universidade de aveiro

deti departamento de eletrónica, telecomunicações e informática

Área de Serviço

Base de Dados para Gestão de uma Área de Serviço

Universidade de Aveiro

Licenciatura em Engenharia de Computadores e Informática

Bases de Dados

P1G8

João Gaspar, 107708

Guilherme Santos, 107961

Índice

Introdução

Este relatório apresenta o projeto de uma base de dados para a gestão de uma área de serviço, desenvolvido como parte da unidade curricular de Bases de Dados da Licenciatura em Engenharia de Computadores e Informática da Universidade de Aveiro.

O objetivo deste projeto é criar uma base de dados que possa gerir todas as operações de uma área de serviço, incluindo a gestão de estacionamentos, lojas, stocks, produtos, funcionários e gerentes.

Neste relatório, descrevemos em detalhes o processo de análise de requisitos, o design das entidades e suas multiplicidades, o Diagrama Entidade Relação (DER), o Esquema Relacional (ER), bem como a implementação de Stored Procedures (SPs), Views, User-Defined Functions (UDFs), Triggers e Indexes. Cada seção do relatório concentra-se numa parte específica do projeto, fornecendo uma visão detalhada de como cada componente foi projetado e implementado.

De modo a ser possível a testagem da nossa interface noutra base de dados, é necessário aceder ao ficheiro DataBaseConnection.cs e mudar os atributos initialCatalog, uid e password para as credenciais desejadas.

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Diagnostics;

namespace form
{
    public static class DatabaseConnection
    {
        private static string dataSource = "tcp:mednat.ieeta.pt\\SQLSERVER,8101";
        private static string initialCatalog = "p1g8";
        private static string uid = "p1g8";
        private static string password = "IrineuSonic28g*";

        public static string connectionString = $"Data Source = {dataSource}; Initial Catalog = {initialCatalog}; uid= {uid}; password = {password}";

