universidade de aveiro

deti universidade de aveiro
departamento de eletrónica,
telecomunicações e informática

# Bases de Dados

Professor Carlos Costa

Departamento de Eletrónica, Telecomunicações e Informática

## Gestão de um supermercado

Grupo P8G2

| | | |
|---|---|---|
| Ana Raquel Paradinha | 102491 | LEI |
| Paulo Pinto | 103234 | LEI |

A participação na realização do trabalho foi igualitária.

# Índice

# 1. Introdução

No âmbito do projeto final da unidade curricular de Bases de Dados propusemo-nos a desenvolver um sistema de gestão para supermercados que abrange tanto a organização dos funcionários como dos produtos em stock.

Nesse sentido, os dois elementos supracitados constituem as entidades centrais do sistema, sendo que a grande maioria das funcionalidades desenvolvidas incidem neles.

No processo de criação do sistema tivemos como principal objetivo a aplicação correta e adequada dos conhecimentos obtidos ao longo do semestre.

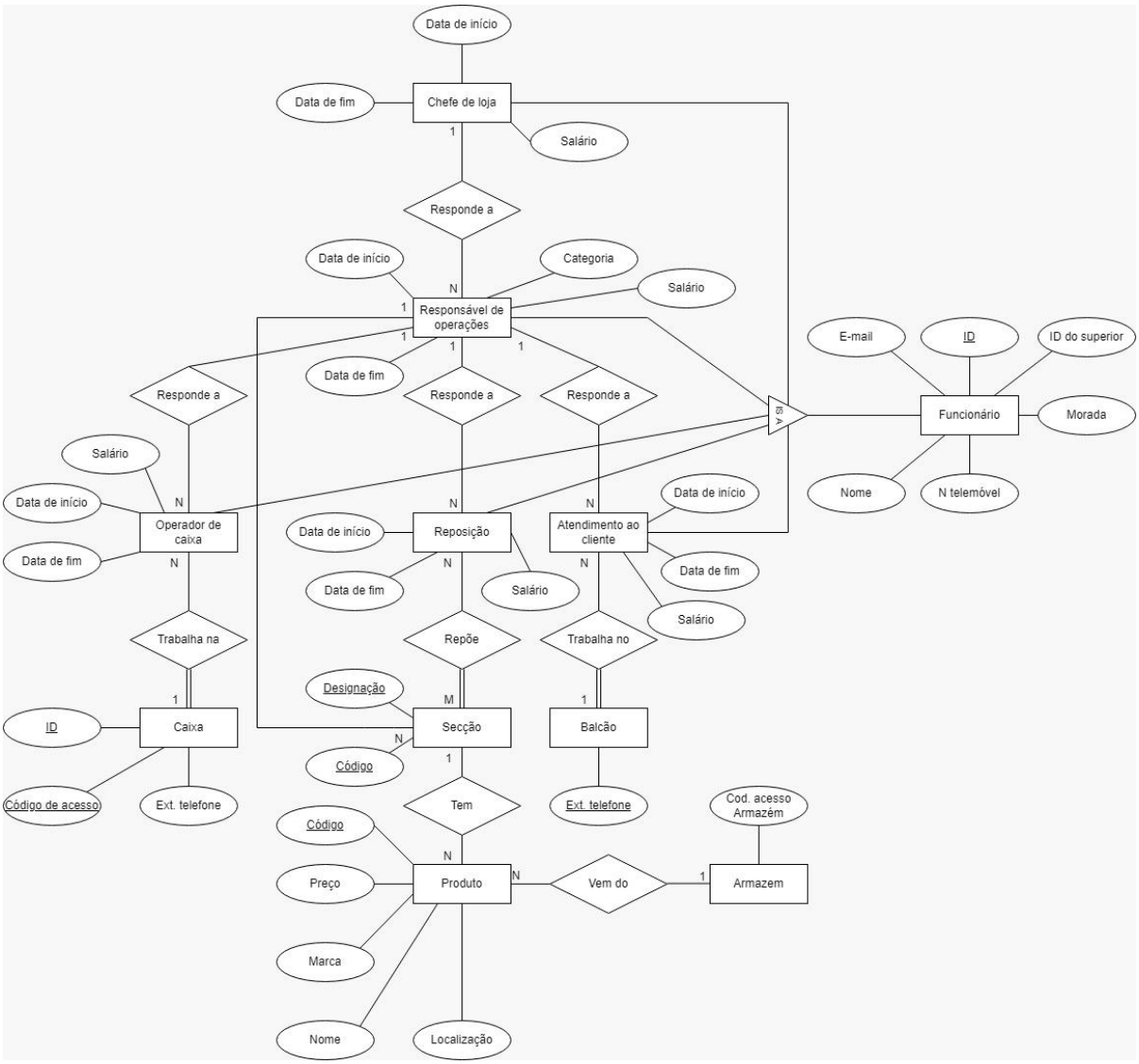
# 2. Análise de Requisitos

Considerando o funcionamento e organização de um supermercado real enunciamos as seguintes premissas:

