Ano Letivo 2023/2024

deti universidade de aveiro
departamento de electrónica, telecomunicações e informática

Base de Dados – P4

Loja Académica e Papelaria

Trabalho realizado por:

André Vasques Dora, nº113613

António Fernandes Alberto, nº114622

Índice

Introdução

O nosso trabalho foca-se na gestão do stock de uma loja académica. Esta escolha foi motivada porque achamos interessante fazer uma implementação de um sistema que poderia existir na vida real.

O conceito é uma loja académica, também com produtos de papelaria. Existiria uma interface em que os clientes podem pesquisar e filtrar o conteúdo da loja, para ver os itens que esta tem para venda, criar um traje e adicionar conteúdos ao seu traje, e outra em que o administrador da loja poderia pesquisar, adicionar, apagar e modificar os detalhes dos produtos da sua loja.

Na sua implementação decidimos apenas criar a interface do administrador do sistema, já que teria as funcionalidades dos clientes juntamente com as dos administradores das lojas.

Estrutura da pasta do projeto

Dentro da pasta “SQL” está todo o código referente à criação das tables, o DML, o código de inicialização das views, triggers, stored procedures, e as imagens dos diagramas DER e ER. Na pasta “TrabalhoFinal Codigo” encontra-se o código fonte do projeto.

Funcionalidades

Entidade	Funcionalidade
Utilizador	• Verificar disponibilidade dos produtos; • Pesquisar produtos da loja; • Filtrar produtos na pesquisa; • Criar um Traje Pessoal • Adicionar Artigos ao seu Traje
Administrador	• Adicionar/remover/alterar produto do stock; • Verificar stock dos produtos; • Editar produtos do stock; • Adicionar/remover Trajes da BD • Adicionar/remover Artigos dos Trajes

Entidades

Loja: Loja académica, papelaria e centro de fotocópias, caracterizada pelo seu Endereço. Vende vários itens.

Peça do traje: Peça constituinte do traje, vendida pela loja e identificada pelo seu ID. Simboliza as diferentes peças do traje, que se diferenciam em si pelo seu tipo. Há peças de várias universidades, tamanhos e géneros.

Traje: Conjunto de Peças do Traje vendidas pela loja, com um certo dono e identificado pelo seu ID. Tem um certo número de peças e de acessórios académicos.

Artigo académico: Categoria de itens vendidos pela loja, cada um identificado pelo seu ID. Cada artigo tem também um nome, e quantidade em stock.

Emblema: Artigo académico vendido pela loja, identificado pelo seu ID. Pode ter diferentes formas e tipos.

Pin: Artigo académico vendido pela loja, identificado pelo seu ID. Pode ser com tacha ou com alfinete de dama.

Pasta: Artigo académico vendido pela loja, identificado pelo seu ID. Pode ser de diferentes universidades.

Nó: Artigo académico vendido pela loja, identificado pelo seu ID. Tem dois tipos, e pode ter duas cores.

Chapéu: Artigo académico vendido pela loja, identificado pelo seu ID. Diferente consoante a universidade e o género.

Artigo de papelaria: Categoria de itens vendidos pela loja, cada um identificado pelo seu ID. Cada artigo de papelaria tem nome, marca e quantidade em stock.

Caneta: Artigo de papelaria vendido pela loja, identificado pelo seu ID. Pode ter diferentes cores e tipo (esferográfica, caneta de gel, etc.).

Lápis: Artigo de papelaria vendido pela loja, identificado pelo seu ID. Pode ter diferentes durezas.

