Relatório

Breve introdução

Neste relatório, iremos abordar os conceitos e objetivos do nosso projeto, bem como as suas funcionalidades e a forma como foram construídas.

O tema escolhido pelo nosso grupo foi um conjunto de oficinas de automóveis, que possui vários funcionários e um gerente por cada oficina. Também há veículos que possuem dono em que é possível efetuar um serviço (reparação ou revisão) que pode precisar de peças.

Análise de requisitos

Após um levantamento detalhado de informação essencial associada ao problema, chegamos às seguintes entidades e atributos:

Pessoa (Nome, Apelido, CC, Endereço, Telefone)

Cliente (Parceiro, atributos de Pessoa)

Funcionário (Salário, Oficina, atributos de Pessoa)

Gerente (Bónus, atributos de Funcionário)

Mecânico (Especialidade, atributos de Funcionário)

Oficina (ID, Localização, Telefone, Email, Gerente)

Veículo (Tipo Veículo, Marca, Matrícula, Dono, Oficina, Data Entrada, Data Saída)

Peça (Nome, Preço, Referência, Número em stock)

Serviço (ID, Preço, Oficina, Veiculo, Preço final, Data início, Data conclusão)

Revisão (Tipo, atributos de Serviço)

Reparação (Peça, atributos de Serviço)

Efetua(Serviço, Mecânico)

Necessita(Peça, Reparação, Número de Peças)

Tipo Veiculo(ID, Nome)

Com as seguintes relações:

Cliente é uma Pessoa; Funcionário é uma Pessoa; Gerente é um Funcionário; Mecânico é um Funcionário; Revisão é um Serviço; Reparação é um Serviço; Mecânico efetua Serviço; Funcionário trabalha para Oficina; Gerente gere Oficina; Oficina possui Cliente; Cliente tem Veículo; Veículo localizado em Oficina; Serviço efetuado em Veículo; Serviço ocorre Oficina; Reparação precisa de Peça.

Dando origem à seguinte tabela de chaves:

Entidade	Chave Primária	Chave Candidata	Chave Estrangeira
Pessoa	NIF	Telefone, endereço	----------
Cliente	NIF	NIF	NIF
Funcionário	NIF	NIF	NIF, Oficina
Gerente	NIF	NIF	NIF
Mecânico	NIF	NIF	NIF
Oficina	ID	Número, telefone, email	----------
Veículo	Matrícula	Matrícula	Dono, Oficina
Peça	Referência	Referência, Nome	----------
Serviço	ID	ID	Oficina, Veículo
Revisão	ID	ID	ID
Reparação	ID	ID	ID, Peça
Peça	Número de série	Número de série, nome	Veículo
Efetuação-Serviço	Veículo, Serviço	Veículo, Serviço	Veículo, Serviço

Diagrama Entidade/Relação

Em relação ao diagrama entidade relação, após várias versões chegamos ao seguinte diagrama:

