universidade de aveiro

deti departamento de eletrónica, telecomunicações e informática

PetroManager

Base de dados de gestão de um posto de combustível

Universidade de Aveiro

Licenciatura em Engenharia Informática

Licenciatura em Engenharia de Automação Industrial

Base de Dados P3G1

Regente: Prof. Carlos Costa

Diogo Sampaio, 112641

Diogo Costa, 112714

Índice

Introdução

O nosso projeto teve como tema a gestão de postos de combustível de uma gasolineira. O sistema que desenvolvemos permite a administração de vários postos de combustível dessa mesma companhia

Na sua interface será possível ao utilizador adicionar e atualizar postos de combustível, registar e editar clientes, registar e atualizar informações de funcionários, inserir e atualizar transações, e gerir preços de combustíveis.

Análise de Requisitos

Um funcionário é caracterizado por um ID único, email, nome, salário e contacto.

Cada funcionário está associado a um posto, identificado pelo ID do posto (ID Posto).

Um posto é caracterizado por um ID único, hora de abertura, endereço, hora de fecho e contacto.

Uma bomba é caracterizada por um ID único e cada bomba está associada a um depósito, identificado pelo ID do depósito (ID Depósito), e a um combustível, identificado pelo ID do combustível (ID Combustível).

Um cliente é caracterizado por um NIF (número de identificação fiscal), contacto, nome e idade.

Um combustível é caracterizado por um ID único e nome.

Uma transação é caracterizada por um ID único, data da transação, hora, quantidade de litros e valor total.

Cada transação está associada a um cliente (identificado pelo ID Cliente), a um funcionário (identificado pelo ID Funcionário), e a uma bomba (identificada pelo ID Bomba).

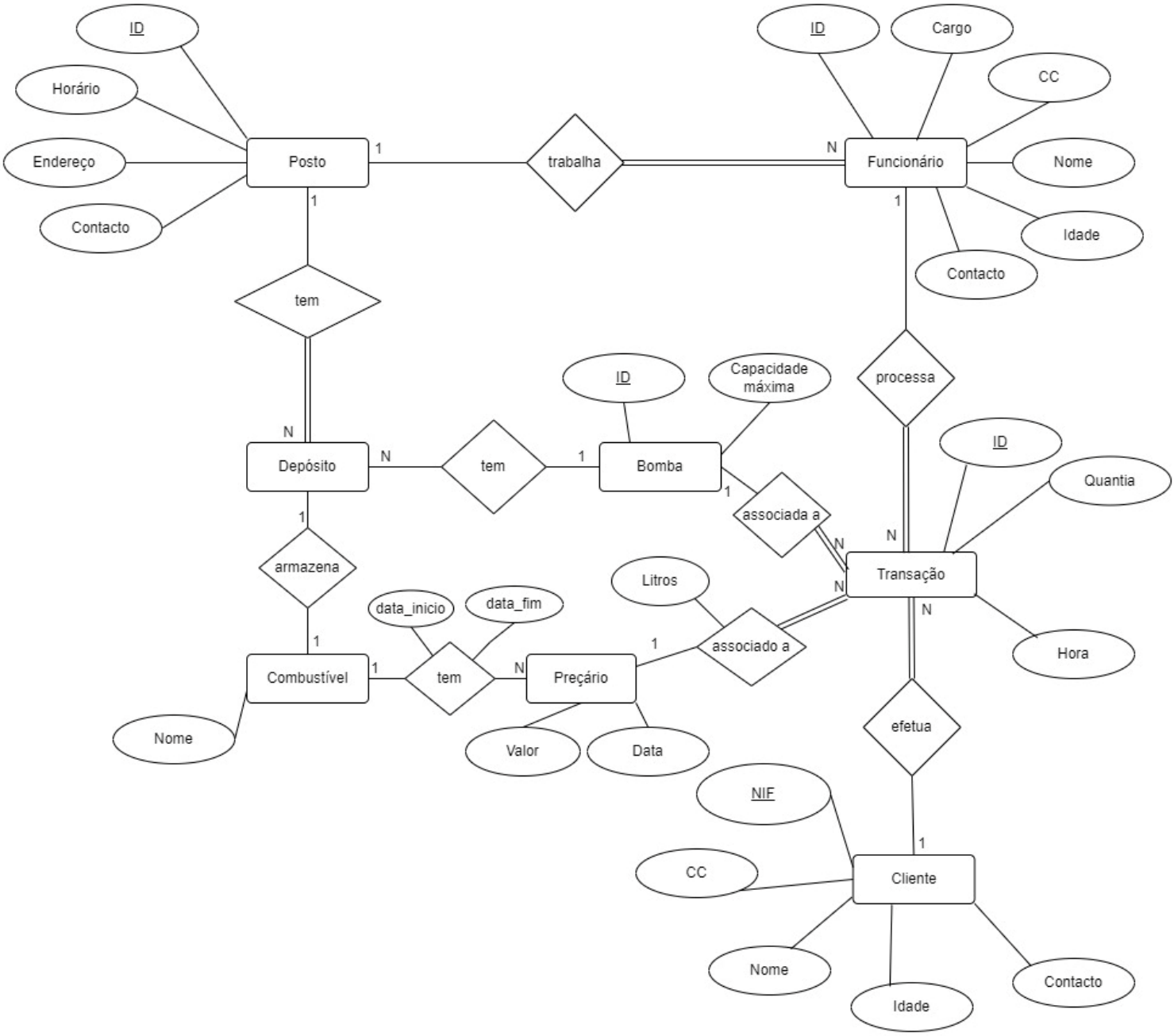
Um depósito é caracterizado por um ID único, capacidade máxima e capacidade atual e cada depósito está associado a um posto (identificado pelo ID Posto).