Diagrama entidade-relacionamento

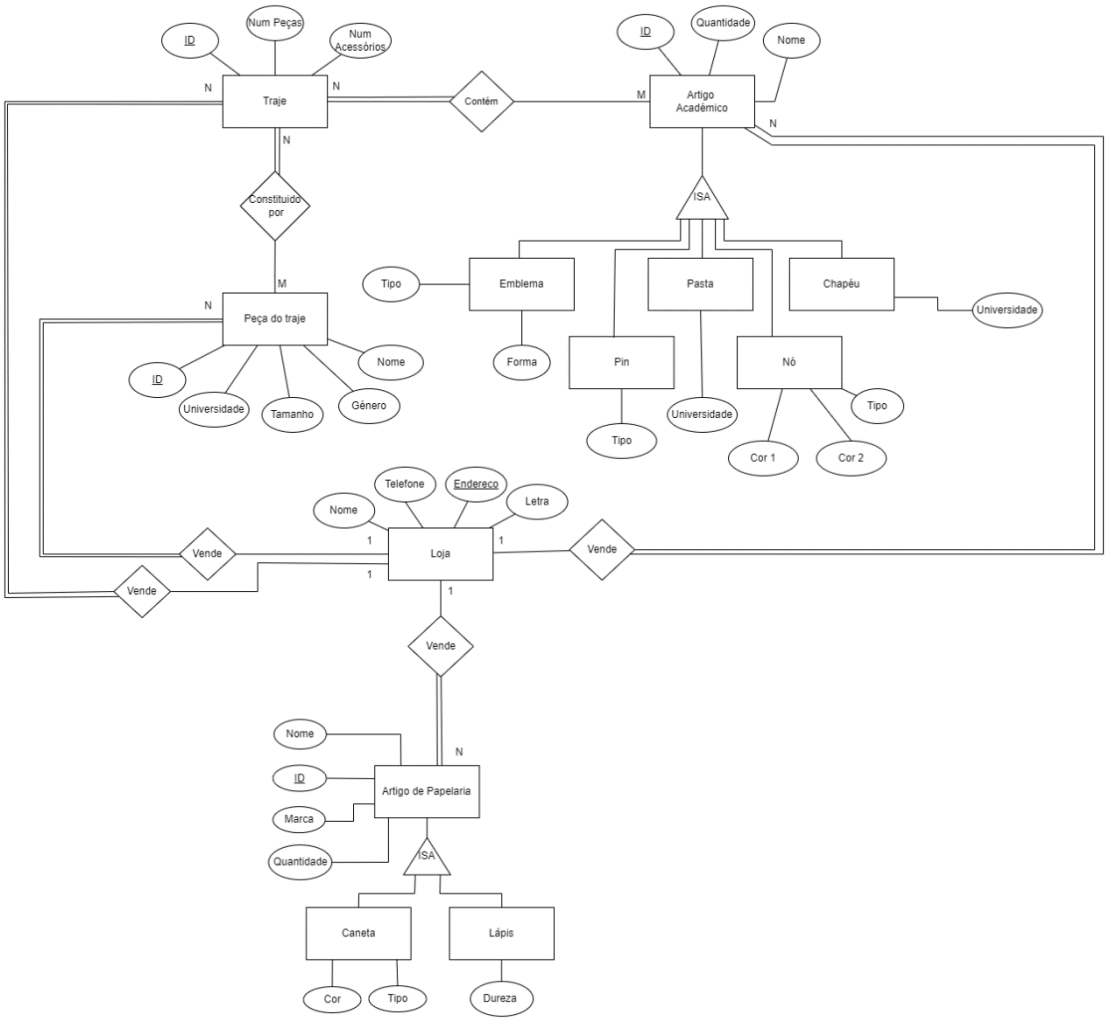

Figura 1 : Diagrama entidade-relacionamento

Esquema Relacional

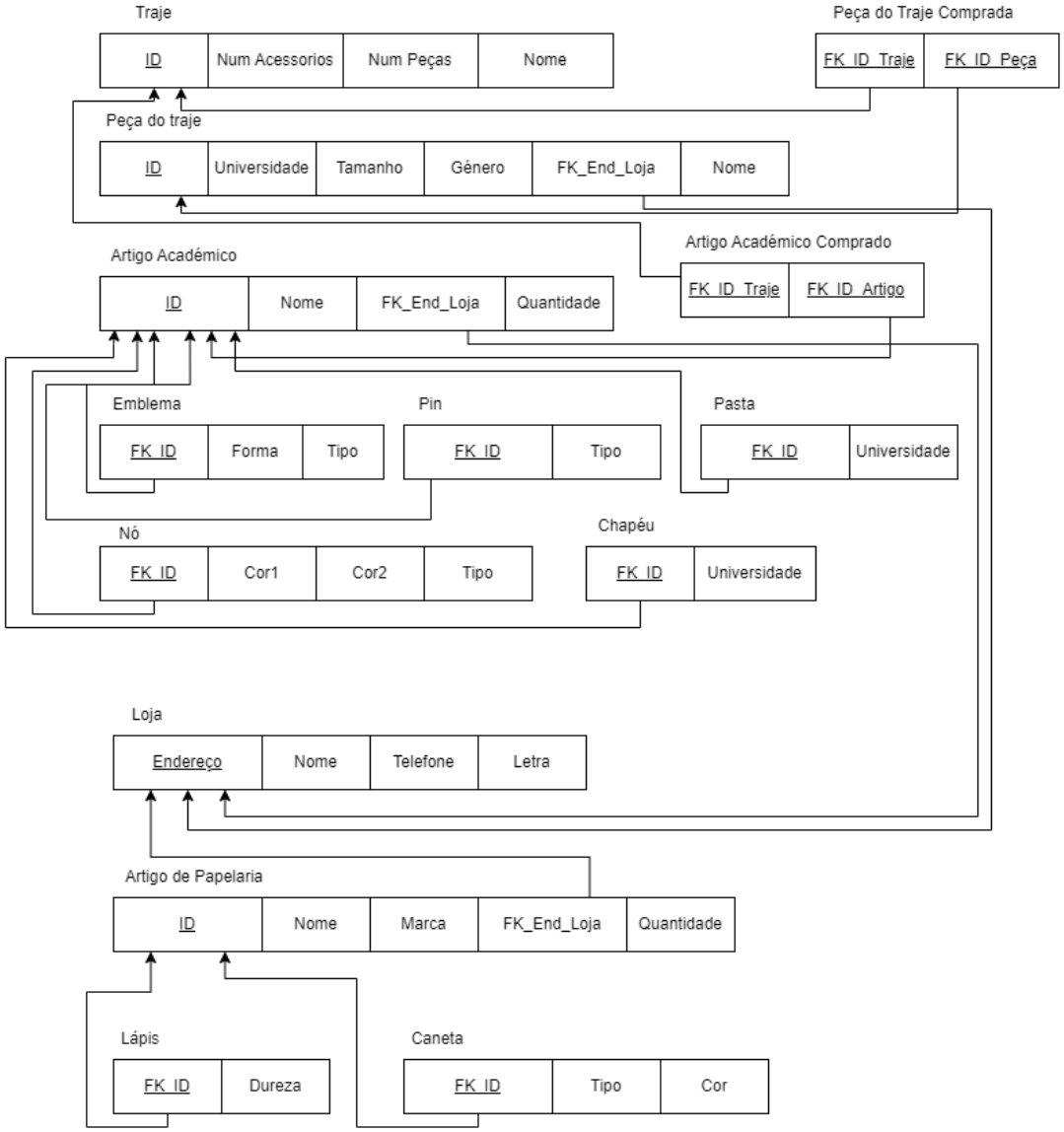

Figura 2 : Esquema Relacional

Diagrama Relacional (SGBD)