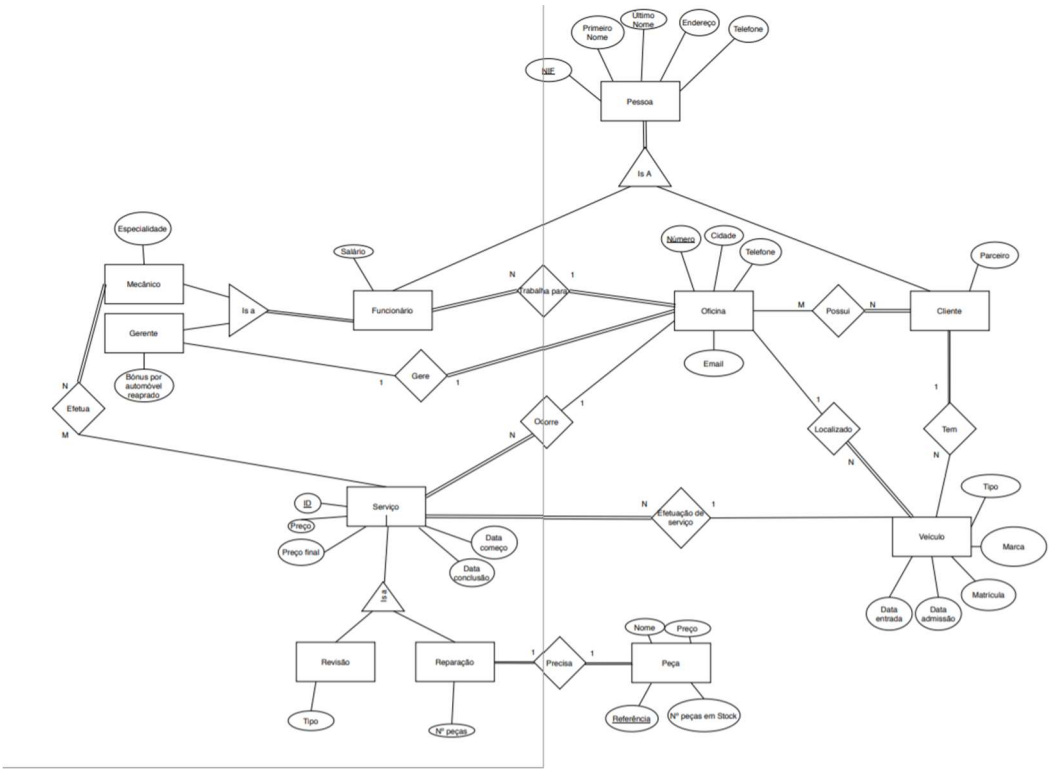

Figura 1 - Diagrama Entidade/Relação

Esquema relacional da BD

Quanto ao esquema relacional, após fazermos uma conversão do diagrama entidade/relação obtivemos o seguinte:

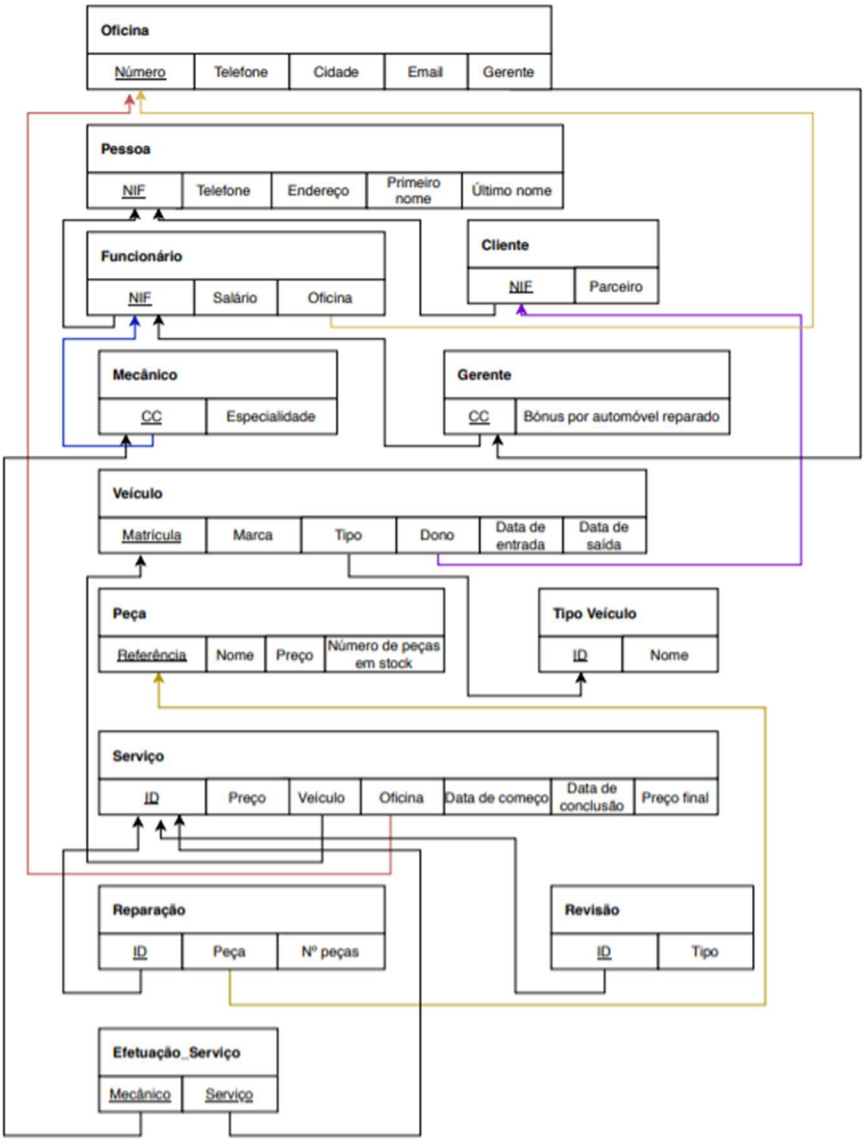

Figura 2 - Esquema Relacional

SQL Data Definition Language

Para especificar a informação acerca das nossas relações, utilizamos o *SQL Data Definition Language.*

No início criamos uma base de dados no disco local, mas como isso não era possível no servidor das aulas, tivemos de a adaptar para um *schema* na nossa base de dados: "`CREATE SCHEMA Oficina_DB;`".

Utilizamos vários tipos de dados, desde *int*, *money*, *varchar(n)*, *bit*, entre outros. Criámos todas as tabelas de acordo com o esquema relacional, tendo em atenção as *primary keys* e *as foreign keys*. Utilizamos também alguns tipos de restrições "*Unique*", por exemplo para o telefone.