Um preçário é caracterizado pelo ID do combustível (ID Combustível), preço, data de início (Data Inicio) e data de fim (Data Fim).

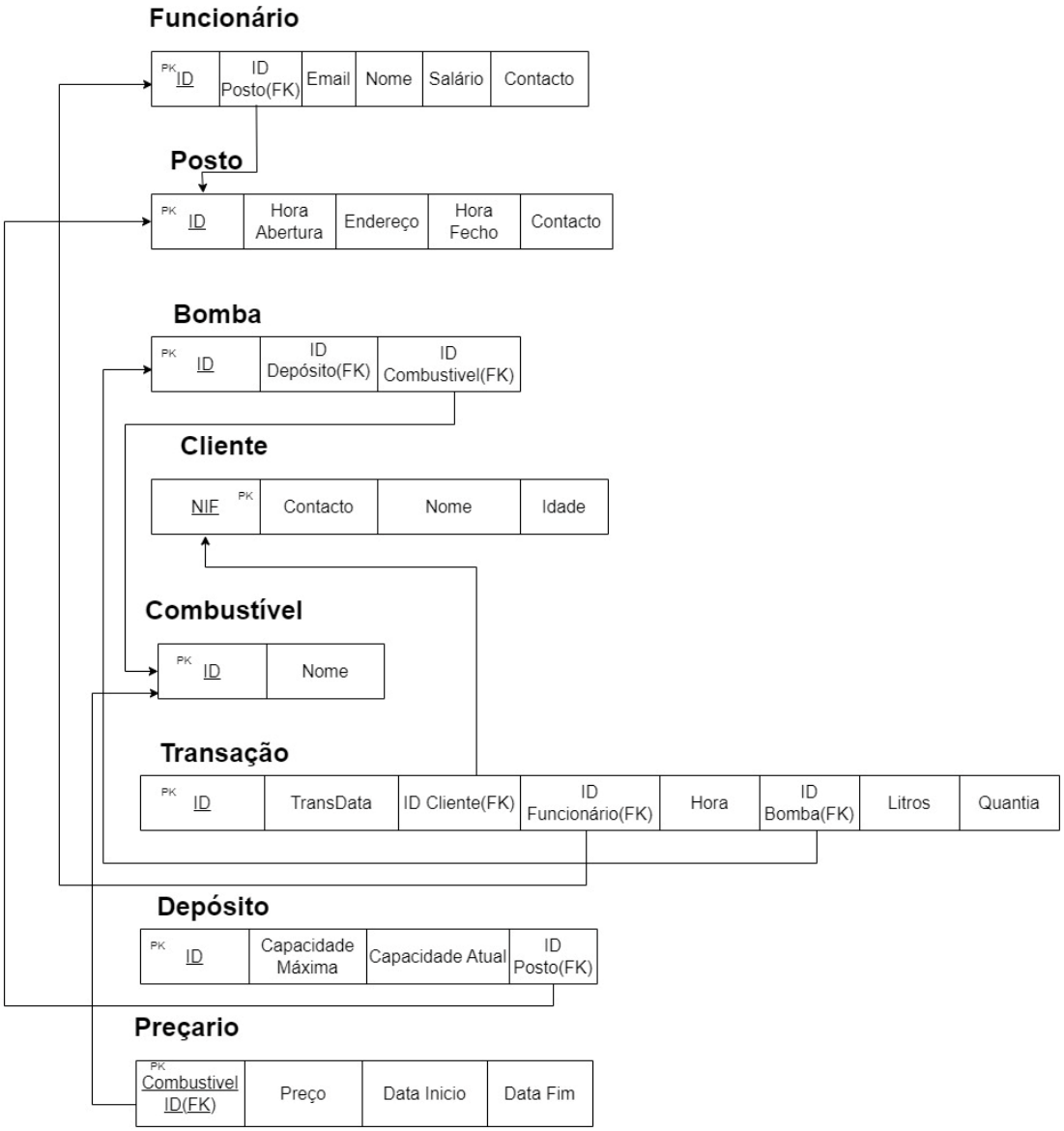
Os utilizadores serão capazes de interagir com o sistema da seguinte forma:

- Consultar/editar/apagar todas as informações de cada posto.
- Consultar/editar/apagar todas as informações de cada funcionário, podendo recorrer a filtros que auxiliam a consulta de informações.
- Consultar/editar/apagar todas as informações de cada cliente, podendo recorrer a filtros que auxiliam a consulta de informações.
- Consultar/editar/apagar todas as informações de cada transação efetuada por um cliente, sendo possível filtrar tendo em conta o cliente que a efetuou ou o funcionário responsável pela mesma.
- Consultar e atualizar os preços de cada combustível para uma determinada semana.

Diagrama Entidade Relação (DER)



Esquema Relacional



DDL

```
ALTER TABLE PM.Transacao DROP CONSTRAINT IF EXISTS fk_Transacao_Cliente;
ALTER TABLE PM.Transacao DROP CONSTRAINT IF EXISTS fk_Transacao_Bomba;
ALTER TABLE PM.Transacao DROP CONSTRAINT IF EXISTS fk_Transacao_Funcionario;

ALTER TABLE PM.Precario DROP CONSTRAINT IF EXISTS fk_Precario_Combustivel;

ALTER TABLE PM.Bomba DROP CONSTRAINT IF EXISTS fk_Bomba_Combustivel;
ALTER TABLE PM.Bomba DROP CONSTRAINT IF EXISTS fk_Bomba_Deposito;

ALTER TABLE PM.Deposito DROP CONSTRAINT IF EXISTS fk_Deposito_Posto;

ALTER TABLE PM.Funcionario DROP CONSTRAINT IF EXISTS fk_Funcionario_Posto;

ALTER TABLE PM.Posto DROP CONSTRAINT IF EXISTS fk_Posto_Manager;

DROP TABLE IF EXISTS PM.Transacao;
DROP TABLE IF EXISTS PM.Precario;
DROP TABLE IF EXISTS PM.Bomba;
DROP TABLE IF EXISTS PM.Deposito;
DROP TABLE IF EXISTS PM.Combustivel;
DROP TABLE IF EXISTS PM.Cliente;
DROP TABLE IF EXISTS PM.Funcionario;
DROP TABLE IF EXISTS PM.Posto;

CREATE TABLE PM.Posto (
    ID INT PRIMARY KEY NOT NULL,
    Cidade VARCHAR(50),
    Contacto INT,
    Hora_Abertura TIME,
    Hora_Fecho TIME,
    MgrID INT
);

CREATE TABLE PM.Funcionario(
    ID INT PRIMARY KEY NOT NULL,
    Nome VARCHAR(25) NOT NULL,
    Salario FLOAT,
    Contacto INT,
    Email VARCHAR(50),Não foi encontrada nenhuma entrada de índice.
    ID_Posto INT
);

CREATE TABLE PM.Cliente(
    NIF INT PRIMARY KEY,
    Nome VARCHAR(25) NOT NULL,
    Contacto INT,
    Idade INT
);

CREATE TABLE PM.Combustivel(
    ID INT PRIMARY KEY NOT NULL,
    Nome VARCHAR(9) NOT NULL CHECK (Nome IN('Diesel','Gasolina','GPL'))
);
CREATE TABLE PM.Deposito(
    ID INT PRIMARY KEY NOT NULL,
    CapacidadeMax INT NOT NULL,
    CapacidadeAtual INT,
    ID_Posto INT
```

```
);

CREATE TABLE PM.Bomba(
    ID INT PRIMARY KEY NOT NULL,
    IDCombustivel INT,
    IDDeposito INT,
       PostoID int
);
CREATE TABLE PM.Precario(
    CombustivelID INT NOT NULL,
    Preco FLOAT,
    DataInicio DATE,
    DataFim DATE,
    PRIMARY KEY (CombustivelID, DataInicio)
);
CREATE TABLE PM.Transacao(
    ID INT  IDENTITY (1,1) PRIMARY KEY,
    TransData DATE,
    Litros INT,
    IDCombustivel INT,
    Quantia FLOAT,
    NIFCliente INT,
    IDFunc INT
);



ALTER TABLE PM.Deposito
    ADD CONSTRAINT fk_Deposito_Posto FOREIGN KEY (ID_Posto)
    REFERENCES PM.Posto(ID) ON DELETE SET NULL ON UPDATE CASCADE;

ALTER TABLE PM.Bomba
    ADD CONSTRAINT fk_Bomba_Combustivel FOREIGN KEY (IDCombustivel)
    REFERENCES PM.Combustivel(ID) ON DELETE SET NULL ON UPDATE CASCADE;

ALTER TABLE PM.Bomba
    ADD CONSTRAINT fk_Bomba_Deposito FOREIGN KEY (IDDeposito)
    REFERENCES PM.Deposito(ID) ON DELETE SET NULL ON UPDATE CASCADE;

ALTER TABLE PM.Bomba
       ADD CONSTRAINT fk_Bomba_PostoID FOREIGN KEY (PostoID)
       REFERENCES PM.Posto(ID);

ALTER TABLE PM.Precario
    ADD CONSTRAINT fk_Precario_Combustivel FOREIGN KEY (CombustivelID)
    REFERENCES PM.Combustivel(ID) ON UPDATE CASCADE;

ALTER TABLE PM.Transacao
    ADD CONSTRAINT fk_Transacao_Cliente FOREIGN KEY (NIFCliente)
    REFERENCES PM.Cliente(NIF) ON UPDATE CASCADE;


ALTER TABLE PM.Transacao
    ADD CONSTRAINT fk_Transacao_Funcionario FOREIGN KEY (IDFunc)
    REFERENCES PM.Funcionario(ID) ON UPDATE CASCADE;


ALTER TABLE PM.Posto
    ADD CONSTRAINT fk_Posto_Manager FOREIGN KEY (MgrID)
    REFERENCES PM.Funcionario(ID) ON DELETE SET NULL ON UPDATE CASCADE;
```

