UNIVERSIDADE
DE TRÁS-OS-MONTES
E ALTO DOURO

TRABALHO EXPERIMENTAL 1

Relatório

Universidade de Trás-os-Montes e Alto-Douro

Licenciatura em Engenharia Informática

Técnicas Avançadas de Base de Dados

Docentes:

António Marques

Paulo Martins

Discentes:

Ana Dias al69691

Diana Alves al68557

Diana Ferreira al68938

Rui Vaz al68565

Índice

1. Introdução

No âmbito da unidade curricular de Técnicas Avançadas de Base de Dados, foi proposto aos alunos a elaboração de um Trabalho Experimental que fosse de encontro com todo o conteúdo lecionado ao longo do semestre.

Assim, a primeira etapa do mesmo, incide na definição de políticas de segurança e acesso aos dados e na resolução de problemas de concorrência no sistema de base de dados, numa Base de Dados previamente elaborada, em outra unidade curricular, Laboratório de Aplicações Web e de Base de Dados.

Portanto, para tornar processo realidade, foi usada a ferramenta Microsoft SQL Server Management Studio 18.

Segue em anexo (Anexo A – Base de Dados e Anexo B – Procedures e Políticos de Segurança e Acesso aos Dados) todo o código referente a esta realização do Trabalho Experimental.

2. Enquadramento Teórico

No contexto das bases de dados, e em termos de distribuição, é possível definir algumas configurações distintas:

- Sistemas centralizados (dados e processamentos centralizados);
- Arquitetura cliente/servidor (dados centralizados e processamentos distribuídos);
- Sistemas distribuídos (dados e processamentos distribuídos).

Os níveis de isolamento transacionais têm a ver com a forma como é gerido o mecanismo de controle de concorrência.

Deste modo, é necessário criar utilizadores em que cada um tem um papel/role para com a base de dados (**User Defined Roles**).

Caso todos os utilizadores possam aceder a todos os dados, não é necessário definir o papel de cada pessoa, pois na criação de um utilizador a base de dados já dá um role por defeito (**Fixed Database Roles**).

Para aceder a certas informações da base de dados, é necessário ver se o utilizador as pode cessar ou não. Isso acontece devido às permissões. Estas são definidas para todos os utilizadores e irão ditar o acesso às tabelas por parte de cada um.

Assim o comando para deixar um usuário aceder às informações é chamado de **GRANT** (concede permissões). O comando contrário é o **DENY** (nega permissões). Para remover as permissões GRANT ou DENY usa-se o **REVOKE**. Assim, essas transações necessitam de salvaguardar a integridade dos dados seguindo quatro propriedades:

- ***Atomicidade***: todas as ações correspondentes a uma transação devem ser concluídas com sucesso (**COMMITTED**), caso contrário a transação falha (**ROLLBACK**);
- ***Consistência***: todas os campos descritos na base de dados devem ser respeitados;
- ***Isolamento***: cada transação funciona de forma separada das outras;
- ***Durabilidade***: resultados de uma transação deve ser permanente.

Devido a estas propriedades e como é necessário gerir o mecanismo de controle de concorrência, existe a necessidade de definir níveis de isolamento transacional. Esses níveis são:

- ***Read Uncommitted***: ao submeter um comando quer ele seja SELECT, DELETE ou UPDATE e caso sejam feitas transações nesse momento, este nível irá incluir nos comandos essa informação;
- ***Read Committed***: nível de isolamento padrão, ignorando dados ainda não submetidos ou transações feitas simultaneamente à consulta;
- ***Repeatable Read***: garante que a mesma leitura de uma ação se repita na mesma transação;
- ***Serializable***: semelhante ao *Repeatable Read* mas com a restrição que a informação selecionada não pode ser alterada ou lida por outra transação, até que a primeira seja lida.

Apesar destes 4 níveis de isolamento, é possível haver ações indesejadas em transações feitas de modo simultâneo, sendo essas:

- ***Dirty Read***: é quando uma conexão pode ler informações onde ainda não foi efetuado "commit", ou seja, a informação lida pode já não existir ou ter sido modificada.
- ***Nonrepeatable Read***: é quando na execução de uma transação se pode ler a informação mais que uma vez diferente, ou seja, na primeira leitura é lida uma informação e na segunda e demais podem ser lidas outras informações, não sendo assim garantida a consistência da informação dentro da mesma transação.
- ***Phanton Read***: é quando na execução de uma transação podem ser inseridos ou apagados registos. Por exemplo, entre a leitura e a atualização de dados, estes podem ter saído ou entrado na cláusula WHERE, podendo ter sido inseridos ou apagados.

Nível de Isolamento	*Dirty Read*	*Nonrepeatable Read*	*Phanton Read*
Read Uncommitted	Sim	Sim	Sim
Read Commited	Não	Sim	Sim
Repeatable Read	Não	Não	Sim
Serializable	Não	Não	Não

Tabela 1 - Níveis de Isolamento Transacionais

Comando para ativar o nível de isolamento:

- SET TRANSACTION ISOLATION LEVEL [nivel_de_isolamento]

3. Objetivos do Trabalho Experimental

Através do protocolo fornecido pelos docentes da unidade curricular, estes definiram alguns objetivos que devem ser cumpridos com a resolução dos trabalhos experimentais, ao longo do semestre. Assim, estes objetivos reúnem-se em:

- Definir políticas de segurança e acesso a dados centralizados;
- Resolver problemas de concorrência em sistemas de bases de dados centralizadas;
- Desenvolver um sistema de bases de dados distribuídas;
- Definir políticas de segurança e acesso a dados distribuídos;
- Resolver problemas de concorrência em sistemas de gestão de bases de dados distribuídas;
- Análise de otimização de questões distribuídas;
- Definir modelos de sincronização de bases de dados.

Desse modo, e relativamente ao relatório do Trabalho Experimental 1, os objetivos resumem-se em:

- Relatório detalhado da execução do trabalho;
- Políticas de segurança e acesso a dados centralizados;
- Resolução para os problemas de concorrência no sistema de bases de dados centralizada;