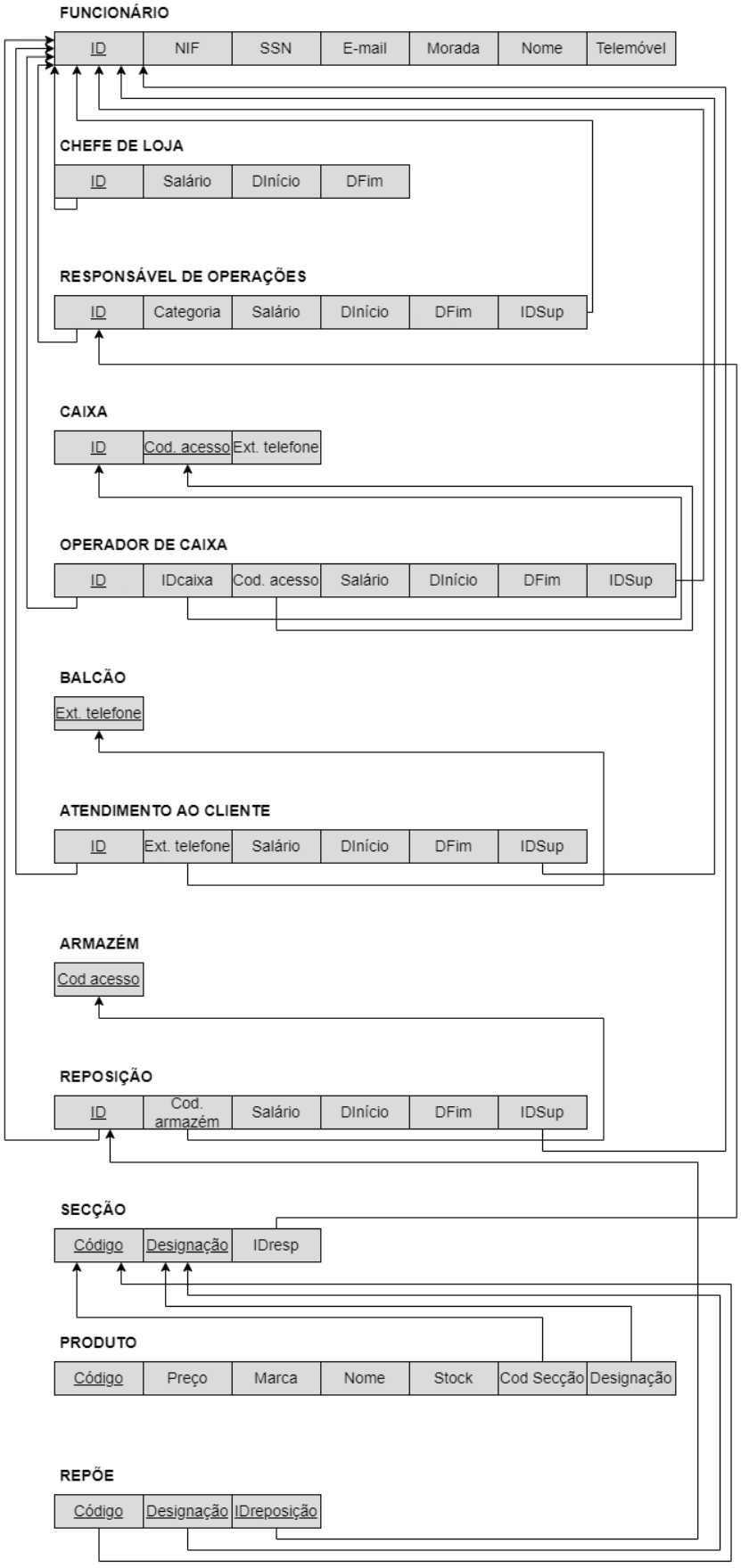
- Cada produto é identificado por um código, preço, marca, nome, localização no supermercado e stock disponível.
- O supermercado tem um armazém com um código de acesso ao mesmo.
- Os produtos são divididos por secções, as quais têm uma designação associada a um número e um responsável.
- Cada caixa tem um código de acesso, um número identificativo e uma extensão de telefone.
- O balcão de atendimento ao cliente tem uma extensão de telefone.
- Todos os funcionários são identificados pelo nome, nº telemóvel, email, morada, número de funcionário, número do superior direto e salário.
- Os operadores de caixa têm uma e uma só caixa associada e o respetivo código de acesso.
- Os empregados da reposição estão divididos por secções específicas, podendo operar em mais do que uma.
- Há também empregados de atendimento ao cliente.
- Os responsáveis operacionais têm uma categoria e um conjunto de empregados associado, todos do mesmo tipo.
- Existe um chefe de loja.
- Em todos os postos de trabalho são guardadas as datas de início e de fim de funções para cada funcionário.