Nota: é possível ver toda a estrutura de dados no ficheiro "Oficina_DB_schema.sql".

```
CREATE TABLE Oficina_DB.Funcionario (
    nif             INT                 NOT NULL
        CHECK (nif > 99999999 AND nif < 1000000000),
    salario         MONEY               NULL,
    n_oficina       INT                 NULL
        CHECK (n_oficina > 0),
    FOREIGN KEY (nif) REFERENCES Oficina_DB.Pessoa(nif)
        ON UPDATE CASCADE,
    PRIMARY KEY (nif)
);
```

Figura 3 - Exemplo DDL

SQL Data Manipulation Language

Grande parte das operações na interface com a base de dados apenas são possíveis devido à implementação do *SQL Data Manipulation Language.* O seguinte foi utilizado para as diversas operações de visualização de conteúdo, onde utilizamos na sua maioria o comando de projeção "*SELECT*" possuindo por vezes restrições usadas com o comando "*WHERE*" entre outros comandos para as diversas operações. Também usamos os comandos "*UPDATE*", "*INSERT INTO*", "*DELETE*" para realizar diversas operações como a adição, edição e eliminação de tabelas na base de dados. Estes foram utilizados maioritariamente em *Stored Procedures* para facilitar a sua invocação. Foram usados por diversas vezes instruções de renomeação, comparação de strings, agregações e junções.

```
DELETE Oficina_DB.Veiculo FROM Oficina_DB.Veiculo JOIN Oficina_DB.Cliente ON dono = nif WHERE nif=@nif;
DELETE Oficina_DB.Cliente WHERE nif = @nif;

SELECT * FROM Oficina_DB.Funcionario WHERE nif=@nif;
```

Figura 4 - Exemplo DML 1

```
SELECT * FROM Oficina_DB.Peca WHERE referencia LIKE '%'+@search+'%'
```

Figura 5 - Exemplo DML 2

Normalização

No nosso projeto não foi necessário fazer normalização, visto que não há informação duplicada nas tabelas e a utilização de NULL foi mínima, apenas no caso de não se pretender fornecer certas informações, como por exemplo a morada ou apelido. Não existem relações dentro de outras relações, atributos compostos, não existem dependências parciais nem transitivas, o que nos permitiu concluir que se encontra na Forma Normal Boyce-Codd, ou seja, todos os atributos estão dependentes apenas da chave.

Views

No nosso projeto apenas recorremos ao uso de *views* para observar o total de ordenados por oficina e para observar o lucro total por oficina. Não são efetuadas quaisquer operações com estas *views* e apenas servem para ver os resultados. Podem ser encontradas no menu início.

User Defined Functions

Não foram utilizadas muitas UDF no nosso projeto, foi preferido o uso de *Stored Procedures* visto que as vantagens são as mesmas (apenas são compiladas a primeira vez) e não foi necessário retornar um valor no fim das operações. No entanto foi criada uma UDF responsável por retornar o NIF do Mecânico responsável por um Serviço. O Serviço é fornecido como argumento de entrada (id do Serviço) e a função retorna o NIF do Mecânico responsável.

Stored Procedures

Stored procedures foram as *batchs* mais utilizadas no nosso projeto. A sua simplicidade, desempenho e capacidade de possuir parâmetros de entrada e saída foram um grande fator para que isso acontecesse. Críamos as mais diversas *Stored Procedures*, desde para adicionar, editar e eliminar dados.

Outro fator que nos levou a implementar *Stored Procedures* foi o facto de, através destes, ser possível implementar *transactions* e melhorar a segurança da base de dados.

```
CREATE PROC searchPeca(
    @search VARCHAR(30)
    )
AS

SET NOCOUNT ON;
BEGIN TRY;
    BEGIN TRANSACTION;
        SELECT * FROM Oficina_DB.Peca WHERE referencia LIKE '%'+@search+'%'
    COMMIT TRANSACTION;
END TRY
BEGIN CATCH
    ROLLBACK TRANSACTION;
    RAISERROR('Inventory Transaction Error', 16, 1);
END CATCH;

GO
```

Transactions

Implementamos *transactions* em todos os *Stored Procedures* para tentar impedir que certos erros pudessem resultar em operações incompletas.

Assim, em caso de insucesso, dá-se um *rollback* na operação.

Segurança

Quanto à segurança, a utilização de *Stored Procedures* em vez de SQL dinâmico faz com que haja uma validação dos dados do utilizador.

A utilização das *transactions* reduz ao mínimo a apresentação da informação de erros.

Isto leva a que seja mais difícil ou até mesmo impossível o processo de *SQL Injection* na nossa base de dados.

Outras funcionalidades

Quando se cria uma oficina ou um serviço, o número de oficina e id do serviço são atribuídos automaticamente, recorrendo ao tipo INDENTITY(1,1), ou seja, começa no valor 1 e é incrementado 1 unidade cada vez que se cria uma oficina ou um serviço, respetivamente.

Ao apagar uma oficina, todos os veiculos e serviços relativos a essa oficina são eliminados. No entanto, os funcionários e clientes continuam presentes na Base de Dados com o valor da oficina a NULL.

Hugo Moinheiro 84931

Pedro Silva 89228