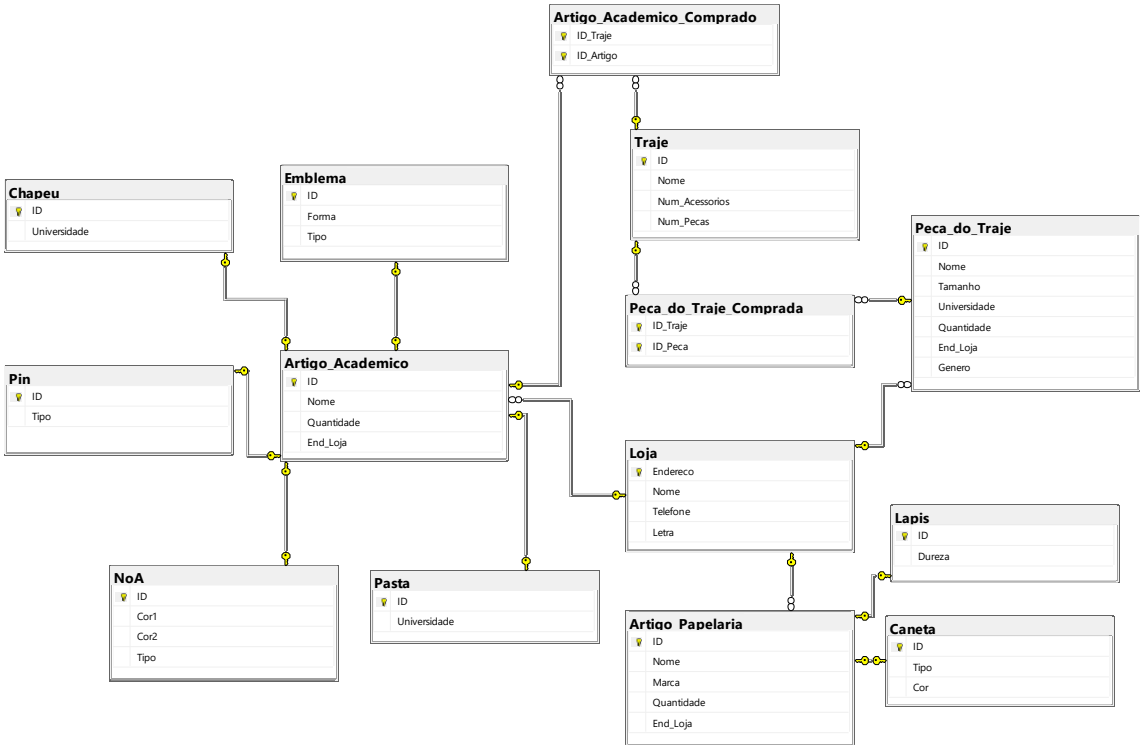

Figura 3 : Diagrama Relacional (SGBD)

Queries SQL

Data Definition Language (DDL)

Para definir a estrutura da base de dados utilizamos a DDL.

Abaixo apresentamos o código feito para gerar as tabelas pertencentes à base de dados que criamos tanto como os atributos associados a cada uma delas.

```
-- Drop tables if they exist
DROP TABLE IF EXISTS Lapis;
DROP TABLE IF EXISTS Caneta;
DROP TABLE IF EXISTS Artigo_Papelaria;
DROP TABLE IF EXISTS Artigo_Academico_Comprado;
DROP TABLE IF EXISTS Peca_do_Traje_Comprada;
DROP TABLE IF EXISTS Traje;
DROP TABLE IF EXISTS NoA;
DROP TABLE IF EXISTS Chapeu;
DROP TABLE IF EXISTS Pasta;
DROP TABLE IF EXISTS Pin;
DROP TABLE IF EXISTS Emblema;
DROP TABLE IF EXISTS Artigo_Academico;
DROP TABLE IF EXISTS Peca_do_Traje;
DROP TABLE IF EXISTS Loja;

-- Create tables
CREATE TABLE Loja (
    Endereco VARCHAR(60) PRIMARY KEY NOT NULL,
    Nome VARCHAR(30) NOT NULL,
    Telefone CHAR(9) NOT NULL,
    Letra CHAR(1) NOT NULL
);

CREATE TABLE Peca_do_Traje (
    ID CHAR(6) NOT NULL PRIMARY KEY,
    Nome VARCHAR(60) NOT NULL,
    Tamanho VARCHAR(3) NOT NULL,
    Universidade VARCHAR(32) NOT NULL,
    Quantidade INT NOT NULL,
    End_Loja VARCHAR(60) FOREIGN KEY REFERENCES Loja (Endereco),
    Genero VARCHAR(1) NOT NULL
);
```

```sql
CREATE TABLE Artigo_Academico (
    ID CHAR(6) NOT NULL PRIMARY KEY,
    Nome VARCHAR(60) NOT NULL,
    Quantidade INT NOT NULL,
    End_Loja VARCHAR(60) FOREIGN KEY REFERENCES Loja (Endereco)
);

CREATE TABLE Emblema (
    ID CHAR(6) NOT NULL PRIMARY KEY REFERENCES Artigo_Academico (ID) ON UPDATE CASCADE ON DELETE CASCADE,
    Forma VARCHAR(16) NOT NULL,
    Tipo VARCHAR(16) NOT NULL
);

CREATE TABLE Pin (
    ID CHAR(6) NOT NULL PRIMARY KEY REFERENCES Artigo_Academico (ID) ON UPDATE CASCADE ON DELETE CASCADE,
    Tipo VARCHAR(16) NOT NULL
);

CREATE TABLE Pasta (
    ID CHAR(6) NOT NULL PRIMARY KEY REFERENCES Artigo_Academico (ID) ON UPDATE CASCADE ON DELETE CASCADE,
    Universidade VARCHAR(32) NOT NULL
);

CREATE TABLE Chapeu (
    ID CHAR(6) NOT NULL PRIMARY KEY REFERENCES Artigo_Academico (ID) ON UPDATE CASCADE ON DELETE CASCADE,
    Universidade VARCHAR(32) NOT NULL
);

CREATE TABLE NoA (
    ID CHAR(6) NOT NULL PRIMARY KEY REFERENCES Artigo_Academico (ID) ON UPDATE CASCADE ON DELETE CASCADE,
    Cor1 VARCHAR(16) NOT NULL,
    Cor2 VARCHAR(16),


CREATE TABLE Traje (
    ID CHAR(6) NOT NULL PRIMARY KEY,
    Nome VARCHAR(60) NOT NULL,
    Num_Acessorios INT NOT NULL,
    Num_Pecas INT NOT NULL,
);

CREATE TABLE Peca_do_Traje_Comprada (
    ID_Traje CHAR(6) NOT NULL,
    ID_Peca CHAR(6) NOT NULL,
    PRIMARY KEY(ID_Traje, ID_Peca),
    FOREIGN KEY (ID_Traje) REFERENCES Traje(ID) ON DELETE NO ACTION ON UPDATE NO ACTION,
    FOREIGN KEY (ID_Peca) REFERENCES Peca_do_Traje(ID) ON DELETE NO ACTION ON UPDATE NO ACTION
);


CREATE TABLE Artigo_Academico_Comprado (
    ID_Traje CHAR(6) NOT NULL,
    ID_Artigo CHAR(6) NOT NULL,
    PRIMARY KEY (ID_Traje, ID_Artigo),
    FOREIGN KEY (ID_Traje) REFERENCES Traje (ID)  ON DELETE NO ACTION ON UPDATE NO ACTION,
    FOREIGN KEY (ID_Artigo) REFERENCES Artigo_Academico (ID) ON DELETE NO ACTION ON UPDATE NO ACTION
);

CREATE TABLE Artigo_Papelaria (
    ID CHAR(6) NOT NULL PRIMARY KEY,
    Nome VARCHAR(30) NOT NULL,
    Marca VARCHAR(30) NOT NULL,
    Quantidade INT NOT NULL,
    End_Loja VARCHAR(60) FOREIGN KEY REFERENCES Loja (Endereco)
);


CREATE TABLE Caneta (
    ID CHAR(6) NOT NULL PRIMARY KEY REFERENCES Artigo_Papelaria (ID) ON UPDATE CASCADE ON DELETE CASCADE,
    Tipo VARCHAR(30) NOT NULL,
    Cor VARCHAR(16) NOT NULL
);

CREATE TABLE Lapis (
    ID CHAR(6) NOT NULL PRIMARY KEY REFERENCES Artigo_Papelaria (ID) ON UPDATE CASCADE ON DELETE CASCADE,
    Dureza VARCHAR(2) NOT NULL
);
```