4. Desenvolvimento

4.1. Diagrama de Base de Dados

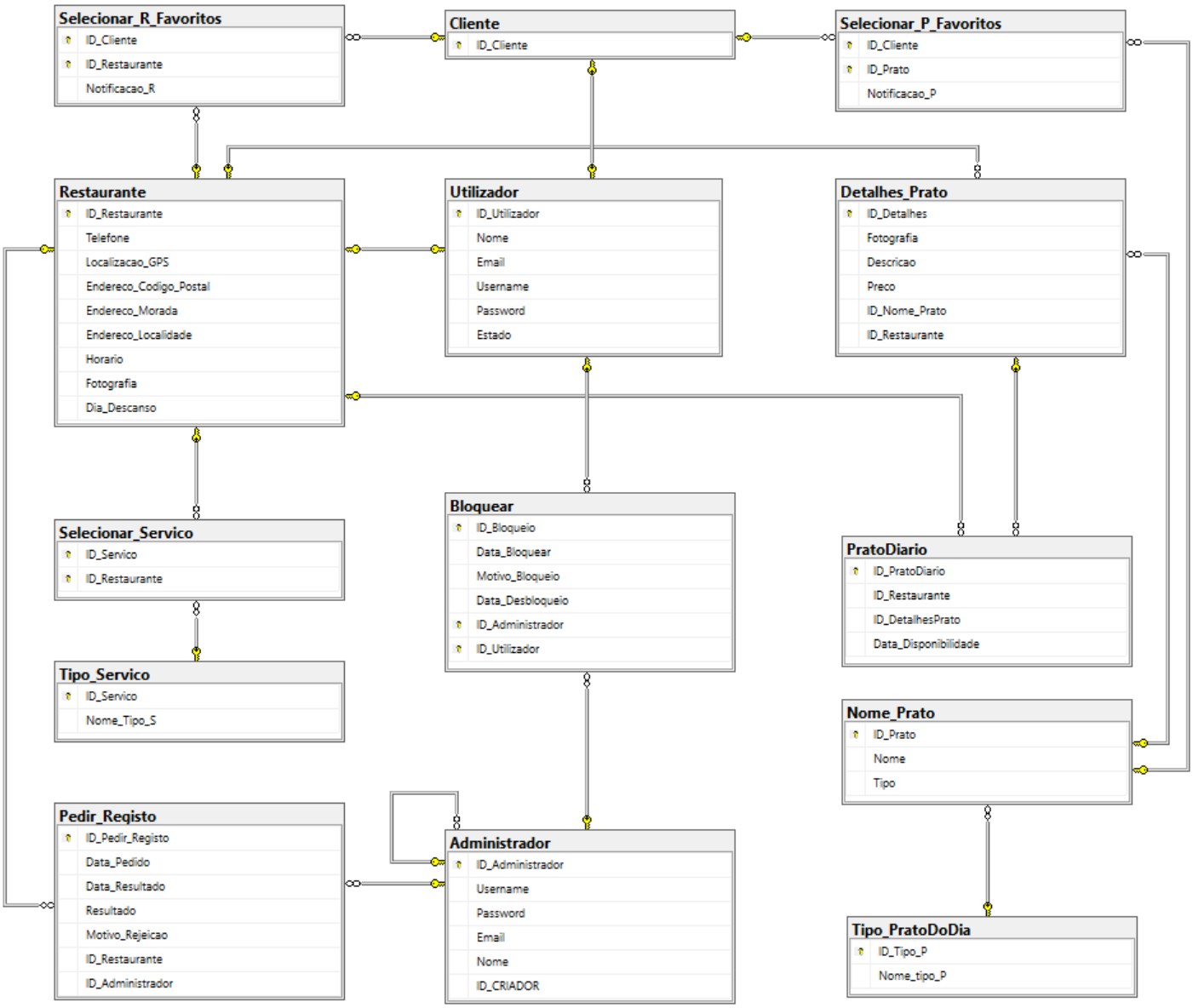

Figura 1 - Diagrama da Base de Dados

4.2. Contextualização

Neste trabalho experimental, foi reutilizada uma Base de Dados previamente elaborada, em outra unidade curricular, Laboratório de Aplicações Web e de Base de Dados. Desse modo, inicialmente começou-se por implementar todas as políticas de segurança e de acesso aos dados.

No decorrer da elaboração do trabalho, deparamo-nos com a existência de atores no sistema, sendo eles Administrador, Utilizadores, com dois tipos distintos (Cliente e Restaurante) e o Visitante. Estes apresentam permissões características do seu role.

- **Administrador**: este ator poderá criar novos administradores, bloquear e consequentemente desbloquear utilizadores, ver pedidos de registos de restaurantes, para além de poder também alterar os seus dados pessoais.
- **Cliente**: este será um ator que poderá alterar dados do seu registo, ou seja, tudo o que tenha a ver com dados pessoais, bem como selecionar favoritos, quer prato do dia, quer restaurante.
- **Restaurante**: este ator poderá registar pratos do dia, bem como, caso já exista, reaproveitar, alterando apenas algumas características; além disso, tal como os restantes atores, este pode alterar os seus dados.
- **Visitante**: este será um ator onde será ser possível consultar a informação dos restaurantes, bem como consultar os pratos do dia dos mesmos; além disso, este também poderá se registar, ou seja, inserir dados em tabelas, tais como: Utilizador, Restaurante ou Cliente. Caso se registe como restaurante, este tem que efetuar um pedido de registo que mais tarde será validado pelo administrador.

4.3.Código SQL

Para a realização desta primeira fase, foi reutilizada uma Base de Dados, que se encontra em Anexo a este PDF (Anexo A). Os outros ficheiros (Procedures) também se encontram em Anexo a este PDF (Anexo B).

Conforme essa Base de Dados, foram introduzidas Políticas de Segurança de Acesso a Dados, através da implementação de Logins, Roles e Users; implementadas Permissões; e por último, implementados Procedures, para a resolução de problemas de concorrência no sistema de Base de Dados centralizada. Todas as implementações vão ser apresentadas e esclarecidas ainda neste subcapítulo.

Políticas de Segurança de Acesso aos Dados - Logins:

```
CREATE LOGIN ADMINISTRADOR WITH PASSWORD='12345'
CREATE LOGIN CLIENTE WITH PASSWORD='12345'
CREATE LOGIN RESTAURANTE WITH PASSWORD='12345'
```

Figura 2 - Políticas de Segurança - Login

Para cada tipo de Utilizador (Administrador, Cliente e Restaurante) foram criados os seus devidos Logins com Password, uma vez que cada um tem acesso aos dados de forma personalizada. Neste caso, não foi implementado o Login para o Visitante, uma vez que não necessita.

Políticas de Segurança de Acesso aos Dados - Users:

```
CREATE USER ADMINISTRADOR1 FOR LOGIN ADMINISTRADOR
CREATE USER CLIENTE1 FOR LOGIN CLIENTE
CREATE USER RESTAURANTE1 FOR LOGIN RESTAURANTE
CREATE USER VISITANTE1 WITHOUT LOGIN
```

Figura 3 - Políticas de Segurança - Users

Em relação aos Users, estes foram implementados para cada tipo de Login (Administrador, Cliente e Restaurante), uma vez que cada um tem acesso aos dados de forma personalizada. Neste caso, foi implementado também o User do Visitante sem o Login, uma vez que este tem acesso à parte pública dos dados.

Políticas de Segurança de Acesso aos Dados - Roles:

```
CREATE ROLE ADMINISTRADORES
CREATE ROLE CLIENTES
CREATE ROLE RESTAURANTES
CREATE ROLE VISITANTES
ALTER ROLE ADMINISTRADORES ADD MEMBER ADMINISTRADOR1
ALTER ROLE CLIENTES ADD MEMBER CLIENTE1
ALTER ROLE RESTAURANTES ADD MEMBER RESTAURANTE1
ALTER ROLE VISITANTES ADD MEMBER VISITANTE1
```

Figura 4 - Políticas de Segurança - Roles

Em relação aos Roles, estes foram implementados para cada tipo de User e Login (Administrador, Cliente e Restaurante, Visitante), associando assim as políticas de acesso entre cada User e Login. Assim, será através destes Roles que as políticas de acesso serão aplicadas e testadas.

Permissões:

Foram implementadas na Base de Dados permissões para garantir o acesso de dados específicos de cada Role.