DML

```
INSERT INTO PM.Funcionario (ID, Nome, Salario, Contacto, Email, ID_Posto) VALUES
(001, 'Joao Silva', 1500.00, 912345678, 'joao.silva@PetroManager.com', 001),
(002, 'Ana Santos', 1600.00, 923456789, 'ana.santos@PetroManager.com', 001),
(003, 'Pedro Costa', 1550.00, 934567890, 'pedro.costa@PetroManager.com', 001),
(004, 'Marta Ferreira', 1700.00, 945678901, 'marta.ferreira@PetroManager.com',
001),
(005, 'Carlos Sousa', 1600.00, 956789012, 'carlos.sousa@PetroManager.com', 001),
(006, 'Sofia Oliveira', 1550.00, 912345678, 'sofia.oliveira@PetroManager.com',
002);


INSERT INTO PM.Posto (ID, Cidade, Contacto, Hora_Abertura, Hora_Fecho, MgrID)
VALUES
(001, 'Lisboa', 213456789, '08:00', '20:00',04),
(002, 'Porto', 223456789, '09:00', '21:00',08),
(003, 'Coimbra', 239876543, '08:30', '19:30',012),
(004, 'Braga', 253456789, '07:00', '22:00', 018),
(005, 'Faro', 289654321, '08:00', '18:00',025);


INSERT INTO PM.Combustivel (ID, Nome) VALUES
(1, 'Diesel'),
(2, 'Gasolina'),
(3, 'GPL');


INSERT INTO PM.Deposito (ID, CapacidadeMax, CapacidadeAtual, ID_Posto) VALUES
(0101, 50000, 40000, 001),
(0102, 50000, 40000, 001),
(0103, 50000, 40000, 001),
(0201, 50000, 40000, 002),
(0202, 50000, 40000, 002),
(0203, 50000, 40000, 002),
(0301, 50000, 40000, 003);


INSERT INTO PM.Bomba(ID,IDCombustivel,IDDeposito,PostoID) VALUES
(0101,1,0101,001),
(0102,2,0102,001),
(0103,3,0103,001),
(0201,1,0201,002),
(0202,2,0202,002),
(0203,3,0203,002),
(0301,1,0301,003);


INSERT INTO PM.Cliente (NIF, Nome, Contacto, Idade) VALUES
(123456789, 'João Silva', 912345678, 35),
(987654321, 'Maria Oliveira', 918765432, 28),
(456123789, 'Carlos Santos', 915678234, 42),
(789456123, 'Ana Martins', 913456789, 31),
(321654987, 'Pedro Ferreira', 919876543, 47),
(654987321, 'Rita Costa', 917654321, 22),
```