Data Manipulation Language (DML)

Para introduzir dados nas tabelas criadas com DDL, usamos DML.

Uma vez que o código DML feito é extenso, aqui apresentamos apenas parte do código que usamos para essa inserção dos dados.

```
insert into Chapeu (ID, Universidade) values
('ARC001', 'Universidade de Aveiro'),
('ARC002', 'Universidade de Aveiro'),
('ARC003', 'Universidade de Aveiro'),
('ARC004', 'Universidade do Porto'),
('ARC005', 'Universidade do Porto'),
('ARC006', 'Universidade do Porto'),
('ARC007', 'Universidade do Porto');
```

```
insert into NoA (ID, Cor1, Cor2, Tipo) values
('ARN001', 'Verde', NULL, 'Coxim Redondo'),
('ARN002', 'Castanho', NULL, 'Coxim Redondo'),
('ARN003', 'Branco Cru', NULL, 'Coxim Redondo')
('ARN004', 'Tijolo', 'Verde', 'Azelha'),
('ARN005', 'Tijolo', 'Cinzento', 'Azelha'),
('ARN006', 'Tijolo', 'Castanho', 'Azelha'),
('ARN007', 'Tijolo', 'Vermelho', 'Azelha'),
('ARN008', 'Tijolo', 'Vermelho', 'Azelha'),
('ARN009', 'Tijolo', 'Cinzento Escuro', 'Azelha
('ARN010', 'Tijolo', 'Roxo', 'Azelha'),
('ARN011', 'Tijolo', 'Branco', 'Azelha'),
('ARN012', 'Azul Escuro', 'Vermelho', 'Azelha')
('ARN013', 'Vermelho', 'Branco', 'Azelha'),
('ARN014', 'Preto', 'Vermelho', 'Azelha'),
('ARN015', 'Azul Claro', NULL, 'Azelha'),
('ARN016', 'Rosa Claro', NULL, 'Azelha'),
('ARN017', 'Azul Claro', NULL, 'Azelha'),
('ARN018', 'Azul Claro', NULL, 'Azelha'),
('ARN019', 'Azul Claro', NULL, 'Azelha'),
('ARN020', 'Azul Claro', 'Amarelo', 'Azelha'),
('ARN021', 'Azul Claro', NULL, 'Azelha');
```

Triggers

Utilizamos vários triggers para assegurar a atualização dos dadoscorretamente, nomeadamente:

Trigger para apagar os items das tabelas Peca_do_Traje_Comprada e Artigo_Academico_Comprado, quando é apagado um traje.

```
CREATE OR ALTER TRIGGER trg_delete_Traje
ON Traje
INSTEAD OF DELETE
AS
BEGIN
    DELETE FROM Peca_do_Traje_Comprada
    WHERE ID_Traje IN (SELECT ID FROM DELETED);

    DELETE FROM Artigo_Academico_Comprado
    WHERE ID_Traje IN (SELECT ID FROM DELETED);

    DELETE FROM Traje WHERE ID IN (SELECT ID FROM DELETED);
END;
GO
```

Trigger para adicionar +1 à quantidade dos Artigos Académicos referentes a um traje quando este é apagado.

```
CREATE OR ALTER TRIGGER trg_delete_traje_artigos
ON Traje
AFTER DELETE
AS
BEGIN
    DECLARE @ID_Traje CHAR(6);

    SELECT @ID_Traje = ID FROM deleted;

    IF @@ROWCOUNT > 0
    BEGIN
        UPDATE Artigo_Academico
        SET Quantidade = Quantidade + 1
        FROM Artigo_Academico_Comprado AS AAC
        INNER JOIN Artigo_Academico AS AA ON AAC.ID_Artigo = AA.ID
        WHERE AAC.ID_Traje = @ID_Traje;


    END;
END;
GO
```

Trigger para adicionar +1 à quantidade das Peças do Traje referentes a um traje quando este é apagado.

```
CREATE OR ALTER TRIGGER trg_delete_traje_peca
ON Traje
AFTER DELETE
AS
BEGIN
    DECLARE @ID_Traje CHAR(6);

    SELECT @ID_Traje = ID FROM deleted;

    IF @@ROWCOUNT > 0
    BEGIN
        UPDATE Peca_Do_Traje
        SET Quantidade = Quantidade + 1
        FROM Peca_Do_Traje_Comprada AS PDTC
        INNER JOIN Peca_Do_Traje AS PDT ON PDTC.ID_Peca = PDT.ID
        WHERE PDTC.ID_Traje = @ID_Traje;

    END;
END;
```