Visitantes

```
GRANT SELECT, INSERT ON Restaurante TO VISITANTES
GRANT SELECT ON Utilizador(ID_Utilizador, Nome, Email, Estado) TO VISITANTES
GRANT SELECT, INSERT ON Selecionar_Servico TO VISITANTES
GRANT SELECT ON Tipo_Servico TO VISITANTES
GRANT SELECT ON Nome_Prato TO VISITANTES
GRANT SELECT ON Tipo_PratoDoDia TO VISITANTES
GRANT SELECT ON Detalhes_Prato TO VISITANTES
GRANT SELECT ON PratoDiario TO VISITANTES

GRANT INSERT ON Utilizador TO VISITANTES
GRANT INSERT ON Cliente TO VISITANTES
GRANT INSERT ON Pedir_Registo TO VISITANTES
```

Figura 5 - Permissões dos Visitantes

Clientes

```
GRANT SELECT ON Cliente TO CLIENTES
GRANT SELECT ON Restaurante TO CLIENTES
GRANT SELECT ON Selecionar_Servico TO CLIENTES
GRANT SELECT ON Tipo_Servico TO CLIENTES
GRANT SELECT ON Utilizador TO CLIENTES
GRANT UPDATE ON Utilizador(ID_Utilizador, Nome, Email, Username, Password) TO CLIENTES
GRANT SELECT ON Nome_Prato TO CLIENTES
GRANT SELECT ON Tipo_PratoDoDia TO CLIENTES
GRANT SELECT ON Detalhes_Prato TO CLIENTES
GRANT SELECT ON PratoDiario TO CLIENTES
GRANT SELECT ON Bloquear TO CLIENTES
GRANT SELECT, INSERT, UPDATE, DELETE ON Selecionar_R_Favoritos TO CLIENTES
GRANT SELECT, INSERT, UPDATE, DELETE ON Selecionar_P_Favoritos TO CLIENTES
```

Figura 6 - Permissões dos Clientes

Restaurantes

```
GRANT SELECT ON Utilizador TO RESTAURANTES
GRANT UPDATE ON Utilizador(Nome, Email, Username, Password) TO RESTAURANTES
GRANT SELECT, UPDATE ON Restaurante TO RESTAURANTES
GRANT SELECT, INSERT, DELETE ON Selecionar_Servico TO RESTAURANTES
GRANT SELECT ON Tipo_Servico TO RESTAURANTES
GRANT SELECT ON Pedir_Registo TO RESTAURANTES
GRANT SELECT, INSERT ON Nome_Prato TO RESTAURANTES
GRANT SELECT ON Tipo_PratoDoDia TO RESTAURANTES
GRANT SELECT, INSERT, UPDATE, DELETE ON Detalhes_Prato TO RESTAURANTES
GRANT SELECT, INSERT, UPDATE, DELETE ON PratoDiario TO RESTAURANTES
GRANT SELECT ON Bloquear TO RESTAURANTES
```

Figura 7 - Permissões dos Restaurantes

Administradores

```
GRANT SELECT, INSERT, UPDATE, DELETE ON Administrador TO ADMINISTRADORES
GRANT SELECT, UPDATE ON Pedir_Registo TO ADMINISTRADORES
GRANT SELECT ON Restaurante TO ADMINISTRADORES
GRANT SELECT ON Selecionar_Servico TO ADMINISTRADORES
GRANT SELECT ON Tipo_Servico TO ADMINISTRADORES
GRANT SELECT ON Utilizador TO ADMINISTRADORES
GRANT SELECT, INSERT, UPDATE ON Bloquear TO ADMINISTRADORES
GRANT UPDATE ON Utilizador(Estado) TO ADMINISTRADORES
GRANT SELECT ON Cliente TO ADMINISTRADORES
```

Figura 8 - Permissões dos Administradores

Resolução de problemas de concorrência no sistema de Base de Dados centralizada – Procedures:

Em relação aos Procedures, neste subcapítulo vamos apresentar e explicitar apenas alguns exemplos de Transações com Insert, Delete e Update, visto que as restantes implementações seguem a mesma lógica e constam também em Anexo.

Visitantes:

```
GRANT EXECUTE ON Criar_Utilizador TO VISITANTES
GRANT EXECUTE ON Criar_Cliente TO VISITANTES
GRANT EXECUTE ON Criar_Restaurante TO VISITANTES
```

Figura 9 - Procedures dos Visitantes

```
CREATE PROCEDURE Criar_Utilizador
    @nome           NVARCHAR(150),
    @email          NVARCHAR(200),
    @username       NVARCHAR(10),
    @password       NVARCHAR(10)
AS
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
BEGIN TRANSACTION Novo_Utilizador
    IF (NOT EXISTS (SELECT * FROM Utilizador U WHERE (U.Email=@email AND U.Username=@username)))
        INSERT INTO Utilizador(Nome, Email, Username, Password)
        VALUES (@nome, @email, @username, @password)
    ELSE
        PRINT 'Email ou username já existe!'
    IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
        GOTO ERRO
COMMIT TRANSACTION Novo_Utilizador
RETURN 1

ERRO:
    ROLLBACK TRANSACTION Novo_Utilizador
    RETURN -1
GO
```

Figura 10 - Procedure Criar_Utilizador

No procedimento Criar_Utilizador, recebemos como parâmetro o nome, email, username e password.

A Transação Novo_Utilizador começa com a verificação da existência do email e username passado como parâmetro na Tabela do Utilizador. Caso não exista, os dados passados como parâmetros são inseridos na Tabela Utilizador (INSERT) e a Transação Novo_Utilizador termina com sucesso. Caso existam, é apresentada uma mensagem como 'Email ou username já existe', terminado assim a Transação sem sucesso.

A Transação termina com sucesso retornando 1. Na existência de algum erro faz um ROLLBACK TRANSACTION retornando -1, ou seja, os dados não são inseridos na Tabela Utilizador.

```
CREATE PROCEDURE Criar_Cliente
    @id_utilizador INTEGER
AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRANSACTION
    IF (EXISTS (SELECT * FROM Utilizador U WHERE (U.ID_Utilizador=@id_utilizador)))
    BEGIN
        IF ((SELECT U.Estado FROM Utilizador U WHERE (U.ID_Utilizador=@id_utilizador)) = 'Registado')
        BEGIN
            INSERT INTO Cliente(ID_Cliente)
            VALUES (@id_utilizador)

            UPDATE Utilizador
            SET Estado = 'Ativo'
            WHERE ID_Utilizador=@id_utilizador
        END
        ELSE IF ((SELECT U.Estado FROM Utilizador U WHERE (U.ID_Utilizador=@id_utilizador)) = 'Ativo')
        BEGIN
            IF (EXISTS (SELECT * FROM Restaurante R WHERE R.ID_Restaurante=@id_utilizador))
                PRINT 'Utilizador é restaurante!'
            ELSE
                PRINT 'Cliente já registado!'
        END
    END
    ELSE
        PRINT 'Utilizador não existe!'

    IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
        GOTO ERRO
COMMIT TRANSACTION
RETURN 1

ERRO:
    ROLLBACK TRANSACTION
    RETURN -1
GO
```

Figura 11 - Procedure Criar_Cliente

No procedimento Criar_Cliente, recebemos como parâmetro o id_utilizador.

A Transação começa com a verificação da existência do id passado como parâmetro na Tabela do Utilizador. Caso exista, verifica-se se o Estado do Utilizador da Tabela Utilizador se encontra 'Registado'. Se for verificado, o id é inserido (INSERT) na Tabela Cliente, atualizando (UPDATE) o seu estado para 'Ativo' e o código prossegue. Caso o id não exista, é apresentada uma mensagem e a Transação termina.

No caso do Estado da Tabela Utilizador se encontrar já 'Ativo' verifica se esse Utilizador está registado como Restaurante, ou se já está registado como Cliente. Em ambos os casos são apresentadas mensagens como 'Utilizador é restaurante' e 'Cliente já registado'.

A Transação termina com sucesso retornando 1. Na existência de algum erro faz um ROLLBACK TRANSACTION retornando -1, ou seja, o Utilizador não é adicionado como Cliente.