Stored Procedures (SP)

No nosso projeto recorremos bastante ao uso de Stored Procedures para a adição, remoção e edição de dados, visto que estes apresentam uma elevada performance devido ao seu armazenamento em cache e pela sua atomicidade.

Lista de Stored Procedures usados:

- PM.RegistarNovoCliente: Regista um novo cliente, realizando verificações do NIF, idade e contacto antes do registo.
- PM.UpdateCliente: Atualiza os dados de um cliente existente, verificando se o NIF e o contacto estão nos intervalos permitidos e se a idade é superior a 18 anos.
- PM.DeleteCliente: Remove um cliente e todas as suas transações associadas.
- PM.updatePrice: Atualiza o preço de um combustível, encerrando o preço atual e inserindo um novo registo com o preço atualizado.
- PM.AdicionarPosto: Adiciona um novo posto de combustível verificando a existência do ID do gerente na tabela de funcionários e a validade de todos os outros dados.
- PM.UpdatePostoDetails: Atualiza os detalhes de um posto existente, incluindo cidade, contacto, horário de abertura e fecho, e ID do gerente.
- PM.DeletePosto: Remove um posto de combustível, incluindo todas as transações, funcionários, bombas e depósitos associados.
- PM.DeleteFunc: Remove um funcionário e todas as suas transações associadas.
- PM.AddFunc: Adiciona um novo funcionário ao sistema.
- PM.UpdateFunc: Atualiza as informações de um funcionário existente.
- PM.UpdateTransacao: Atualiza uma transação existente, garantindo a existência do cliente e do funcionário relacionados.
- PM.AddTransacao: Adiciona uma nova transação, validando a existência do cliente e do funcionário e arredondando a quantia para duas casas decimais.
- PM.DeleteTransacao: Remove uma transação específica, após verificar a sua existência.

User Defined Function (UDF)

Para facilitar a consulta de dados no nosso projeto, decidimos usar as UDF´s fazendo com que a consulta de dados seja mais simplificada para o utilizador fornecendo principalmente filtragem de dados.

Alguns exemplos de UDF´s usadas:

```
CREATE FUNCTION PM.FilterFuncByCity
(
    @Cidade NVARCHAR(100)
)
RETURNS TABLE
AS
RETURN
(
    SELECT *
    FROM PM.FuncDisplay
    WHERE PCidade LIKE '%' + @Cidade + '%'
);


CREATE FUNCTION PM.FilterFuncByName
(
    @Nome NVARCHAR(100)
)
RETURNS TABLE
AS
RETURN
(
    SELECT *
    FROM PM.FuncDisplay
    WHERE Nome LIKE '%' + @Nome + '%'
);

CREATE FUNCTION PM.FilterClienteByName
(
    @Nome NVARCHAR(100)
)
RETURNS TABLE
AS
RETURN
(
    SELECT *
    FROM PM.Cliente
    WHERE Nome LIKE '%' + @Nome + '%'
);

CREATE FUNCTION PM.FilterTransacaoByFuncName
```

```
(
    @Nome NVARCHAR(100)
)
RETURNS TABLE
AS
RETURN
(
    SELECT *
    FROM PM.TransacaoDetalhada
    WHERE Funcionario_Nome LIKE '%' + @Nome + '%'
);

CREATE FUNCTION PM.FilterTransacaoByClienteName
(
    @Nome NVARCHAR(100)
)
RETURNS TABLE
AS
RETURN
(
    SELECT *
    FROM PM.TransacaoDetalhada
    WHERE Cliente_Nome LIKE '%' + @Nome + '%');
```

Views

De modo a simplificar o acesso aos dados presente na base de dados neste projeto utilizamos views para simplificar a complexidade das tabelas facilitando assim a apresentação dos dados na interface.