Trigger para incrementar e decrementar número de Artigos Académicos referentes a um traje quando um Artigo é comprado.

```
CREATE OR ALTER TRIGGER trg_add_remove_acessorio
ON artigo_academico_comprado
AFTER INSERT, DELETE
AS
BEGIN
    DECLARE @ID_Traje CHAR(6);
    DECLARE @ID_Artigo CHAR(6);

    -- insert case
    IF EXISTS (SELECT * FROM inserted)
    BEGIN
        SELECT @ID_Traje = ID_Traje, @ID_Artigo = ID_Artigo FROM inserted;

        -- Increment Num_Acessorios in Traje
        UPDATE Traje
        SET Num_Acessorios = Num_Acessorios + 1
        WHERE ID = @ID_Traje;

        -- Decrement Quantidade in Artigo_Academico
        UPDATE Artigo_Academico
        SET Quantidade = Quantidade - 1
        WHERE ID = @ID_Artigo;
    END

    -- delete case
    IF EXISTS (SELECT * FROM deleted)
    BEGIN
        SELECT @ID_Traje = ID_Traje, @ID_Artigo = ID_Artigo FROM deleted;

        -- Decrement Num_Acessorios in Traje
        UPDATE Traje
        SET Num_Acessorios = Num_Acessorios - 1
        WHERE ID = @ID_Traje;

        -- Increment Quantidade in Artigo_Academico
        UPDATE Artigo_Academico
        SET Quantidade = Quantidade + 1
        WHERE ID = @ID_Artigo;
    END
END;
GO
```

Trigger para incrementar e decrementar número de Peças do Traje referentes a um traje quando um Artigo é comprado.

```
CREATE OR ALTER TRIGGER trg_add_remove_peca
ON peca_do_traje_comprada
AFTER INSERT, DELETE
AS
BEGIN
    DECLARE @ID_Traje CHAR(6);
    DECLARE @ID_Peca CHAR(6);

    -- insert case
    IF EXISTS (SELECT * FROM inserted)
    BEGIN
        SELECT @ID_Traje = ID_Traje, @ID_Peca = ID_Peca FROM inserted;

        -- Increment Num_Pecas in Traje
        UPDATE Traje
        SET Num_Pecas = Num_Pecas + 1
        WHERE ID = @ID_Traje;

        -- Decrement Quantidade in Peca_do_Traje
        UPDATE Peca_do_Traje
        SET Quantidade = Quantidade - 1
        WHERE ID = @ID_Peca;
    END

    -- delete case
    IF EXISTS (SELECT * FROM deleted)
    BEGIN
        SELECT @ID_Traje = ID_Traje, @ID_Peca = ID_Peca FROM deleted;

        -- Decrement Num_Pecas in Traje
        UPDATE Traje
        SET Num_Pecas = Num_Pecas - 1
        WHERE ID = @ID_Traje;

        -- Increment Quantidade in Peca_do_Traje
        UPDATE Peca_do_Traje
        SET Quantidade = Quantidade + 1
        WHERE ID = @ID_Peca;
    END
END;
GO
```

Views

Utilizamos várias views para ajudar na apresentação dos dados.

```
CREATE OR ALTER VIEW ArtigosPapelariaView AS
SELECT ID, Nome
FROM Artigo_Papelaria;
GO

CREATE OR ALTER VIEW EmblemaDetalhesview AS
SELECT ID, Forma, Tipo
FROM Emblema;
GO

CREATE OR ALTER VIEW PinDetalhes AS
SELECT ID, Tipo
FROM Pin;
GO

CREATE OR ALTER VIEW PastaDetalhes AS
SELECT ID, Universidade
FROM Pasta;
GO

CREATE OR ALTER VIEW ChapeuDetalhes AS
SELECT ID, Universidade
FROM Chapeu;
GO

CREATE OR ALTER VIEW NoDetalhes AS
SELECT ID, Tipo, Cor1, Cor2
FROM NoA;
GO

CREATE OR ALTER VIEW ArtigoPapelariaDetalhesView AS
SELECT ID, Marca, Quantidade, End_Loja
FROM Artigo_Papelaria;
GO

CREATE OR ALTER VIEW CanetaDetalhesView AS
SELECT ID, Tipo, Cor
FROM Caneta;
GO

CREATE OR ALTER VIEW LapisDetalhesView AS
SELECT ID, Dureza
FROM Lapis;
GO
```

Stored Procedures

Utilizamos várias stored procedures para criação, atualização, remoção dos dados, nomeadamente:

Stored procedure para adicionar um chapéu, e assegurar a criação automática de um ID válido.

```
CREATE OR ALTER PROCEDURE AddChapeu
    @Nome VARCHAR(60),
    @Quantidade INT,
    @End_Loja VARCHAR(60),
    @Universidade VARCHAR(60)
AS
BEGIN
    DECLARE @LetraLoja CHAR(1);
    DECLARE @NovoID VARCHAR(6);
    DECLARE @IDExistente CHAR(6);
    DECLARE @IDBase VARCHAR(5);
    DECLARE @Count INT;
    DECLARE @Exists INT;

    -- Inicializar o contador
    SET @Count = 1;

    -- Obter a letra da loja
    SELECT @LetraLoja = Letra FROM Loja WHERE Endereco = @End_Loja;

    IF @LetraLoja IS NULL
    BEGIN
        RAISERROR('Endereço da loja não encontrado.', 16, 1);
        RETURN;
    END

    -- Gerar o ID base
    SET @IDBase = 'A' + @LetraLoja + 'C';

    WHILE 1 = 1
    BEGIN
        -- Calcular o próximo ID
        SET @NovoID = @IDBase + RIGHT('000' + CAST(@Count AS VARCHAR(3)), 3);

        -- Verificar se o ID já existe
        SELECT @Exists = COUNT(*) FROM Artigo_Academico WHERE ID = @NovoID;

        -- Se o ID não existir, sair do loop
        IF @Exists = 0
            BREAK;

        -- Incrementar o contador
        SET @Count = @Count + 1;
    END

    -- Inserir no Artigo_Academico
    INSERT INTO Artigo_Academico (ID, Nome, Quantidade, End_Loja)
    VALUES (@NovoID, @Nome, @Quantidade, @End_Loja);

    -- Inserir no Chapeu
    INSERT INTO Chapeu (ID, Universidade)
    VALUES (@NovoID, @Universidade);
END;
GO
```