Clientes

```
CREATE PROCEDURE Alterar_Utilizador
    @id             INTEGER,
    @nome           NVARCHAR(150),
    @email          NVARCHAR(200),
    @username       NVARCHAR(10),
    @password       NVARCHAR(10)
AS
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
BEGIN TRANSACTION
    IF ( NOT EXISTS (SELECT * FROM Utilizador u WHERE (u.ID_Utilizador=@id)))
        PRINT 'ID de Utilizador não existe!'
    ELSE IF ( EXISTS (SELECT * FROM Utilizador u WHERE (u.Username=@username) AND u.ID_Utilizador<>@id ) OR EXISTS (SELECT * FROM Administrador A WHERE (A.Username=@username)))
        PRINT 'Username já existente!'
    ELSE IF ( EXISTS (SELECT * FROM Utilizador u WHERE (u.Email=@email) AND u.ID_Utilizador<>@id ) OR EXISTS (SELECT * FROM Administrador A WHERE (A.Email=@email)))
        PRINT 'Email já existente!'
    ELSE
    BEGIN
        UPDATE Utilizador
        SET Nome = @nome, Email = @email, Username = @username, Password = @password
        WHERE (ID_Utilizador = @id)
    END

    IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
        GOTO ERRO
COMMIT
RETURN 1

ERRO:
    ROLLBACK
    RETURN -1
GO
```

Figura 12 - Procedure Alterar_Utilizador

No procedimento Alterar_Utilizador, recebemos como parâmetro o id, nome, email, username e password.

A Transação começa com a verificação da existência do id passado como parâmetro na Tabela do Utilizador. Caso não se verifique é apresentada uma mensagem como 'ID de Utilizador não existe'. Caso se verifique, passa para outras condições onde ocorre a verificação da existência dos parâmetros únicos (email e username) na Tabela Utilizador ou na Tabela Administrador. Se estes já existirem, é apresentada uma mensagem como 'Username já existente' e 'Email já existente' e a Transação termina sem sucesso, ou seja, os dados passados como parâmetros não vão substituir os dados já existentes na Tabela Utilizador.

Caso todas as condições não sejam verificadas, ocorre uma atualização (UPDATE) dos dados existentes na Tabela Utilizador pelos dados passados como parâmetros, terminando assim a Transação com sucesso.

A Transação termina com sucesso retornando 1. Na existência de algum erro faz um ROLLBACK TRANSACTION retornando -1, ou seja, os dados do Utilizador não são atualizados.

Restaurantes

```
CREATE PROCEDURE Apagar_Detalhes_PratoDia
    @id_detalhes            INTEGER
AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRANSACTION
    IF (EXISTS (SELECT * FROM Detalhes_Prato D WHERE ( D.ID_Detalhes = @id_detalhes)))
    BEGIN
        DELETE FROM Detalhes_Prato
        WHERE ID_Detalhes=@id_detalhes
    END
    ELSE
            PRINT 'Não existe nenhum prato com os dados indicados!'

    IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
    GOTO ERRO

COMMIT
RETURN 1

ERRO:
    ROLLBACK
    RETURN -1
GO
```

Figura 13 - Procedure Apagar_Detalhes_PratoDia

No procedimento Apagar_Detalhes_PratoDia, recebemos como parâmetro o id detalhes.

A Transação começa com a verificação da existência do id passado como parâmetro na Tabela Detalhes_Prato. Caso não se verifique, é apresentada uma mensagem como 'Não existe nenhum prato com os dados indicados' e a Transação termina sem sucesso. Caso se verifique, apaga os detalhes do prato da Tabela Detalhes_Prato com o id correspondente ao id passado por parâmetro.

A Transação termina com sucesso retornando 1. Na existência de algum erro faz um ROLLBACK TRANSACTION retornando -1, ou seja, os detalhes/dados do prato não são apagados.

5. Conclusão

Após a finalização desta primeira fase, acreditamos ter conseguido reunir o máximo de informação e elementos necessários para o seu desenvolvimento, que futuramente nos vão permitir realizar todos os próximos relatórios propostos, e todas as operações necessárias para que no final se cumpra o objetivo do trabalho.

Com o trabalho experimental 1, para além de testarmos de forma prática a matéria lecionada através da implementação de políticas de segurança e acesso a dados, também resolvemos problemas de concorrência no sistema de Base de Dados centralizada. Assim, na Base de Dados utilizada foram criados Logins, Roles, Users e Procedures.

Para a resolução de todos os objetivos pretendidos nesta primeira fase, foram feitas modificações à nossa Base de Dados reutilizada, de maneira a melhorar o seu funcionamento.

Apesar de todas as dificuldades e obstáculos, com a ajuda dos professores que lecionam esta cadeira e de todo o material disponibilizado por eles, finalizamos o trabalho experimental 1 e achamos que conseguimos cumprir todos os seus objetivos.

6. Bibliografia

- Martins, P, Marques, A. (2021). Protocolos dos Trabalhos Experimentais 1 e 2 [2019-2020]. UTAD, Vila Real.
- Martins, P. (2021). Acetatos das Aulas Teóricas. UTAD, Vila Real.
- Stored Procedures (Database Engine), Microsoft. https://docs.microsoft.com/en-us/sql/relational-databases/stored-procedures/stored-procedures-database-engine?view=sql-server-ver15