Alguns exemplos de Views usadas:

```
drop view IF exists clientsID ;

CREATE VIEW PM.clientsID AS
SELECT
    *
FROM
    Cliente;

drop view if exists PostosList      ;


CREATE VIEW PM.PostosList AS
SELECT
    *
       FROM
    Posto;

DROP VIEW IF EXISTS TransacoesID;

CREATE VIEW PM.TransacoesID AS
SELECT
    ID
```

```
        FROM
    Transacao;

drop view if exists PM.managerList;

CREATE VIEW PM.managerList AS
SELECT f.Nome AS 'Nome', f.Email AS 'Email',f.Contacto as 'Contacto',f.ID_Posto as
'PostoID', p.Cidade AS 'Cidade',p.Contacto as 'PostoContacto', p.Hora_Abertura as
'Abertura', p.Hora_Fecho as 'Fecho'
FROM Funcionario f
INNER JOIN Posto p ON f.ID = p.MgrID;


drop view if exists capacidadesDepositos

CREATE VIEW PM.capacidadesDepositos AS
SELECT
    p.ID AS Posto,
        d.ID AS IDDeposito,
    d.CapacidadeMax AS CapacidadeMaxima,
    d.CapacidadeAtual AS CapacidadeAtual,
    c.Nome AS Combustivel
FROM Posto p
LEFT JOIN Deposito d ON p.ID = d.id_posto
LEFT JOIN Bomba b ON d.ID = b.IDDeposito
LEFT JOIN Combustivel c ON b.IDCombustivel = c.ID;


CREATE VIEW PM.lastIDFunc AS
SELECT
       MAX(ID) AS 'LastID' from PM.Funcionario

select * from PM.lastIDFunc


CREATE VIEW PM.lastIDPosto AS
SELECT
       MAX(ID) AS 'LastID' from PM.Posto

CREATE VIEW PM.LastIDTransacao AS
SELECT
       MAX(ID) AS 'LastID' from PM.Transacao

CREATE VIEW PM.managersID AS
SELECT
              p.ID as 'PID',
              f.ID as 'FID'
       from Funcionario f
INNER JOIN Posto p ON f.ID = p.MgrID;



drop view if exists PM.FuncDisplay

CREATE VIEW PM.FuncDisplay AS
SELECT
  f.ID AS 'ID',
  f.Nome AS 'Nome',
  f.Contacto AS 'Contacto',
  f.Email AS 'Email',
  f.Salario AS 'Salario',
  p.ID AS 'PID',
  p.Cidade AS 'PCidade'
FROM PM.Funcionario f
INNER JOIN PM.Posto p ON f.ID_Posto = p.ID
```

```sql
drop view if exists PM.CityID;

CREATE VIEW PM.CityID AS

SELECT ID, Cidade from Posto

select * from PM.CityID

CREATE VIEW PM.CustomerTotalSpent AS
SELECT c.NIF AS Customer_NIF, SUM(t.Quantia) AS Total_Spent
FROM PM.Cliente c
JOIN PM.Transacao t ON c.NIF = t.NIFCliente
GROUP BY c.NIF, c.Nome;

DROP VIEW PM.TransacaoDetalhada

CREATE VIEW PM.TransacaoDetalhada AS
SELECT
    t.ID AS Transacao_ID,
    t.TransData AS Data,
    t.IDCombustivel AS Combustivel_ID,
    t.Litros AS Quantidade,
    p.Preco AS Preco_Combustivel,
    t.Quantia AS Total_Compra,
    c.Nome AS Cliente_Nome,
    c.Contacto AS Cliente_Contacto,
    c.NIF AS Cliente_NIF,
    f.Nome AS Funcionario_Nome,
       f.ID AS Funcionario_ID
FROM
    PM.Transacao t
JOIN
    PM.Cliente c ON t.NIFCliente = c.NIF
JOIN
    PM.Funcionario f ON t.IDFunc = f.ID
JOIN     PM.Precario p ON t.IDCombustivel = p.CombustivelID
AND t.TransData BETWEEN p.DataInicio AND p.DataFim;
```

Indexes

De modo a melhorar o desempenho na procura de dados na base de dados decidimos usar indexers. Eles permitem que o sistema de base de dados localize rapidamente as linhas de interesse sem a necessidade de percorrer a tabela inteira, tornando as operações de leitura mais eficientes e rápidas.