Stored procedure para adicionar um emblema, e assegurar a criação automática de um ID válido.

```
-- adicionar emblema
CREATE OR ALTER PROCEDURE AddEmblema
    @Nome VARCHAR(60),
    @Quantidade INT,
    @End_Loja VARCHAR(60),
    @Forma VARCHAR(16),
    @Tipo VARCHAR(16)
AS
BEGIN
    DECLARE @LetraLoja CHAR(1);
    DECLARE @NovoID VARCHAR(6);
    DECLARE @Exists INT;
    DECLARE @Contador INT;
    DECLARE @IDBase VARCHAR(5);


    SET @Contador = 1;
    -- Obter a letra da loja
    SELECT @LetraLoja = Letra FROM Loja WHERE Endereco = @End_Loja;

    IF @LetraLoja IS NULL
    BEGIN
        RAISERROR('Endereço da loja não encontrado.', 16, 1);
        RETURN;
    END

    -- Gerar o ID base
    SET @IDBase = 'A' + @LetraLoja + 'E';

    WHILE 1 = 1
    BEGIN
        -- Calcular o próximo ID
        SET @NovoID = @IDBase + RIGHT('000' + CAST(@Contador AS VARCHAR(3)), 3);

        -- Verificar se o ID já existe
        SELECT @Exists = COUNT(*) FROM Artigo_Academico WHERE ID = @NovoID;

        -- Se o ID não existir, sair do loop
        IF @Exists = 0
            BREAK;

        -- Incrementar o contador
        SET @Contador = @Contador + 1;
    END

    -- Inserir no Artigo_Academico
    INSERT INTO Artigo_Academico (ID, Nome, Quantidade, End_Loja)
    VALUES (@NovoID, @Nome, @Quantidade, @End_Loja);

    -- Inserir no Emblema
    INSERT INTO Emblema (ID, Forma, Tipo)
    VALUES (@NovoID, @Forma, @Tipo);
END;
GO
```

Stored procedure para adicionar um pin, e assegurar a criação automática de um ID válido.

```
-- adicionar pin
CREATE OR ALTER PROCEDURE AddPin
    @Nome VARCHAR(60),
    @Quantidade INT,
    @End_Loja VARCHAR(60),
    @Tipo VARCHAR(16)
AS
BEGIN
    DECLARE @LetraLoja CHAR(1);
    DECLARE @NovoID VARCHAR(6);
    DECLARE @Exists INT;
    DECLARE @Contador INT = 1;
    DECLARE @IDBase VARCHAR(5);

    -- Obter a letra da loja
    SELECT @LetraLoja = Letra FROM Loja WHERE Endereco = @End_Loja;

    IF @LetraLoja IS NULL
    BEGIN
        RAISERROR('Endereço da loja não encontrado.', 16, 1);
        RETURN;
    END

    -- Gerar o ID base
    SET @IDBase = 'A' + @LetraLoja + 'P';

    WHILE 1 = 1
    BEGIN
        -- Calcular o próximo ID
        SET @NovoID = @IDBase + RIGHT('000' + CAST(@Contador AS VARCHAR(3)), 3);

        -- Verificar se o ID já existe
        SELECT @Exists = COUNT(*) FROM Artigo_Academico WHERE ID = @NovoID;

        -- Se o ID não existir, sair do loop
        IF @Exists = 0
            BREAK;

        -- Incrementar o contador
        SET @Contador = @Contador + 1;
    END

    -- Inserir no Artigo_Academico
    INSERT INTO Artigo_Academico (ID, Nome, Quantidade, End_Loja)
    VALUES (@NovoID, @Nome, @Quantidade, @End_Loja);

    -- Inserir no Pin
    INSERT INTO Pin (ID, Tipo)
    VALUES (@NovoID, @Tipo);
END;
GO
```

Stored procedure para adicionar uma pasta, e assegurar a criação automática de um ID válido.

```
-- adicionar pasta
CREATE OR ALTER PROCEDURE AddPasta
    @Nome VARCHAR(60),
    @Quantidade INT,
    @End_Loja VARCHAR(60),
    @Universidade VARCHAR(32)
AS
BEGIN
    DECLARE @LetraLoja CHAR(1);
    DECLARE @NovoID VARCHAR(6);
    DECLARE @Exists INT;
    DECLARE @Contador INT = 1;
    DECLARE @IDBase VARCHAR(5);

    -- Obter a letra da loja
    SELECT @LetraLoja = Letra FROM Loja WHERE Endereco = @End_Loja;

    IF @LetraLoja IS NULL
    BEGIN
        RAISERROR('Endereço da loja não encontrado.', 16, 1);
        RETURN;
    END

    -- Gerar o ID base
    SET @IDBase = 'A' + @LetraLoja + 'T';

    WHILE 1 = 1
    BEGIN
        -- Calcular o próximo ID
        SET @NovoID = @IDBase + RIGHT('000' + CAST(@Contador AS VARCHAR(3)), 3);

           -- Verificar se o ID já existe
        SELECT @Exists = COUNT(*) FROM Artigo_Academico WHERE ID = @NovoID;

        -- Se o ID não existir, sair do loop
        IF @Exists = 0
            BREAK;

        -- Incrementar o contador
        SET @Contador = @Contador + 1;
    END

    -- Inserir no Artigo_Academico
    INSERT INTO Artigo_Academico (ID, Nome, Quantidade, End_Loja)
    VALUES (@NovoID, @Nome, @Quantidade, @End_Loja);

   -- Inserir na Pasta
    INSERT INTO Pasta (ID, Universidade)
    VALUES (@NovoID, @Universidade);
END;
GO
```