# 3. Desenho da base de dados

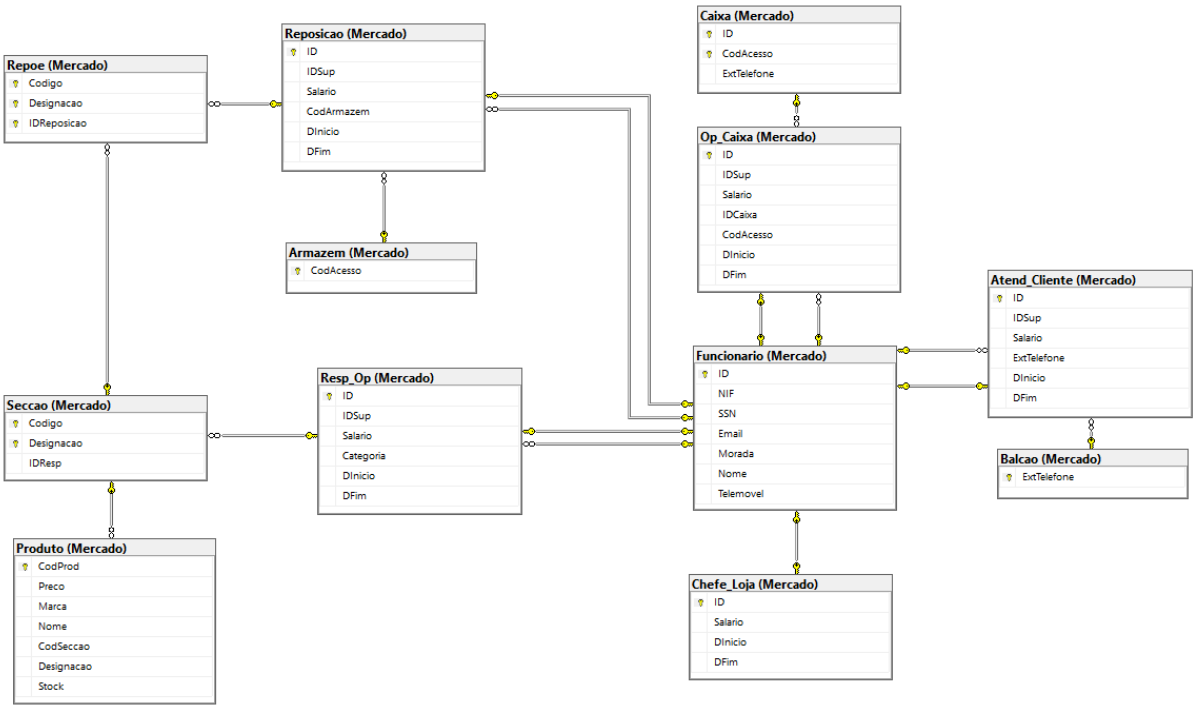
## 3.1. Diagrama de Entidade Relacionamento



## 3.2. Esquema Relacional



# 3.3. Esquema Relacional – SQL



# 4. Construção da base de dados

Para a construção da base de dados recorremos à linguagem SQL, que foi lecionada nas aulas práticas ao longo do semestre.

Com efeito, começámos pela criação das tabelas necessárias (*Data Definition Language*) e posteriormente fizemos as inserções de dados em cada uma delas (*Data Manipulation Language*).

## 4.1. Criação das tabelas

De modo a criar as diferentes entidades identificadas no processo de desenho da base de dados, assim como as restrições e tipos de chaves de cada uma, utilizamos a sintaxe DDL.