7. Anexo A – Base de Dados

```
USE MASTER
GO

CREATE DATABASE TABD_RISTORANTIS
GO
--Ana Dias al69691
--Diana Alves al68557
--Diana Ferreira al68938
--Rui Vaz al68565

USE TABD_RISTORANTIS
GO

---ENTIDADES---

CREATE TABLE Administrador(
	ID_Administrador	INTEGER IDENTITY (1,1) NOT NULL,
	Username			NVARCHAR(10) NOT NULL,
	Password			NVARCHAR(10) NOT NULL,
	Email				NVARCHAR(200) NOT NULL,
	Nome				NVARCHAR(150) NOT NULL,
	ID_Criador			INTEGER,

	PRIMARY KEY (ID_Administrador),
	FOREIGN KEY(ID_CRIADOR) REFERENCES Administrador(Id_Administrador)
)

CREATE TABLE Utilizador(
	ID_Utilizador		INTEGER IDENTITY(1,1) NOT NULL,
	Nome				NVARCHAR(150) NOT NULL,
	Email				NVARCHAR(200) NOT NULL,
	Username			NVARCHAR(10) NOT NULL,
	Password			NVARCHAR(10) NOT NULL,
	Estado				NVARCHAR(10) DEFAULT 'Registado' NOT NULL,

	PRIMARY KEY (ID_Utilizador)
)

CREATE TABLE Cliente(
	ID_Cliente			INTEGER NOT NULL,

	PRIMARY KEY(ID_Cliente),
	FOREIGN KEY(ID_Cliente) REFERENCES Utilizador(ID_Utilizador)
)

CREATE TABLE Restaurante(
	ID_Restaurante		INTEGER NOT NULL,
	Telefone			NVARCHAR(9) NOT NULL,
	Localizacao_GPS		NVARCHAR(100) NOT NULL,
	Endereco_Codigo_Postal	NVARCHAR(8) NOT NULL,
	Endereco_Morada		NVARCHAR(50) NOT NULL,
	Endereco_Localidade	NVARCHAR(50) NOT NULL,
	Horario				NVARCHAR(MAX) NOT NULL,
	Fotografia			NVARCHAR(MAX) NOT NULL,
	Dia_Descanso		NVARCHAR(50) NOT NULL,



	CHECK(Telefone LIKE '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
	CHECK(Endereco_Codigo_Postal LIKE '[0-9][0-9][0-9][0-9]-[0-9][0-9][0-9]'),
```

```
	PRIMARY KEY(ID_Restaurante),
	FOREIGN KEY (ID_Restaurante) REFERENCES Utilizador(ID_Utilizador)
)

CREATE TABLE Tipo_Servico(
	ID_Servico			INTEGER IDENTITY(1,1) NOT NULL,
	Nome_Tipo_S			NVARCHAR(50) NOT NULL,

	PRIMARY KEY(ID_Servico)
)

INSERT INTO Tipo_Servico
VALUES		('Local'),
		('Take-Away'),
		('Entrega')
GO

INSERT INTO Tipo_PratoDoDia
VALUES		('Carne'),
		('Peixe'),
		('Vegan')
GO

CREATE TABLE Tipo_PratoDoDia(
	ID_Tipo_P			INTEGER IDENTITY(1,1) NOT NULL,
	Nome_tipo_P			NVARCHAR(50) NOT NULL

	PRIMARY KEY (ID_Tipo_P)
)

CREATE TABLE Nome_Prato(
	ID_Prato			INTEGER IDENTITY(1,1) NOT NULL,
	Nome				NVARCHAR(50) NOT NULL,
	Tipo				INTEGER NOT NULL,

	PRIMARY KEY (ID_Prato),
	Foreign Key(Tipo) references Tipo_PratoDoDia(ID_Tipo_P)
)

CREATE TABLE Detalhes_Prato (
	ID_Detalhes			INTEGER IDENTITY(1,1) NOT NULL,
	Fotografia			NVARCHAR(MAX),	--opcional--
	Descricao			NVARCHAR(50) NOT NULL,
	Preco				MONEY NOT NULL,
	ID_Nome_Prato			INTEGER NOT NULL,
	ID_Restaurante			INTEGER NOT NULL

	PRIMARY KEY (ID_Detalhes),
	Foreign Key(ID_Nome_Prato) references Nome_Prato(ID_Prato),
	Foreign Key(ID_Restaurante) references Restaurante(ID_Restaurante)
)

---RELACIONAMENTOS---
CREATE TABLE PratoDiario
	ID_PratoDiario			INTEGER IDENTITY(1,1) NOT NULL,
	ID_Restaurante			INTEGER NOT NULL,
	ID_DetalhesPrato		INTEGER NOT NULL,
	Data_Disponibilidade		DATE NOT NULL,



	PRIMARY KEY (ID_PratoDiario),
```

```
	FOREIGN KEY (ID_Restaurante) REFERENCES Restaurante(ID_Restaurante),
	FOREIGN KEY (ID_DetalhesPrato) REFERENCES Detalhes_Prato(ID_Detalhes) ON
DELETE CASCADE,
)

CREATE TABLE Selecionar_Servico(
	ID_Servico			INTEGER  NOT NULL,
	ID_Restaurante			INTEGER  NOT NULL,

	PRIMARY KEY(ID_Servico,ID_Restaurante),
	FOREIGN KEY(ID_Servico) REFERENCES Tipo_Servico(ID_Servico),
	FOREIGN KEY(ID_Restaurante) REFERENCES Restaurante(ID_Restaurante)
)

CREATE TABLE Pedir_Registo (
	ID_Pedir_Registo		INTEGER IDENTITY(1,1) NOT NULL,
	Data_Pedido			DATE DEFAULT GETDATE() NOT NULL,
	Data_Resultado			DATE,
	Resultado			BIT, --ou é aceite ou nao
	Motivo_Rejeicao			NVARCHAR(100), --opcional--
	ID_Restaurante			INTEGER NOT NULL,
	ID_Administrador		INTEGER,

	PRIMARY KEY (ID_Pedir_Registo),
	FOREIGN KEY (ID_Restaurante) REFERENCES Restaurante(ID_Restaurante),
	FOREIGN KEY (ID_Administrador) REFERENCES Administrador(ID_Administrador),
)

CREATE TABLE Selecionar_R_Favoritos(
	ID_Cliente			INTEGER  NOT NULL,
	ID_Restaurante			INTEGER  NOT NULL,
	Notificacao_R			BIT  DEFAULT 0 NOT NULL,

	PRIMARY KEY(ID_Cliente, ID_Restaurante),
	FOREIGN KEY(ID_Cliente) REFERENCES Cliente(ID_Cliente),
	FOREIGN KEY (ID_Restaurante) REFERENCES Restaurante(ID_Restaurante)
)

CREATE TABLE Selecionar_P_Favoritos(
	ID_Cliente			INTEGER  NOT NULL,
	ID_Prato			INTEGER  NOT NULL,
	Notificacao_P			BIT NOT NULL,

	PRIMARY KEY (ID_Cliente, ID_Prato),
	FOREIGN KEY(ID_Cliente) REFERENCES Cliente(ID_Cliente),
	FOREIGN KEY(ID_Prato) REFERENCES Nome_Prato(ID_Prato)
)

CREATE TABLE Bloquear(
	ID_Bloqueio			INTEGER IDENTITY(1,1),
	Data_Bloquear			DATE DEFAULT GETDATE() NOT NULL,
	Motivo_Bloqueio			NVARCHAR (100) NOT NULL,
	Data_Desbloqueio		DATE, --opcional--
	ID_Administrador		INTEGER  NOT NULL,
	ID_Utilizador			INTEGER NOT NULL,

	PRIMARY KEY(ID_Bloqueio),
	FOREIGN KEY(ID_Administrador) REFERENCES Administrador(ID_Administrador),
	FOREIGN KEY(ID_Utilizador) REFERENCES Utilizador(ID_Utilizador)
)
```

