todos feitos através dos Stored Procedures. Apenas os inserts iniciais para encher a base de dados, presentes no ficheiro "Insert_Stand.sql", foram feitos através de SQL DML. Na aplicação feita em visual studio são apenas chamados os SPs para manuseamento dos dados.

Exemplo de utilização de um SP:

```
private void bttnEntregar_Click(object sender, EventArgs e)
{
    cn.Open();

    current = listBox1.SelectedIndex;
    if (current < 0)
    {
        MessageBox.Show("Please select a car to be delivered");
        return;
    }

    SqlCommand cmd = new SqlCommand();
    cmd.CommandText = "exec sp_deliverVehicle @Chassis_ID";

    cmd.Connection = cn;
    cmd.Parameters.Clear();
    cmd.Parameters.AddWithValue("@Chassis_ID", ((VeiculoReparar)listBox1.Items[current]).ChassisId);

    try
    {
        cmd.ExecuteNonQuery();
    }
    catch (Exception ex)
    {
        throw new Exception("Failed to update in database. \n ERROR MESSAGE: \n" + ex.Message);
    }
    finally
    {
        cn.Close();
    }
    MessageBox.Show("Carro entregue com sucesso!");
    listBox1.Items.Clear();
    ShowEntregar();
}
```

Informação Extra

A segurança da nossa base de dados é ínfima. Não foi colocada proteção contra possíveis códigos de intenção maliciosa. Não foi implementada segurança realista por se ter dado prioridade ao funcionamento do projeto a tempo de entrega.

O projeto em Visual Studio não está 100% adaptado à possível tentativa de crash por parte de inserts mal formatados mas tem alguns error handlings que voltam a pedir para colocar a informação de maneira correta sem crashar o programa e condições que apenas permitem serem visualizadas de acordo com a sua função.

Apesar de tudo isto, se todos os inserts estiverem corretamente formatados, o sistema deverá funcionar sem qualquer problema e a sua utilização não seria de todo obsoleta.

A maior complexidade está nas funções utilizadas na zona de Clientes e Mecânicos que inclui alguns stored procedures e UDFs.