De seguida, podem ser observados alguns exemplos deste processo:

```
create table Mercado.Funcionario (
    ID                      int,
    NIF                     char(9)                 not null,
    SSN                     char(11)                not null,
    Email                   varchar(150),
    Morada                  varchar(150),
    Nome                    varchar(150)            not null,
    Telemovel               char(9),
    primary key (ID),
    unique (NIF), unique (SSN)
);
```

*Figura 1 - Criação da tabela Funcionário*

```
create table Mercado.Resp_Op (
    ID                  int,
    IDSup               int                 not null,
    Salario             decimal(6, 2)       not null,
    Categoria           varchar(50)         not null,
    DInicio             date                not null,
    DFim                date,
    primary key (ID)
);

alter table Mercado.Resp_Op add constraint Resp_Op_ID_FK foreign key (ID) references Mercado.Funcionario (ID);
alter table Mercado.Resp_Op add constraint Resp_Op_IDSup_FK foreign key (IDSup) references Mercado.Funcionario (ID);
```

*Figura 2 - Criação da tabela Res_Op*

## 4.2. Inserção de dados

Finalizado o passo anterior prosseguimos para a inserção da informação nas tabelas, utilizando DML.

```
-- Chefe de Loja
insert into Mercado.Chefe_Loja values ('1', 1993.05, '2000-01-01', '2008-12-27');
insert into Mercado.Chefe_Loja values ('2', 1897.05, '2008-12-27', '2022-02-20');
insert into Mercado.Chefe_Loja values ('5', 1560.22, '2022-02-20', null);
```

*Figura 3 - Inserção de tuplos na tabela Chefe_Loja*

```
-- Repoe
insert into Mercado.Repoe values ('344312', 'Mercearia', 13);
insert into Mercado.Repoe values ('914671', 'Padaria e Pastelaria', 13);
insert into Mercado.Repoe values ('227853', 'Peixaria e Talho', 17);
insert into Mercado.Repoe values ('142815', 'Frutas e Legumes', 17);
insert into Mercado.Repoe values ('992369', 'Beleza e Higiene', 19);
insert into Mercado.Repoe values ('352480', 'Limpeza', 19);
insert into Mercado.Repoe values ('227853', 'Peixaria e Talho', 28);
insert into Mercado.Repoe values ('142815', 'Frutas e Legumes', 28);
```

*Figura 4 - Inserção de tuplos na tabela Repoe*

# 5. SQL Programming

## 5.1. Views

Para facilitar a consulta de alguns dados não acessíveis diretamente a partir das tabelas definidas e que são utilizados frequentemente decidimos criar algumas views, otimizando este processo.

Essencialmente, definimos views para obter o separadamente a lista de funcionários antigos e atuais de cada cargo.

```
create view Mercado.Reposicao_atual as
    select  F.ID, F.NIF, F.SSN, F.Email, F.Morada, F.Nome, F.Telemovel, R.IDSup, R.Salario, R.CodArmazem, R.DInicio, R.DFim
    from Mercado.Reposicao as R
    join Mercado.Funcionario as F on R.ID = F.ID
    where R.DFim is null;
go
```

*Figura 5 - View funcionários de Reposição atuais*

```
create view Mercado.Reposicao_antigo as
    select  F.ID, F.NIF, F.SSN, F.Email, F.Morada, F.Nome, F.Telemovel, R.IDSup, R.Salario, R.CodArmazem, R.DInicio, R.DFim
    from Mercado.Reposicao as R
    join Mercado.Funcionario as F on R.ID = F.ID
    where R.DFim is not null;
go
```

*Figura 6 - View funcionários de Reposição antigos*

## 5.2. Indexes

Decidimos implementar esta estrutura, apesar da nossa base de dados ter uma dimensão reduzida, associando-a às tabelas Funcionário e Produto, pois são as mais utilizadas.

Assim, fazemos uso dos indexes quando pesquisamos um Funcionário pelo seu nome ou ID e um Produto pelo código ou nome.

```
create index searchNomeFuncionario
    on Mercado.Funcionario (Nome)
go

create index searchIDFuncionario
    on Mercado.Funcionario (ID)
go

create index searchCodProduto
    on Mercado.Produto (CodProd)
go

create index searchNomeProduto
    on Mercado.Produto (Nome)
go
```