Stored procedure para adicionar um nó, e assegurar a criação automática de um ID válido.

```
CREATE OR ALTER PROCEDURE AddNo
    @Nome VARCHAR(60),
    @Quantidade INT,
    @End_Loja VARCHAR(60),
    @Cor1 VARCHAR(16),
    @Cor2 VARCHAR(16),
    @Tipo VARCHAR(16)
AS
BEGIN
    DECLARE @LetraLoja CHAR(1);
    DECLARE @NovoID VARCHAR(6);
    DECLARE @Exists INT;
    DECLARE @Contador INT = 1;
    DECLARE @IDBase VARCHAR(5);


    -- Obter a letra da loja
    SELECT @LetraLoja = Letra FROM Loja WHERE Endereco = @End_Loja;

    IF @LetraLoja IS NULL
    BEGIN
        RAISERROR('Endere◆o da loja n◆o encontrado.', 16, 1);
        RETURN;
    END

    -- Gerar o ID base
    SET @IDBase = 'A' + @LetraLoja + 'N';

    WHILE 1 = 1
    BEGIN
        -- Calcular o pr◆ximo ID
        SET @NovoID = @IDBase + RIGHT('000' + CAST(@Contador AS VARCHAR(3)), 3);

        -- Verificar se o ID j◆ existe
        SELECT @Exists = COUNT(*) FROM Artigo_Academico WHERE ID = @NovoID;

        -- Se o ID n◆o existir, sair do loop
        IF @Exists = 0
            BREAK;

        -- Incrementar o contador
        SET @Contador = @Contador + 1;
    END

    -- Verificar se o contador ultrapassou o limite
    IF @Contador > 999
    BEGIN
        RAISERROR('N◆o foi poss◆vel gerar um novo ID. Limite atingido.', 16, 1);
        RETURN;
    END

    -- Inserir no Artigo_Academico
    INSERT INTO Artigo_Academico (ID, Nome, Quantidade, End_Loja)
    VALUES (@NovoID, @Nome, @Quantidade, @End_Loja);

    -- Inserir no NoA
    INSERT INTO NoA (ID, Cor1, Cor2, Tipo)
    VALUES (@NovoID, @Cor1, @Cor2, @Tipo);
END;
```

Stored procedure para adicionar um lápis, e assegurar a criação automática de um ID válido.

```
CREATE OR ALTER PROCEDURE AddLapis
    @Nome VARCHAR(60),
    @Quantidade INT,
    @Marca VARCHAR(60),
    @End_Loja VARCHAR(60),
    @Dureza VARCHAR(2)

AS
BEGIN
    DECLARE @LetraLoja CHAR(1);
    DECLARE @NovoID VARCHAR(6);
    DECLARE @IDExistente CHAR(6);
    DECLARE @IDBase VARCHAR(5);
    DECLARE @Count INT;
    DECLARE @Exists INT;

    -- Inicializar o contador
    SET @Count = 1;

    -- Obter a letra da loja
    SELECT @LetraLoja = Letra FROM Loja WHERE Endereco = @End_Loja;

    IF @LetraLoja IS NULL
    BEGIN
        RAISERROR('Endere◆o da loja n◆o encontrado.', 16, 1);
        RETURN;
    END

    -- Gerar o ID base
    SET @IDBase = 'P' + @LetraLoja + 'L';

    WHILE 1 = 1
    BEGIN
        -- Calcular o pr◆ximo ID
        SET @NovoID = @IDBase + RIGHT('000' + CAST(@Count AS VARCHAR(3)), 3);

        -- Verificar se o ID j◆ existe
        SELECT @Exists = COUNT(*) FROM Artigo_Papelaria WHERE ID = @NovoID;

        -- Se o ID n◆o existir, sair do loop
        IF @Exists = 0
            BREAK;

        -- Incrementar o contador
        SET @Count = @Count + 1;
    END

    -- Inserir no Artigo
    INSERT INTO Artigo_Papelaria (ID, Nome, Quantidade, Marca, End_Loja)
    VALUES (@NovoID, @Nome, @Quantidade, @Marca, @End_Loja);

    -- Inserir no Lapis
    INSERT INTO Lapis (ID, Dureza)
    VALUES (@NovoID, @Dureza);
END;
```

Stored procedure para adicionar uma caneta, e assegurar a criação automática de um ID válido.

```
    -- adicionar uma caneta
CREATE OR ALTER PROCEDURE AddCaneta
    @Nome VARCHAR(60),
    @Quantidade INT,
    @Marca VARCHAR(60),
    @End_Loja VARCHAR(60),
    @Cor VARCHAR(16),
    @Tipo VARCHAR(30)

AS
BEGIN
    DECLARE @LetraLoja CHAR(1);
    DECLARE @NovoID VARCHAR(6);
    DECLARE @IDExistente CHAR(6);
    DECLARE @IDBase VARCHAR(5);
    DECLARE @Count INT;
    DECLARE @Exists INT;

    -- Inicializar o contador
    SET @Count = 1;

    -- Obter a letra da loja
    SELECT @LetraLoja = Letra FROM Loja WHERE Endereco = @End_Loja;

    IF @LetraLoja IS NULL
    BEGIN
        RAISERROR('Endereço da loja não encontrado.', 16, 1);
        RETURN;
    END

    -- Gerar o ID base
    SET @IDBase = 'P' + @LetraLoja + 'C';

    WHILE 1 = 1
    BEGIN
        -- Calcular o próximo ID
        SET @NovoID = @IDBase + RIGHT('000' + CAST(@Count AS VARCHAR(3)), 3);

        -- Verificar se o ID já existe
        SELECT @Exists = COUNT(*) FROM Artigo_Papelaria WHERE ID = @NovoID;

        -- Se o ID não existir, sair do loop
        IF @Exists = 0
            BREAK;

        -- Incrementar o contador
        SET @Count = @Count + 1;
    END

    -- Inserir no Artigo
    INSERT INTO Artigo_Papelaria (ID, Nome, Quantidade, Marca, End_Loja)
    VALUES (@NovoID, @Nome, @Quantidade, @Marca, @End_Loja);

    -- Inserir na caneta
    INSERT INTO Caneta (ID, Tipo, Cor)
    VALUES (@NovoID, @Tipo, @Cor);
END;
GO
```