Alguns exemplos de indexers usados:

```sql
CREATE INDEX idx_Cliente_NIF ON PM.Cliente(NIF);
CREATE INDEX idx_Cliente_Nome ON PM.Cliente(Nome);
CREATE INDEX idx_Cliente_Contacto ON PM.Cliente(Contacto);
CREATE INDEX idx_Cliente_Idade ON PM.Cliente(Idade);


CREATE UNIQUE INDEX idx_Combustivel_ID ON PM.Combustivel(ID);
CREATE INDEX idx_Combustivel_Nome ON PM.Combustivel(Nome);
```

```sql
CREATE UNIQUE INDEX idx_Deposito_ID ON PM.Deposito(ID);
CREATE INDEX idx_Deposito_ID_Posto ON PM.Deposito(ID_Posto);
CREATE INDEX idx_Deposito_CapacidadeMax ON PM.Deposito(CapacidadeMax);
CREATE INDEX idx_Deposito_CapacidadeAtual ON PM.Deposito(CapacidadeAtual);


CREATE UNIQUE INDEX idx_Bomba_ID ON PM.Bomba(ID);
CREATE INDEX idx_Bomba_IDCombustivel ON PM.Bomba(IDCombustivel);
CREATE INDEX idx_Bomba_IDDeposito ON PM.Bomba(IDDeposito);
CREATE INDEX idx_Bomba_PostoID ON PM.Bomba(PostoID);


CREATE INDEX idx_Precario_CombustivelID ON PM.Precario(CombustivelID);
CREATE INDEX idx_Precario_DataInicio ON PM.Precario(DataInicio);
CREATE INDEX idx_Precario_Preco ON PM.Precario(Preco);
CREATE INDEX idx_Precario_DataFim ON PM.Precario(DataFim);


CREATE INDEX idx_Transacao_TransData ON PM.Transacao(TransData);
CREATE INDEX idx_Transacao_IDCombustivel ON PM.Transacao(IDCombustivel);
CREATE INDEX idx_Transacao_NIFCliente ON PM.Transacao(NIFCliente);
CREATE INDEX idx_Transacao_IDFunc ON PM.Transacao(IDFunc);
CREATE INDEX idx_Transacao_Litros ON PM.Transacao(Litros);
CREATE INDEX idx_Transacao_Quantia ON PM.Transacao(Quantia);



CREATE INDEX idx_Posto_ID ON PM.Posto(ID);
CREATE INDEX idx_Posto_Cidade ON PM.Posto(Cidade);
CREATE INDEX idx_Posto_Contacto ON PM.Posto(Contacto);
CREATE INDEX idx_Posto_Hora_Abertura ON PM.Posto(Hora_Abertura);
CREATE INDEX idx_Posto_Hora_Fecho ON PM.Posto(Hora_Fecho);
CREATE INDEX idx_Posto_MgrID ON PM.Posto(MgrID);


CREATE UNIQUE INDEX idx_Funcionario_ID ON PM.Funcionario(ID);
CREATE INDEX idx_Funcionario_ID_Posto ON PM.Funcionario(ID_Posto);
CREATE INDEX idx_Funcionario_Nome ON PM.Funcionario(Nome);
CREATE INDEX idx_Funcionario_Salario ON PM.Funcionario(Salario);
CREATE INDEX idx_Funcionario_Contacto ON PM.Funcionario(Contacto);
CREATE INDEX idx_Funcionario_Email ON PM.Funcionario(Email);
```

Triggers

Só recorremos ao uso de um trigger para a verificação de dados pois incluímos esta nos próprios SP's ou no próprio DDL.

```
CREATE TRIGGER trg_DeleteFuncBeforeDelete
ON PM.Funcionario
FOR DELETE
AS
BEGIN
    DECLARE @funcID INT;

    SELECT @funcID = ID FROM deleted;

    IF @funcID IS NOT NULL
    BEGIN
        DELETE FROM PM.Transacao
        WHERE IDFunc = @funcID;
    END
END;
```

Conclusão

Concluindo, desenvolvemos uma solução para a gestão de postos de combustíveis, garantindo a integridade e a eficiência nas operações de manipulação de dados.

Podemos então dizer que os objetivos principais do projeto foram alcançados com sucesso.