*Figura 7 - Index criados*

## 5.3. Cursor

Os cursores permitem-nos percorrer sequencialmente todos os tuplos retornados por uma consulta e, por isso, a sua implementação foi-nos útil na udf criada para selecionar todos os produtos de uma dada secção, como podemos observar de seguida.

```
create function Mercado.getProdutosSeccao (@Designacao varchar(50)) returns @produtos table (
    CodProd                 char(3),
    Preco                   decimal(5, 2)               not null,
    Marca                   varchar(150),
    Nome                    varchar(150),
    Stock                   int)
as
    begin
        declare @CodProd as char(3);
        declare @Preco   as decimal(5, 2);
        declare @Marca   as varchar(150);
        declare @Nome    as varchar(150);
        declare @Desgn   as varchar(50);
        declare @Stock   as int;

        declare c cursor
        for select CodProd, Preco, Marca, Nome, Designacao, Stock from Mercado.Produto;

        open c;
        fetch c into @CodProd, @Preco, @Marca, @Nome, @Desgn, @Stock;

        while @@FETCH_STATUS = 0
            begin
                if @Desgn = @Designacao
                    begin
                        insert into @produtos values (@CodProd, @Preco, @Marca, @Nome, @Stock);
                    end
                fetch c into @CodProd, @Preco, @Marca, @Nome, @Desgn, @Stock;
            end

        close c;
        deallocate c;
        return;
    end
```

*Figura 8 - Utilização de um cursor numa UDF*

## 5.4. Stored Procedure

Decidimos implementar alguns stored procedures, uma vez que estas estruturas nos permitem guardar uma sequência de consultas SQL sem ter de recompilar cada vez que queremos usá-las.

Utilizámos SPs para adicionar e eliminar elementos nas tabelas.

```
create proc Mercado.addReposicao(
@NIF char(9), @SSN char(11), @Email varchar(150), @Morada varchar(150), @Nome varchar(150), @Telemovel char(9),
@Salario decimal(6, 2), @DInicio date, @DFim date)
as
    begin
        begin try
            begin tran
                declare @newID int;
                declare @IDSup int;
                declare @CodArmazem char(6);
                set @IDSup = (select ID from Mercado.Resp_Op_atual where Categoria = 'Reposicao');
                set @CodArmazem = (select CodAcesso from Mercado.Armazem);
                exec Mercado.addFunc @NIF, @SSN, @Email, @Morada, @Nome, @Telemovel, @newID output;
                insert into Mercado.Reposicao values (@newID, @IDSup, @Salario, @CodArmazem, @DInicio, @DFim);
            commit tran
        end try
        begin catch
            rollback tran
            raiserror('Funcionario não inserido! Algum dado está incorreto.', 16, 1);
        end catch
    end
go
```

*Figura 9 - SP para adicionar um funcionário de Reposição*

```
go
create proc Mercado.deleteProduto @CodProd char(3)
as
    begin
        begin tran
            delete from Mercado.Produto where  CodProd = @CodProd;
        commit
    end
go
```

*Figura 10 - SP para eliminar um Produto*

## 5.5. User Defined Function

As UDFs são semelhantes aos stored procedures a nível de benefícios e como podem retornar tabelas e ser usadas dentro de SPs decidimos implementar algumas.

Nesse sentido, utilizamos esta estrutura para obter a lista de todos os funcionários (antigos e atuais) de cada cargo, calcular IDs sequenciais para funcionários e caixas, verificar se os dados inseridos e definidos como únicos não estão repetidos e obter um atributo com base noutro.

```
go
create function Mercado.obterChefeLoja() returns table
as
    return (select * from Mercado.Chefe_Loja_atual
            union
            select * from Mercado.Chefe_Loja_antigo)
go
```

*Figura 11 - UDF para obter todos os Chefes de Loja*

```
create function Mercado.nextID() returns int
as
    begin
        declare @ID as int;
        select @ID = max(ID) + 1 from Mercado.Funcionario;
        return @ID;
    end
go
```

*Figura 12 - UDF para calcular o próximo ID de funcionário a utilizar*

```
GO
CREATE FUNCTION Mercado.checkSSN (@SSN char(11)) RETURNS int
AS
    BEGIN
        DECLARE @counter INT
        SELECT @counter = COUNT(1) FROM Mercado.Funcionario as F WHERE F.SSN=@SSN
        RETURN @counter
    END
GO
```

*Figura 13 - UDF para verificar se o SSN é único*

## 5.6. Trigger

Na nossa base de dados, utilizamos um trigger que atua quando são feitas alterações nas tabelas dos funcionários.

```
create trigger Mercado.checkAddFunc on Mercado.Funcionario
after insert, update
as
    set nocount on;
    declare @NIF as char(9);
    declare @SSN as char(11);
    select @NIF = NIF from inserted;
    select @SSN = SSN from inserted;

    if len(@NIF) <> 9
    begin
        raiserror('NIF mal inserido!', 16, 1);
        rollback tran;
    end
    if len(@SSN) <> 11
    begin
        raiserror('SSN mal inserido!', 16, 1);
        rollback tran;
    end
go
```