        public static SqlConnection GetConnection()
        {
            return new SqlConnection(connectionString);
        }
    }
}
```

Figura 1 - Ficheiro DataBaseConnection.cs onde é possível alterar a conexão à base de dados.

Em anexo segue todo o código usado tanto para a interface como para a base de dados, a apresentação em .pdf e o vídeo demonstrativo.

Análise de Requisitos

Podem-se registar na interface gerentes, identificados distintamente pelo seu NIF e contacto bem como o número da área de serviço em que atuam.

- O sistema deve ser capaz de gerenciar as áreas de serviço, que são identificadas por um número único. Cada área de serviço tem uma localização e um nome.
- Existem 2 tipos de funcionários: gerentes, funcionários.
- Para um gerente registar a entrada de um novo produto no estoque, é necessário que o produto já esteja registado no sistema.
- O gerente pode fazer encomendas a fornecedores de produtos que estejam em falta.
- O gerente pode demitir funcionários ou então adicionar novos.
- O gerente pode verificar os produtos que tem em stock nas lojas.

Entidades e Multiplicidade

- Area_Servico: Identificada pelo número da área (Num_Area). Possui atributos não únicos como localização e nome da área.
- EstacionamentoON: Identificado pelo número do estacionamento (Num_Estacionamento). Possui atributos não únicos como capacidade e taxa. Relaciona-se com a entidade Area_Servico.
- Loja: Identificada pelo ID. Possui atributo não único como nome. Relaciona-se com a entidade Area_Servico.
- Estoque: Identificado pelo número do estoque (Num_Estoque). Possui atributo não único como quantidade. Relaciona-se com a entidade Loja.
- Produto: Identificado pelo código (Codigo). Possui atributos não únicos como taxa de IVA, quantidade e preço. Relaciona-se com a entidade Estoque.
- BombaCombustivel: Identificada pelo número da bomba (Num_Bomba). Possui atributos não únicos como marca e preço. Relaciona-se com a entidade Produto.
- Pessoa: Identificada pelo NIF. Possui atributos não únicos como nome, contato e número da área de serviço. Relaciona-se com a entidade Area_Servico.

- Fornecedor: Identificado pelo NIF. Possui atributo não único como marca.
- Cliente: Identificado pelo NIF. Possui atributo não único como classificação de crédito.
- Gerente: Identificado pelo NIF. Possui atributos não únicos como gabinete e senha.
- Funcionario: Identificado pelo NIF. Possui atributos não únicos como cargo. Relaciona-se com a entidade Gerente.
- Encomenda: Identificada pelo número da encomenda (Num_Encomenda). Possui atributo não único como data de entrega. Relaciona-se com as entidades Fornecedor e Gerente.
- Item: Identificado pelo ID do item (ItemID). Possui atributo não único como quantidade. Relaciona-se com as entidades Produto e Encomenda.

Diagrama Entidade Relação (DER)

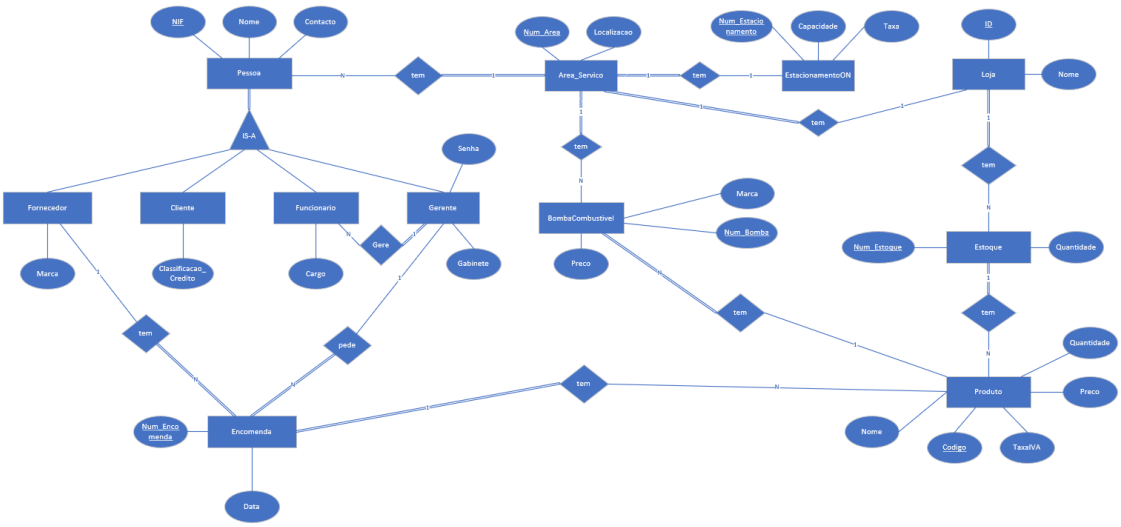

Figura 2 - Diagrama Entidade Relação.

Esquema Relacional (ER)

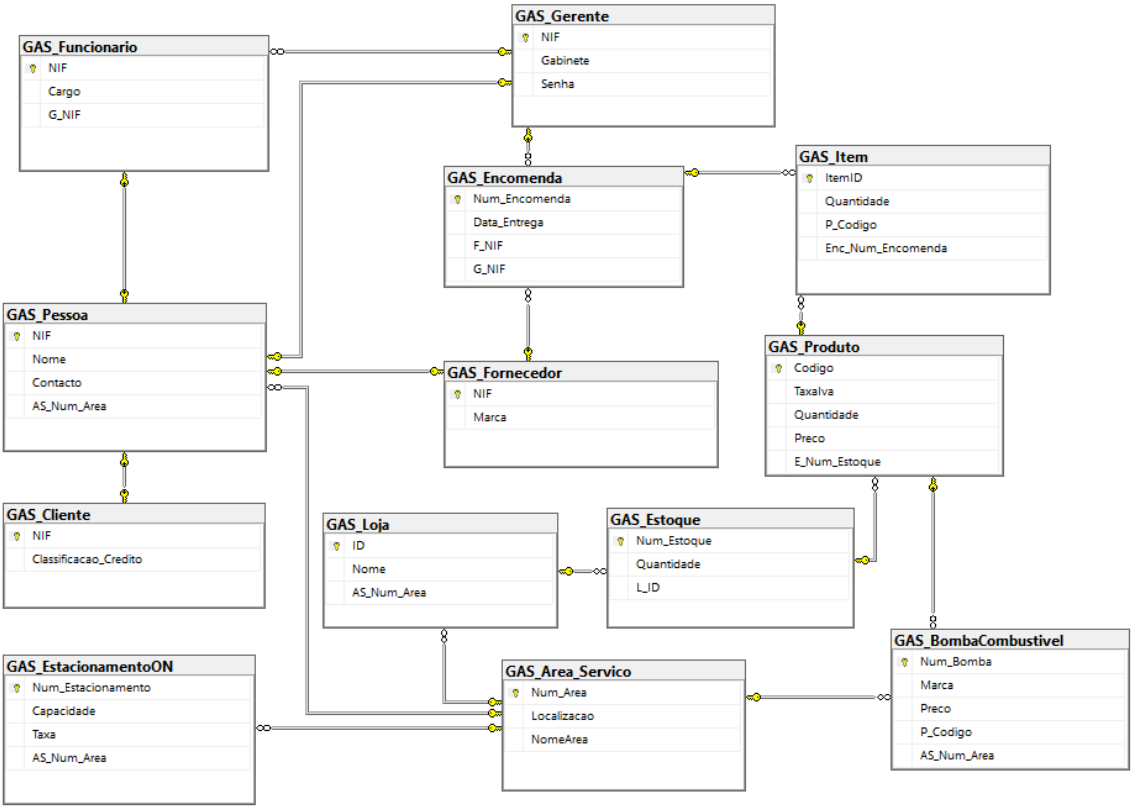

Figura 3 – *Esquema Relacional*.

Stored Procedure (SP)

Foram criadas vários Stored Procedures para suportar várias funções dentro do sistema, entre elas:

- Inserir um estacionamento;
- Inserir uma loja;
- Inserir um funcionário;
- Registar um novo gerente;
- Verificar o gerente no login;
- Atualizar o gerente atual;
- Procurar encomendas por NIF;
- Procurar detalhes de uma encomenda;

- Criar um pedido e Itens.

Neste documento iremos explicar o funcionamento de uma das funcionalidades que temos, neste caso, a função de registar um novo gerente.

Para registar um gerente nós usamos o SP "RegisterUser" onde tem como parâmetros de entrada: o número, localização e nome da área de serviço; o NIF, nome, contacto, gabinete e senha do gerente. O SP começa por iniciar uma transação, onde insere primeiro os dados da área de serviço na tabela "GAS_Area_Servico", seguidamente insere os dados principais do gerente (NIF, nome, contacto e número da área de serviço) na tabela "GAS_Pessoa" e finalmente insere os restantes dados do gerente (NIF, gabinete e senha) na tabela "GAS_Gerente". É importante notar que a senha inserida na tabela é um valor encriptado originado da função "EncryptByPassPhrase", que usa uma frase chave para encriptar um determinado valor (para obter o valor original, é necessário usar a função "DecryptByPassPhrase" com a mesma frase chave usada na encriptação). Caso seja detetado algum erro durante a inserção dos valores, é executado um rollback da transição.