Stored procedures para adicionar um traje, e assegurar a criação automática de um ID válido, e associar Artigos Académicos e Peças do Traje a um Traje.

```
CREATE OR ALTER PROCEDURE AddTraje
    @Nome VARCHAR(60),
    @End_Loja VARCHAR(60)
AS
BEGIN
    DECLARE @NewID CHAR(6);
    DECLARE @Count INT;
    DECLARE @Exists INT;

    -- Inicializar o contador
    SET @Count = 1;

    -- Loop para encontrar um ID dispon?vel
    WHILE 1 = 1
    BEGIN
        -- Calcular o pr?ximo ID
        SET @NewID = 'T' + RIGHT('0000' + CAST(@Count AS VARCHAR(5)), 5);

        -- Verificar se o ID j? existe
        SELECT @Exists = COUNT(*) FROM Traje WHERE ID = @NewID;

        -- Se o ID n?o existir, sair do loop
        IF @Exists = 0
            BREAK;

        -- Incrementar o contador
        SET @Count = @Count + 1;
    END

    -- Inserir o novo traje
    INSERT INTO Traje (ID, Nome, Num_Acessorios, Num_Pecas, Completo, End_Loja)
    VALUES (@NewID, @Nome, 0, 0, 0, @End_Loja);
END;
GO

--procedure para comprar peca e ela ser adicionada ao traje
-- ela depois ? adicionada por causa do trigger
CREATE OR ALTER PROCEDURE ComprarPeca
    @IDPECA VARCHAR(6),
    @IDTRAJE VARCHAR(6)
AS
BEGIN
    INSERT INTO peca_do_traje_comprada (ID_peca, ID_traje)
    VALUES (@IDPECA, @IDTRAJE);

END;
GO
--procedure para ocmprar um artigo academico e adicionar ao traje
-- ele depois ? adicionado por causa do trigger
CREATE OR ALTER PROCEDURE ComprarArtigo
    @IDARTIGO VARCHAR(6),
    @IDTRAJE VARCHAR(6)
AS
BEGIN
    -- Inserir no Artigo_Papelaria
    INSERT INTO artigo_academico_comprado (ID_artigo, ID_traje)
    VALUES (@IDARTIGO, @IDTRAJE);

END;
```

Stored procedures para apagar Artigos Académicos, Artigos de Papelaria, Peças do Traje. Procedure para alterar os detalhes de uma peça do traje.

```
---- PARA DELETAR
CREATE OR ALTER PROCEDURE DeleteArtigo
    @IDARTIGO VARCHAR(6)
AS
BEGIN
    delete from artigo_academico where id=@IDARTIGO
END;
GO
-- depois o trigger trata de remover do traje (se for o caso)


CREATE OR ALTER PROCEDURE DeletePapelaria
    @IDPAPEL VARCHAR(6)
AS
BEGIN
    delete from artigo_papelaria where id=@IDPAPEL
END;
GO


CREATE OR ALTER PROCEDURE DeletePeca
    @IDPECA VARCHAR(6)
AS
BEGIN
    delete from peca_do_traje where id=@IDPECA
END;
GO
-- depois o trigger trata de remover do traje (se for o caso)


CREATE PROCEDURE AlterarPecaDoTraje
    @pecaID CHAR(6),
    @novoNome VARCHAR(60),
    @novoTamanho VARCHAR(3),
    @novaUniversidade VARCHAR(32),
    @novaQuantidade INT,
    @novoGenero VARCHAR(1)
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE Peca_do_Traje
    SET Nome = @novoNome,
        Tamanho = @novoTamanho,
        Universidade = @novaUniversidade,
        Quantidade = @novaQuantidade,
        Genero = @novoGenero
    WHERE ID = @pecaID;
END;
GO
```

Além disso, também temos stored procedures que retornam os dados de vários objetos da base de dados e que permitem a pesquisa por nome.

Interface

A nossa interface consiste em 4 tabs.

Na primeira, estão presentes os Artigos Académicos. Aqui, é possível filtrar por nome, filtrar por loja, tipo de artigo, ,etc., adicionar novos artigos à BD, remover artigos da BD e adicionar artigos a um traje já criado, e ver os detalhes dos artigos selecionados.

A segunda tab é referente aos Artigos de Papelaria, e nela é possível também filtrar por nome, por loja, tipo de artigo, etc., adicionar novos artigos à BD, e remover artigos à BD, e ver os detalhes dos artigos selecionados.

A terceira tab é referente a Peças do Traje, e nela é possível filtrar por nome, remover peças da BD e alterar peças já existentes na BD, tal como adicionar peças a um traje já criado, e ver os detalhes das peças selecionadas.

A quarta tab é referente aos Trajes, e nela é possível criar um novo traje, remover trajes existentes, ver as peças e artigos associados a cada traje e removê-las do traje.

Os processos de adição, alterações e remoção estão feitos tal que a interface seja atualizada para evidenciar as mudanças.

Tentamos fazer a interface intuitiva e responsiva, com pop-ups a indicar quando os objetos são adicionados, removidos e alterados, e com avisos para prevenção de erro, tal como, por exemplo, quando a quantidade de um certo artigo não é um número, ou quando qualquer campo não está preenchido, quando se cria um novo artigo.

Conclusão

Com este projeto, foi possível verificar na prática como funciona um SGBD.

Além disso, aprofundamos os nossos conhecimentos em C# e na criação e manipulação de interfaces gráficas, apesar de não nos termos aprofundado muito nesse aspecto.

Em geral, tentamos alcançar todos os objetivos que tínhamos proposto inicialmente, mas tivemos que mudar um pouco o esquema do trabalho, para a sua simplificação.

Em suma, este projeto consolidou o aprendizado teórico através da aplicação prática, evidenciando a importância de um SGBD para a gestão eficiente e segura das informações.