*Figura 14 - Trigger ativado quando se adiciona um funcionário*

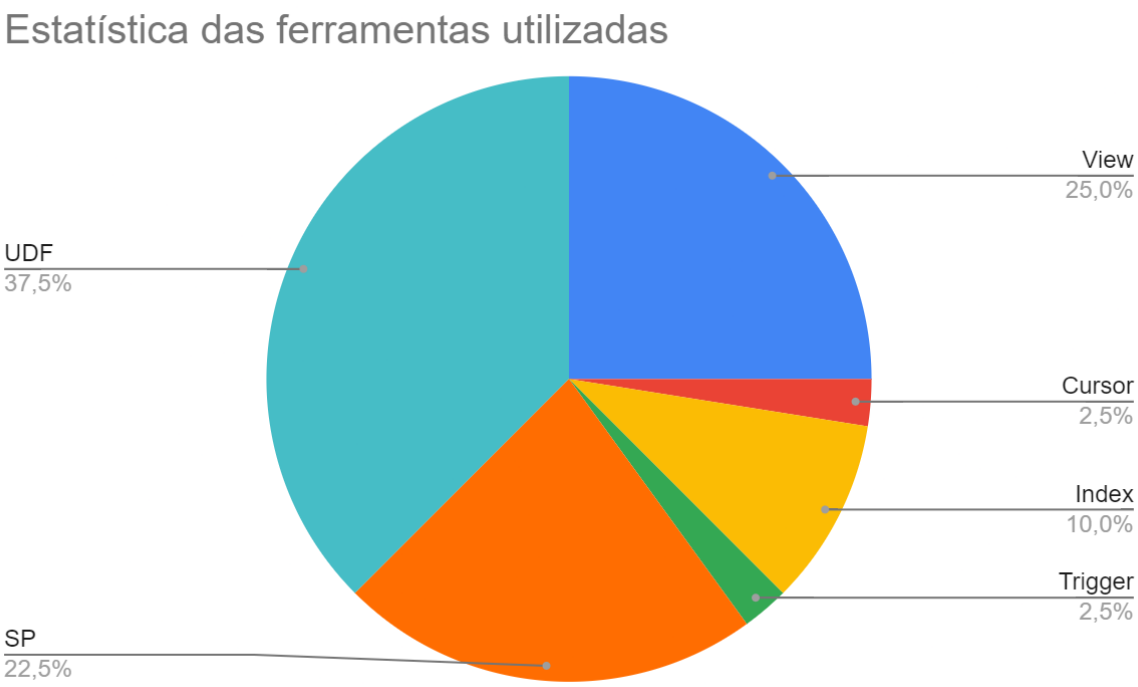



*Figura 15 - Gráfico da distribuição de cada ferramenta utilizada*

# 6. Interface Gráfica

O desenvolvimento da interface gráfica foi feito no Visual Studio através da aplicação de Windows Forms e utilizando C# para o código. Com estas ferramentas conseguimos apresentar a informação da base de dados de forma simples e intuitiva.

Assim, a nossa interface inicial divide-se em 3 partes. A primeira, do lado esquerdo, apresenta uma lista dos dados selecionados, na seguinte encontram-se os campos com as informações de cada dado e nos quais estes podem ser alterados e, por fim, temos alguns botões que nos permitem escolher o que queremos que seja apresentado na lista.

Por conseguinte, consoante o botão selecionado aparecem novas ações que podem ser executadas, inserção e remoção de dados das tabelas. Além disso, são apresentados dados sobre o número de funcionários de cada tipo, quando o respetivo botão é escolhido.

Quando a inserção ou remoção é efetuada o utilizador é notificado do sucesso ou insucesso das suas ações, assim como da validade dos dados que insere.

De seguida, podemos ver algumas imagens demonstrativas destes processos.