```
-- Registar um novo gerente
CREATE PROCEDURE dbo.RegisterUser
    @Num_Area INT,
    @Localizacao NVARCHAR(100),
    @NomeArea NVARCHAR(100),
    @NIF int,
    @NomePessoa NVARCHAR(100),
    @Contacto int,
    @Gabinete NVARCHAR(50),
    @Senha NVARCHAR(50)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        INSERT INTO dbo.GAS_Area_Servico (Num_Area, Localizacao, NomeArea)
        VALUES (@Num_Area, @Localizacao, @NomeArea);

        INSERT INTO dbo.GAS_Pessoa (NIF, Nome, Contacto, AS_Num_Area)
        VALUES (@NIF, @NomePessoa, @Contacto, @Num_Area);

        INSERT INTO dbo.GAS_Gerente (NIF, Gabinete, Senha)
        VALUES (@NIF, @Gabinete, EncryptByPassPhrase('WhySoSerious', @Senha));

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        THROW;
    END CATCH
```

Figura 4 – SP para registar um novo gerente.

View

Foram definidas views para obter todos os detalhes dos:

- Itens;
- Funcionários;
- Estoques;
- Bombas de combustível;
- Encomendas;
- Áreas de serviço;
- Pessoas;
- Fornecedores;
- Lojas;
- Produtos;
- Gerentes.

Podemos observar por exemplo a criação de uma view para obter os detalhes completo do item na imagem abaixo.

```
-- View para obter detalhes completos do item
CREATE VIEW vw_Item_Detalhes AS
SELECT i.ItemID, i.Quantidade, p.Codigo, p.TaxaIva, p.Preco, e.Num_Encomenda, e.Data_entrega
FROM GAS_Item i
JOIN GAS_Produto p ON i.P_Codigo = p.Codigo
JOIN GAS_Encomenda e ON i.Enc_Num_Encomenda = e.Num_Encomenda;
GO
```

Figura 5 – *View para obter detalhes completos do item*.

User-Defined Function (UDF)

Foram criados UDFs para:

- Obter o número de bombas de gasolina de uma área de serviço;
- Obter a capacidade total de um estacionamento de uma área de serviço.

- Obter o total de funcionários por gerente.
- Obter os nomes das lojas de um gerente.
- Verificar se a senha do gerente está correta.

Na imagem abaixo é possível ver a criação da UDF para obter a capacidade total de um estacionamento na área de serviço.