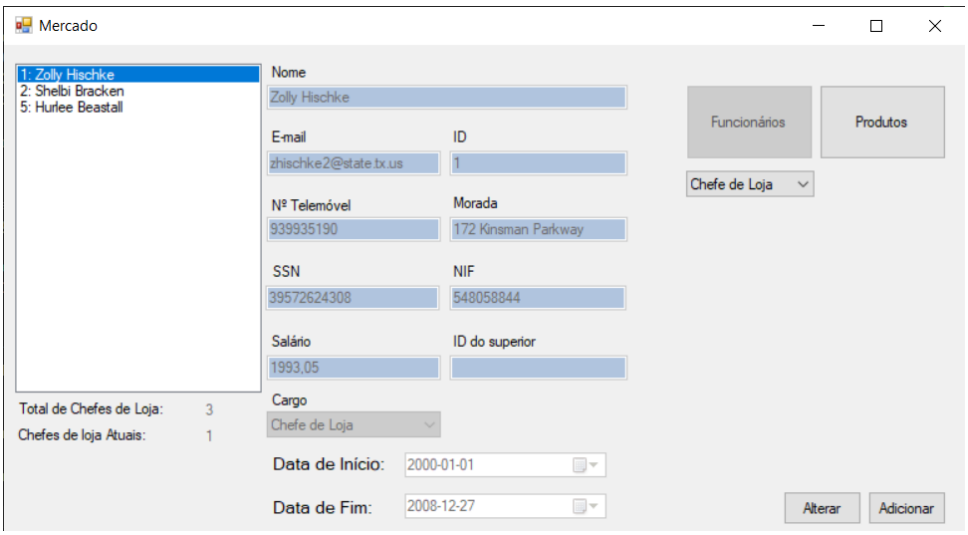



*Figura 16 - Mostrar todos os chefes de loja*

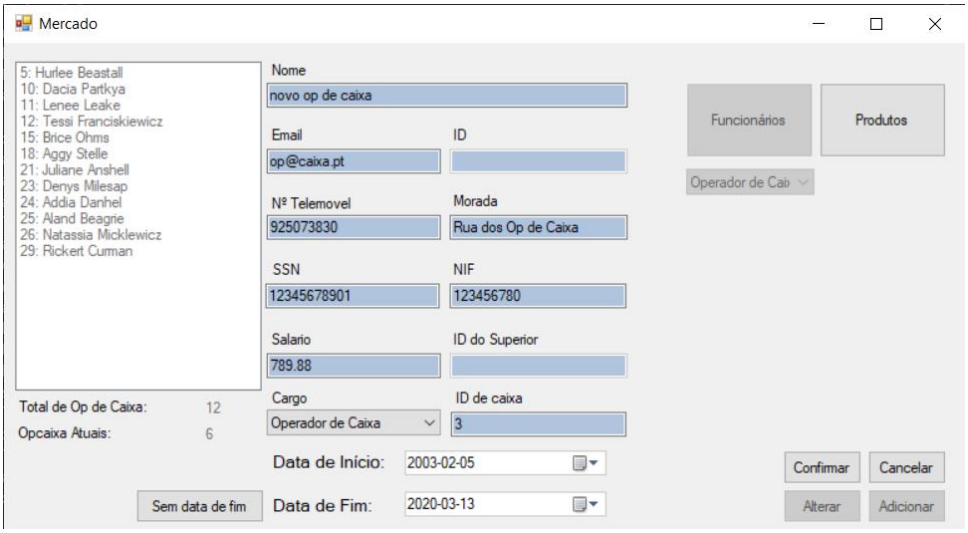



*Figura 17 - Adicionar operador de caixa*

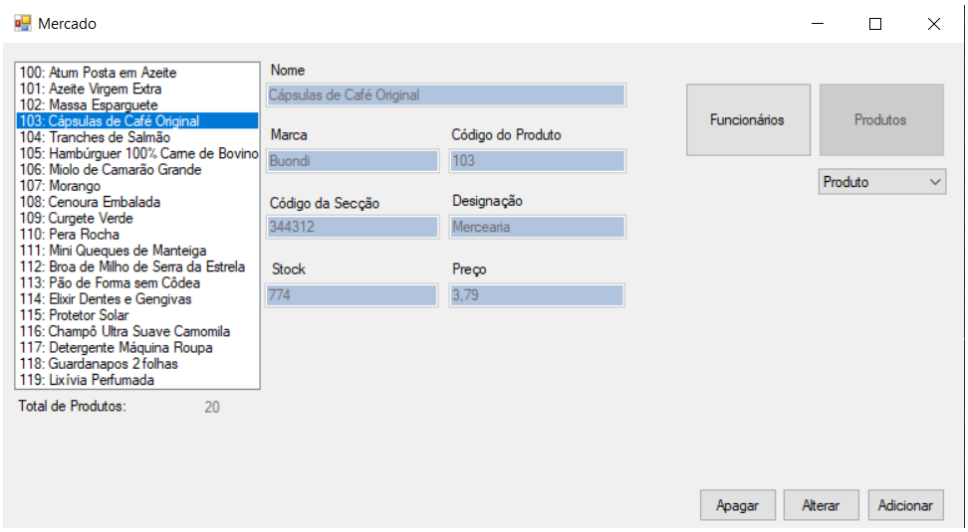



*Figura 18 - Mostrar dados de um produto*

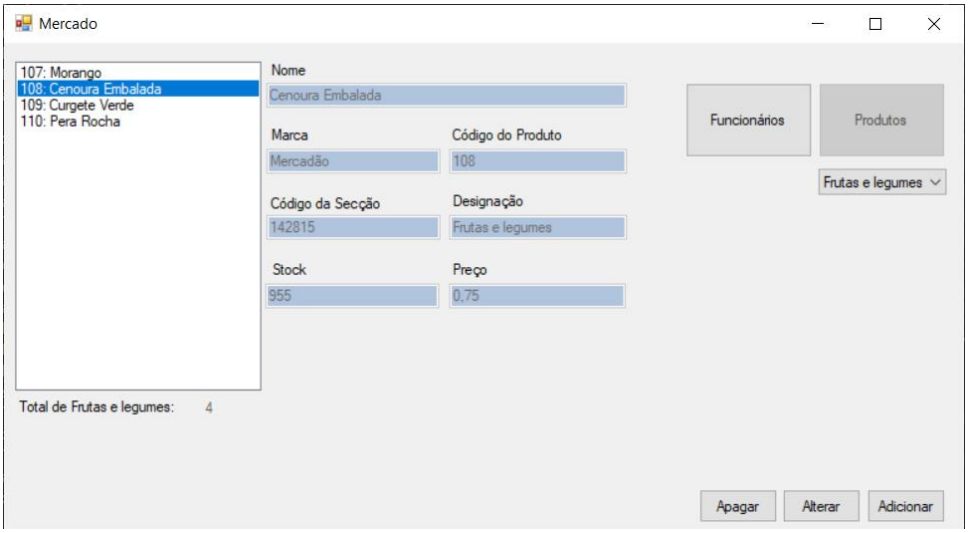



*Figura 19 - Mostrar produtos por secção*

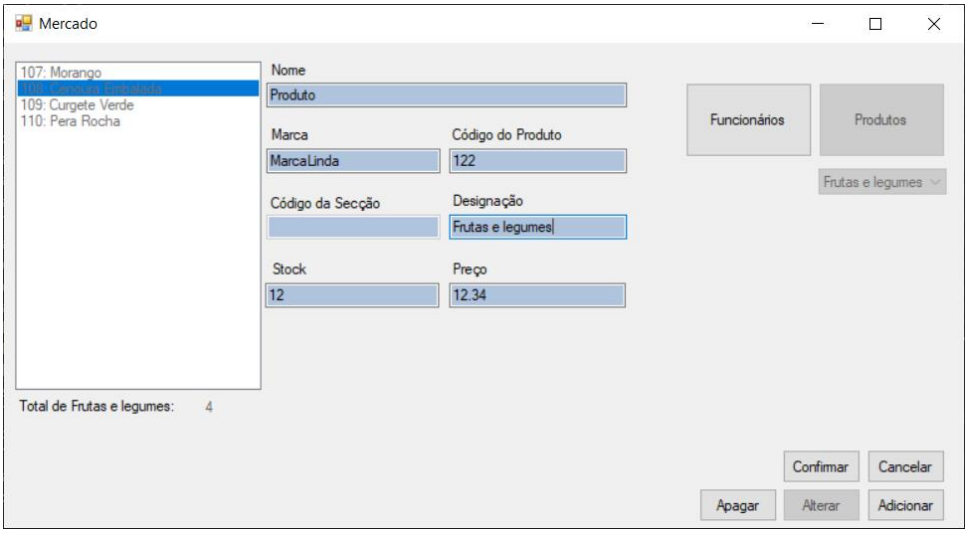



*Figura 20 - Adicionar produto*

# 7. Segurança

Para efeitos de segurança da base de dados tivemos em conta alguns critérios:

- Verificação dos dados inseridos pelo utilizador;
- Apresentação de mensagens de erro.

# 8. Anexos

Juntamente com o presente relatório incluímos na entrega os ficheiros necessários para compilar e executar o programa desenvolvido, assim como um vídeo que demonstra o funcionamento da interface.

# 9. Conclusão

Concluindo, consideramos que atingimos os objetivos pretendidos, já que, ao aplicar os conceitos aprendidos ao longo do semestre, conseguimos implementar a interação da interface gráfica com a base de dados corretamente.

Ao longo do processo de desenvolvimento a maior dificuldade foi aprender como criar a interface, pois nunca tínhamos trabalhado com C#.

Como trabalho futuro, gostaríamos de implementar mais funcionalidades e dados estatísticos na interface.