```
-- Obter a capacidade total de um estacionamento de uma área de serviço
CREATE FUNCTION dbo.GetCapacidadeTotalEstacionamento(@AS_Num_Area int)
RETURNS int
AS
BEGIN
    DECLARE @CapacidadeTotal int;

    SELECT @CapacidadeTotal = SUM(Capacidade)
    FROM GAS_EstacionamentoON
    WHERE AS_Num_Area = @AS_Num_Area;

    RETURN ISNULL(@CapacidadeTotal, 0);
END;
GO
```

Figura 6 – *UDF para obter a capacidade total de um estacionamento.*

Trigger

Foram criados triggers para:

- Verificar se a quantidade de um produto não é negativa;
- Verificar se a capacidade de um estacionamento não é negativa;
- Verificar se a quantidade de um produto está abaixo do limite.

Podemos observar por exemplo o trigger que verifica se a quantidade de um produto está abaixo do limite, onde verifica os valores atualizados do produto (se existem produtos suficientes), identifica os responsáveis (fornecedor e gerente), e se os responsáveis estiverem disponíveis, cria uma nova encomenda e adiciona os itens necessários. Caso contrário, a transação é cancelada e um erro é gerado, assegurando que o processo só prossiga quando todas as condições forem atendidas.

```sql
-- Criação do trigger que verifica se a quantidade de um produto está abaixo do limite
CREATE TRIGGER AutoOrderProduct
ON GAS_Produto
FOR UPDATE
AS
BEGIN
    DECLARE @CodigoProduto INT;
    DECLARE @QuantidadeAtual INT;
    DECLARE @LimiteQuantidade INT = 10;
    DECLARE @NumEncomenda INT;
    DECLARE @DataEntrega DATE = DATEADD(day, 7, GETDATE());
    DECLARE @FornecedorNIF INT;
    DECLARE @GerenteNIF INT;

    SELECT @CodigoProduto = Codigo, @QuantidadeAtual = Quantidade FROM inserted;

    IF @QuantidadeAtual < @LimiteQuantidade
    BEGIN
        SELECT @FornecedorNIF = NIF FROM GAS_Fornecedor WHERE NIF IN (
            SELECT NIF FROM GAS_Pessoa WHERE AS_Num_Area IN (
                SELECT AS_Num_Area FROM GAS_Produto WHERE Codigo = @CodigoProduto));
        SELECT @GerenteNIF = NIF FROM GAS_Gerente WHERE NIF IN (
            SELECT NIF FROM GAS_Pessoa WHERE AS_Num_Area IN (
                SELECT AS_Num_Area FROM GAS_Produto WHERE Codigo = @CodigoProduto));

        IF @FornecedorNIF IS NOT NULL AND @GerenteNIF IS NOT NULL
        BEGIN
            INSERT INTO GAS_Encomenda (Data_Entrega, F_NIF, G_NIF)
            VALUES (@DataEntrega, @FornecedorNIF, @GerenteNIF);

            SET @NumEncomenda = SCOPE_IDENTITY();

            INSERT INTO GAS_Item (Quantidade, P_Codigo, Enc_Num_Encomenda)
            VALUES (@LimiteQuantidade - @QuantidadeAtual, @CodigoProduto, @NumEncomenda);
        END
        ELSE
        BEGIN
            RAISERROR ('Fornecedor ou gerente não disponível para processar a encomenda.', 16, 1);
            ROLLBACK TRANSACTION;
        END
    END
END;
GO
```

Figura 7 – Trigger que verifica se a quantidade de um produto está abaixo do limite.

Index

Neste projeto não tivemos necessidade de criar nenhum index.

Conclusão

Podemos concluir com o estado final do projeto que todos os requerimentos, definidos anteriormente, da aplicação de gestão de áreas de serviço foram implementados com sucesso.

Uma adição que fizemos depois da apresentação presencial do projeto foi a implementação da encriptação e decriptação das senhas dos gerentes.

Uma modificação que se pode acrescentar num possível futuro desenvolvimento do projeto seria criar indexes para aumentar a performance de leitura da aplicação.