8. Anexo B – Procedures e Políticas de Segurança e Acesso aos Dados

```
USE TABD_RISTORANTIS
GO

SET IMPLICIT_TRANSACTIONS OFF

--- POLÍTICAS DE SEGURANÇA E ACESSO A DADOS
--CRIAÇÃO DE LOGINS
CREATE LOGIN ADMINISTRADOR WITH PASSWORD='12345'
CREATE LOGIN CLIENTE WITH PASSWORD='12345'
CREATE LOGIN RESTAURANTE WITH PASSWORD='12345'

-- CRIAÇÃO DE USERS
USE TABD_RISTORANTIS
GO

CREATE USER ADMINISTRADOR1 FOR LOGIN ADMINISTRADOR
CREATE USER CLIENTE1 FOR LOGIN CLIENTE
CREATE USER RESTAURANTE1 FOR LOGIN RESTAURANTE
CREATE USER VISITANTE1 WITHOUT LOGIN

--CRIAÇÃO DE ROLES
CREATE ROLE ADMINISTRADORES
CREATE ROLE CLIENTES
CREATE ROLE RESTAURANTES
CREATE ROLE VISITANTES
ALTER ROLE ADMINISTRADORES ADD MEMBER ADMINISTRADOR1
ALTER ROLE CLIENTES ADD MEMBER CLIENTE1
ALTER ROLE RESTAURANTES ADD MEMBER RESTAURANTE1
ALTER ROLE VISITANTES ADD MEMBER VISITANTE1

--- ATRIBUIÇÃO DE PERMISSÕES NAS TABELAS
--- PERMISSÕES DOS VISITANTES
USE TABD_RISTORANTIS
GO

GRANT SELECT, INSERT ON Restaurante TO VISITANTES
GRANT SELECT ON Utilizador(ID_Utilizador, Nome, Email, Estado) TO VISITANTES
GRANT SELECT, INSERT ON Selecionar_Servico TO VISITANTES
GRANT SELECT ON Tipo_Servico TO VISITANTES
GRANT SELECT ON Nome_Prato TO VISITANTES
GRANT SELECT ON Tipo_PratoDoDia TO VISITANTES
GRANT SELECT ON Detalhes_Prato TO VISITANTES
GRANT SELECT ON PratoDiario TO VISITANTES

GRANT INSERT ON Utilizador TO VISITANTES
GRANT INSERT ON Cliente TO VISITANTES
GRANT INSERT ON Pedir_Registo TO VISITANTES

--- PERMISSÕES DOS CLIENTES
USE TABD_RISTORANTIS
GO

GRANT SELECT ON Cliente TO CLIENTES
GRANT SELECT ON Restaurante TO CLIENTES
GRANT SELECT ON Selecionar_Servico TO CLIENTES
GRANT SELECT ON Tipo_Servico TO CLIENTES
GRANT SELECT ON Utilizador TO CLIENTES
GRANT UPDATE ON Utilizador(ID_Utilizador, Nome, Email, Username, Password) TO
CLIENTES
GRANT SELECT ON Nome_Prato TO CLIENTES
```

```
GRANT SELECT ON Tipo_PratoDoDia TO CLIENTES
GRANT SELECT ON Detalhes_Prato TO CLIENTES
GRANT SELECT ON PratoDiario TO CLIENTES
GRANT SELECT ON Bloquear TO CLIENTES
GRANT SELECT, INSERT, UPDATE, DELETE ON Selecionar_R_Favoritos TO CLIENTES
GRANT SELECT, INSERT, UPDATE, DELETE ON Selecionar_P_Favoritos TO CLIENTES

--- PERMISSÕES DOS RESTAURANTES
USE TABD_RISTORANTIS
GO

GRANT SELECT ON Utilizador TO RESTAURANTES
GRANT UPDATE ON Utilizador(Nome, Email, Username, Password) TO RESTAURANTES
GRANT SELECT, UPDATE ON Restaurante TO RESTAURANTES
GRANT SELECT, INSERT, DELETE ON Selecionar_Servico TO RESTAURANTES
GRANT SELECT ON Tipo_Servico TO RESTAURANTES
GRANT SELECT ON Pedir_Registo TO RESTAURANTES
GRANT SELECT, INSERT ON Nome_Prato TO RESTAURANTES
GRANT SELECT ON Tipo_PratoDoDia TO RESTAURANTES
GRANT SELECT, INSERT, UPDATE, DELETE ON Detalhes_Prato TO RESTAURANTES
GRANT SELECT, INSERT, UPDATE, DELETE ON PratoDiario TO RESTAURANTES
GRANT SELECT ON Bloquear TO RESTAURANTES

--- PERMISSÕES DOS ADMINISTRADORES
USE TABD_RISTORANTIS
GO

GRANT SELECT, INSERT, UPDATE, DELETE ON Administrador TO ADMINISTRADORES
GRANT SELECT, UPDATE ON Pedir_Registo TO ADMINISTRADORES
GRANT SELECT ON Restaurante TO ADMINISTRADORES
GRANT SELECT ON Selecionar_Servico TO ADMINISTRADORES
GRANT SELECT ON Tipo_Servico TO ADMINISTRADORES
GRANT SELECT ON Utilizador TO ADMINISTRADORES
GRANT SELECT, INSERT, UPDATE ON Bloquear TO ADMINISTRADORES
GRANT UPDATE ON Utilizador(Estado) TO ADMINISTRADORES
GRANT SELECT ON Cliente TO ADMINISTRADORES

--PROCEDURES
--VISITANTES
GRANT EXECUTE ON Criar_Utilizador TO VISITANTES
GRANT EXECUTE ON Criar_Cliente TO VISITANTES
GRANT EXECUTE ON Criar_Restaurante TO VISITANTES

--CLIENTES
GRANT EXECUTE ON Alterar_Utilizador TO CLIENTES
GRANT EXECUTE ON Selecionar_R_Favorito TO CLIENTES
GRANT EXECUTE ON Eliminar_Selecionar_R_Favoritos TO CLIENTES
GRANT EXECUTE ON Selecionar_P_Favorito TO CLIENTES
GRANT EXECUTE ON Eliminar_Selecionar_P_Favoritos TO CLIENTES

--ADMINISTRADOR
GRANT EXECUTE ON Novo_Administrador TO ADMINISTRADORES
GRANT EXECUTE ON Alterar_Administrador TO ADMINISTRADORES
GRANT EXECUTE ON Bloquear_Utilizador TO ADMINISTRADORES
GRANT EXECUTE ON Desbloquear_Utilizador TO ADMINISTRADORES
GRANT EXECUTE ON Verificar_Pedido_Registo TO ADMINISTRADORES

--RESTAURANTES
GRANT EXECUTE ON Alterar_Utilizador TO RESTAURANTES
GRANT EXECUTE ON Alterar_Restaurante TO RESTAURANTES
GRANT EXECUTE ON Registar_Novo_Prato TO RESTAURANTES
GRANT EXECUTE ON Criar_Detalhes_PratoDia TO RESTAURANTES
GRANT EXECUTE ON Alterar_Detalhes_PratoDia TO RESTAURANTES
GRANT EXECUTE ON Apagar_Detalhes_PratoDia TO RESTAURANTES
```

```
GRANT EXECUTE ON Criar_PratoDiario TO RESTAURANTES
GRANT EXECUTE ON Alterar_PratoDiario TO RESTAURANTES
GRANT EXECUTE ON Apagar_PratoDiario TO RESTAURANTES


--PROCEDURES DOS VISITANTES
USE TABD_RISTORANTIS
GO

CREATE PROCEDURE Criar_Utilizador
	@nome		NVARCHAR(150),
	@email		NVARCHAR(200),
	@username	NVARCHAR(10),
	@password	NVARCHAR(10)
AS
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
BEGIN TRANSACTION Novo_Utilizador
	IF ((EXISTS (SELECT * FROM Utilizador U WHERE (U.Username=@username))) OR
(EXISTS (SELECT * FROM Administrador A WHERE (A.Username=@username))))
		PRINT 'Username já existente!'
	ELSE IF ((EXISTS (SELECT * FROM Utilizador U WHERE (U.Email=@email))) OR
(EXISTS (SELECT * FROM Administrador A WHERE (A.Email=@email))))
		PRINT 'Email já existente!'
	ELSE
		INSERT INTO Utilizador(Nome, Email, Username, Password)
		VALUES (@nome, @email, @username, @password)
	IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
		GOTO ERRO
COMMIT TRANSACTION Novo_Utilizador
RETURN 1

ERRO:
	ROLLBACK TRANSACTION Novo_Utilizador
	RETURN -1
GO

CREATE PROCEDURE Criar_Cliente
	@id_utilizador INTEGER
AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRANSACTION
	IF (EXISTS (SELECT * FROM Utilizador U WHERE
(U.ID_Utilizador=@id_utilizador)))
	BEGIN
		IF ((SELECT U.Estado FROM Utilizador U WHERE
(U.ID_Utilizador=@id_utilizador)) = 'Registado')
		BEGIN
			INSERT INTO Cliente(ID_Cliente)
			VALUES (@id_utilizador)

			UPDATE Utilizador
			SET Estado = 'Ativo'
			WHERE ID_Utilizador=@id_utilizador
		END
		ELSE
		BEGIN
			IF (EXISTS (SELECT * FROM Restaurante R WHERE
R.ID_Restaurante=@id_utilizador))
				PRINT 'Utilizador é restaurante!'
			ELSE IF ((SELECT U.Estado FROM Utilizador U WHERE
(U.ID_Utilizador=@id_utilizador)) = 'Ativo')
				PRINT 'Cliente já registado!'
		END
```

```
	END
	ELSE
		PRINT 'Utilizador não existe!'

	IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
		GOTO ERRO
COMMIT TRANSACTION
RETURN 1

ERRO:
	ROLLBACK TRANSACTION
	RETURN -1
GO

CREATE TYPE ServicoType AS TABLE(ID INTEGER)
GO

--DECLARE @lista ServicoType;
--INSERT @lista VALUES (1),(2)
--EXECUTE Criar_Restaurante '3','123456789','','4960-236','','','','','',@lista
--GO

CREATE PROCEDURE Criar_Restaurante
	@id_utilizador			INTEGER,
	@telefone			NVARCHAR(9),
	@localizacao_GPS		NVARCHAR(100),
	@Codigo_Postal			NVARCHAR(8),
	@Morada				NVARCHAR(50),
	@Localidade			NVARCHAR(50),
	@horario			NVARCHAR(MAX),
	@fotografia			NVARCHAR(MAX),
	@dia_descanso			NVARCHAR(50),
	@id_servico			ServicoType READONLY
AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRANSACTION Novo_Restaurante
	IF (EXISTS (SELECT * FROM Utilizador U WHERE
(U.ID_Utilizador=@id_utilizador)))
	BEGIN
		IF ((SELECT U.Estado FROM Utilizador U WHERE
(U.ID_Utilizador=@id_utilizador)) = 'Registado')
		BEGIN
			INSERT INTO Restaurante(ID_Restaurante, Telefone,
Localizacao_GPS,
			Endereco_Codigo_Postal, Endereco_Morada, Endereco_Localidade,
Horario, Fotografia, Dia_Descanso)
			VALUES (@id_utilizador, @telefone, @localizacao_GPS,
@Codigo_Postal, @Morada, @Localidade,
			@horario, @fotografia, @dia_descanso)

			IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
			GOTO ERRO

			--Variável que guarda o número de serviços que um restaurante
tem
			DECLARE @nr_servicos INTEGER
			DECLARE @servico INTEGER
			SET @nr_servicos = (SELECT COUNT(*) FROM @id_servico)
			WHILE @nr_servicos > 0
			BEGIN
				SET @servico = (SELECT ID FROM (SELECT ROW_NUMBER()
OVER (ORDER BY ID ASC) AS RowNum, * FROM @id_servico) T2 WHERE RowNum =
@nr_servicos)
```

```
                        IF (EXISTS (SELECT * FROM Tipo_Servico T WHERE
(T.ID_Servico = @servico)))
                        BEGIN
                                INSERT INTO Selecionar_Servico(ID_Restaurante,
ID_Servico)
                                VALUES (@id_utilizador, @servico)
                        END
                        ELSE
                        BEGIN
                                PRINT 'Tipo de serviço não existe!'
                        END

                                SET @nr_servicos = @nr_servicos - 1
                END

                IF (@@ERROR <> 0)
                GOTO ERRO

                INSERT INTO Pedir_Registo(ID_Restaurante)
                VALUES (@id_utilizador)

                IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
                GOTO ERRO

                UPDATE Utilizador
                SET Estado = 'Espera'
                WHERE ID_Utilizador=@id_utilizador

                IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
                GOTO ERRO
        END
        ELSE IF ((SELECT U.Estado FROM Utilizador U WHERE
(U.ID_Utilizador=@id_utilizador)) = 'Ativo')
        BEGIN
                IF (EXISTS (SELECT * FROM Cliente C WHERE
C.ID_Cliente=@id_utilizador))
                        PRINT 'Utilizador é cliente!'
                ELSE
                        PRINT 'Restaurante já registado!'
        END
        ELSE IF ((SELECT U.Estado FROM Utilizador U WHERE
(U.ID_Utilizador=@id_utilizador)) = 'Espera')
                PRINT 'Restaurante a aguardar aprovação!'

    END
    ELSE
        PRINT 'Utilizador não existe!'

    COMMIT TRANSACTION Novo_Restaurante
    RETURN 1

ERRO:
    ROLLBACK TRANSACTION Novo_Restaurante
    RETURN -1

GO


--PROCEDURES DOS CLIENTES
USE TABD_RISTORANTIS
GO

CREATE PROCEDURE Alterar_Utilizador
    @id                         INTEGER,
```

```
	@nome		NVARCHAR(150),
	@email		NVARCHAR(200),
	@username	NVARCHAR(10),
	@password	NVARCHAR(10)
AS
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
BEGIN TRANSACTION
	IF ( NOT EXISTS (SELECT * FROM Utilizador u WHERE (u.ID_Utilizador=@id)))
		PRINT 'ID de Utilizador não existe!'
	ELSE IF ( EXISTS (SELECT * FROM Utilizador u WHERE (u.Username=@username)
AND u.ID_Utilizador<>@id ) OR EXISTS (SELECT * FROM Administrador A WHERE
(A.Username=@username)))
		PRINT 'Username já existente!'
	ELSE IF ( EXISTS (SELECT * FROM Utilizador u WHERE (u.Email=@email) AND
u.ID_Utilizador<>@id ) OR EXISTS (SELECT * FROM Administrador A WHERE
(A.Email=@email)))
		PRINT 'Email já existente!'
	ELSE
	BEGIN
		UPDATE Utilizador
		SET Nome = @nome, Email = @email, Username = @username, Password =
@password
		WHERE (ID_Utilizador = @id)
	END

	IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
		GOTO ERRO
COMMIT
RETURN 1

ERRO:
	ROLLBACK
	RETURN -1
GO

--dá para adicionar na tabela caso ainda nao exista, ou alterar caso já exista
CREATE PROCEDURE Selecionar_R_Favorito
	@id_Cliente			INTEGER,
	@id_Restaurante			INTEGER,
	@notificacao		BIT
AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRANSACTION
	IF (EXISTS (SELECT * FROM Selecionar_R_Favoritos R WHERE
(R.ID_Cliente=@id_Cliente AND R.ID_Restaurante=@id_Restaurante)))
	BEGIN
		UPDATE Selecionar_R_Favoritos
		SET Notificacao_R=@notificacao
		WHERE (ID_Cliente=@id_Cliente AND ID_Restaurante=@id_Restaurante)
	END
	ELSE
	BEGIN
		IF (EXISTS (SELECT * FROM Cliente C WHERE ( C.ID_Cliente =
@id_Cliente)) AND EXISTS (SELECT * FROM Restaurante R WHERE ( R.ID_Restaurante =
@id_Restaurante)))
			INSERT INTO Selecionar_R_Favoritos(ID_Cliente,ID_Restaurante,
Notificacao_R)
			VALUES (@id_Cliente, @id_Restaurante, @notificacao)
		ELSE
			PRINT 'Não existe nenhum Cliente ou Restaurante com os dados
indicados!'
	END

	IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
```

```
	GOTO ERRO
COMMIT
RETURN 1

ERRO:
	ROLLBACK
	RETURN -1
GO

CREATE PROCEDURE Eliminar_Selecionar_R_Favoritos
	@id_Cliente			INTEGER,
	@id_Restaurante			INTEGER
AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRANSACTION
	IF (EXISTS (SELECT * FROM Selecionar_R_Favoritos R WHERE ( R.ID_Cliente =
@id_Cliente AND R.ID_Restaurante = @id_Restaurante)))
		DELETE FROM Selecionar_R_Favoritos
		WHERE (ID_Cliente=@id_Cliente AND ID_Restaurante=@id_Restaurante)
	ELSE
		PRINT 'Não existe nenhum Favorito com os dados indicados!'

	IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
	GOTO ERRO
COMMIT
RETURN 1

ERRO:
	ROLLBACK
	RETURN -1
GO

--dá para adicionar na tabela caso ainda nao exista, ou alterar caso já exista
CREATE PROCEDURE Selecionar_P_Favorito
	@id_Cliente			INTEGER,
	@id_Prato			INTEGER,
	@notificacao		BIT
AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRANSACTION
	IF (EXISTS (SELECT * FROM Selecionar_P_Favoritos P WHERE
(P.ID_Cliente=@id_Cliente AND P.ID_Prato=@id_Prato)))
	BEGIN
		UPDATE Selecionar_P_Favoritos
		SET Notificacao_P=@notificacao
		WHERE (ID_Cliente=@id_Cliente AND ID_Prato=@id_Prato)
	END
	ELSE
	BEGIN
		IF (EXISTS (SELECT * FROM Cliente C WHERE ( C.ID_Cliente =
@id_Cliente)) AND EXISTS (SELECT * FROM Nome_Prato P WHERE ( P.ID_Prato =
@id_Prato)))
			INSERT INTO Selecionar_P_Favoritos(ID_Cliente,ID_Prato,
Notificacao_P)
			VALUES (@id_Cliente, @id_Prato, @notificacao)
		ELSE
			PRINT 'Não existe nenhum Cliente ou Prato com os dados
indicados!'
	END

	IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
	GOTO ERRO
COMMIT
RETURN 1
```

```
ERRO:
	ROLLBACK
	RETURN -1
GO

CREATE PROCEDURE Eliminar_Selecionar_P_Favoritos
	@id_Cliente 			INTEGER,
	@id_Prato 		INTEGER
AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRANSACTION
	IF (EXISTS (SELECT * FROM Selecionar_P_Favoritos P WHERE ( P.ID_Cliente =
@id_Cliente AND P.ID_Prato = @id_Prato)))
		DELETE FROM Selecionar_P_Favoritos
		WHERE (ID_Cliente=@id_Cliente AND ID_Prato=@id_Prato)
	ELSE
		PRINT 'Não existe nenhum Favorito com os dados indicados!'

	IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
	GOTO ERRO
COMMIT
RETURN 1

ERRO:
	ROLLBACK
	RETURN -1
GO

--PROCEDURES DOS ADMINISTRADOR
USE TABD_RISTORANTIS
GO

CREATE PROCEDURE Novo_Administrador
	@username 		NVARCHAR(10),
	@password 		NVARCHAR(10),
	@email 			NVARCHAR(200),
	@nome 			NVARCHAR(150),
	@id_criador 		INTEGER
AS
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
BEGIN TRANSACTION
	IF ((EXISTS (SELECT * FROM Utilizador U WHERE (U.Username=@username))) OR
(EXISTS (SELECT * FROM Administrador A WHERE (A.Username=@username))))
		PRINT 'Username já existente!'
	ELSE IF ((EXISTS (SELECT * FROM Utilizador U WHERE (U.Email=@email))) OR
(EXISTS (SELECT * FROM Administrador A WHERE (A.Email=@email))))
		PRINT 'Email já existente!'
	ELSE
	BEGIN
		INSERT INTO Administrador(Username, Password, Email, Nome,
ID_Criador)
		VALUES(@username, @password, @email, @nome, @id_criador)
	END
	IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
	GOTO ERRO

	IF ((NOT EXISTS (SELECT * FROM Administrador A WHERE
(A.ID_Administrador=@id_criador))) AND @id_criador <> NULL)
		BEGIN
			PRINT 'Administrador criador não existe!'
			GOTO ERRO
		END
COMMIT
```

```
RETURN 1

ERRO:
	ROLLBACK
	RETURN -1
GO

CREATE PROCEDURE Alterar_Administrador
	@id				INTEGER,
	@username		NVARCHAR(10),
	@password		NVARCHAR(10),
	@email			NVARCHAR(200),
	@nome			NVARCHAR(150)
AS
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
BEGIN TRANSACTION
	IF ( NOT EXISTS (SELECT * FROM Administrador A WHERE
(A.ID_Administrador=@id)))
		PRINT 'ID de Administrador não existe!'
	ELSE IF ( EXISTS (SELECT * FROM Utilizador u WHERE (u.Username=@username))
OR EXISTS (SELECT * FROM Administrador A WHERE (A.Username=@username) AND
A.ID_Administrador<>@id ))
		PRINT 'Username já existente!'
	ELSE IF ( EXISTS (SELECT * FROM Utilizador u WHERE (u.Email=@email)) OR
EXISTS (SELECT * FROM Administrador A WHERE (A.Email=@email) AND
A.ID_Administrador<>@id ))
		PRINT 'Email já existente!'
	ELSE
	BEGIN
		UPDATE Administrador
		SET Nome = @nome, Email = @email, Username = @username, Password =
@password
		WHERE (ID_Administrador = @id)
	END
	IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
		GOTO ERRO
COMMIT
RETURN 1

ERRO:
	ROLLBACK
	RETURN -1
GO

CREATE PROCEDURE Bloquear_Utilizador
	@id_Utilizador			INTEGER,
	@id_Administrador	INTEGER,
	@motivo_Bloqueio	NVARCHAR(100)

AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRANSACTION
	IF (NOT EXISTS (SELECT * FROM Utilizador U WHERE
(U.ID_Utilizador=@id_Utilizador)))
		PRINT 'Username não existe!'
	ELSE IF (NOT EXISTS (SELECT * FROM Administrador A WHERE
(A.ID_Administrador=@id_Administrador)))
		PRINT 'Administrador não existe!'
	ELSE
		INSERT INTO Bloquear(Motivo_Bloqueio, ID_Administrador,
ID_Utilizador)
		VALUES(@motivo_Bloqueio, @id_Administrador, @id_Utilizador)

	IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
```

```
	GOTO ERRO

	UPDATE Utilizador
	SET Estado = 'Bloqueado'
	WHERE (ID_Utilizador=@id_Utilizador)

	IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
	GOTO ERRO
COMMIT
RETURN 1

ERRO:
	ROLLBACK
	RETURN -1
GO

CREATE PROCEDURE Desbloquear_Utilizador
	@id_Bloqueio		INTEGER
AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRANSACTION
	IF (EXISTS (SELECT * FROM Bloquear B INNER JOIN Utilizador U ON
B.ID_Utilizador=U.ID_Utilizador WHERE (B.ID_Bloqueio=@id_Bloqueio)))
	BEGIN
		IF ((SELECT U.Estado FROM Bloquear B INNER JOIN Utilizador U ON
B.ID_Utilizador=U.ID_Utilizador WHERE (B.ID_Bloqueio=@id_Bloqueio))='Bloqueado')
		BEGIN
			UPDATE Bloquear
			SET Data_Desbloqueio= GETDATE()
			WHERE (ID_Bloqueio=@id_Bloqueio)

			IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
			GOTO ERRO

			UPDATE Utilizador
			SET Estado = 'Ativo'
			WHERE (ID_Utilizador=(SELECT U.ID_Utilizador FROM Utilizador
U INNER JOIN Bloquear B ON U.ID_Utilizador=B.ID_Utilizador WHERE
(B.ID_Bloqueio=@id_Bloqueio)))

			IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
			GOTO ERRO
		END
		ELSE
			PRINT 'Utilizador não está bloqueado!'
	END
	ELSE
		PRINT 'ID de Bloqueio incorreto!'

COMMIT
RETURN 1

ERRO:
	ROLLBACK
	RETURN -1
GO

CREATE PROCEDURE Verificar_Pedido_Registo
	@id_Pedir_Registo		INTEGER,
	@resultado			BIT,
	@motivo_Rejeicao		NVARCHAR(100),
	@id_Administrador		INTEGER
AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
```

```
BEGIN TRANSACTION
	IF (EXISTS (SELECT * FROM Pedir_Registo P WHERE
(P.ID_Pedir_Registo=@id_Pedir_Registo)))
	BEGIN
		IF (EXISTS (SELECT * FROM Administrador A WHERE
(A.ID_Administrador=@id_Administrador)))
		BEGIN
			UPDATE Pedir_Registo
			SET Data_Resultado= GETDATE(), Resultado=@resultado,
Motivo_Rejeicao=@motivo_Rejeicao, ID_Administrador=@id_Administrador
			WHERE (ID_Pedir_Registo=@id_Pedir_Registo)

			IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
			GOTO ERRO

			UPDATE Utilizador
			SET Estado = 'Ativo'
			WHERE ID_Utilizador = (SELECT P.ID_Restaurante FROM
Pedir_Registo P WHERE P.ID_Pedir_Registo=@id_Pedir_Registo)

			IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
			GOTO ERRO
		END
		ELSE
			PRINT 'Administrador não existe!'
	END
	ELSE
		PRINT 'Pedido de Registo não existe!'

COMMIT
RETURN 1

ERRO:
	ROLLBACK
	RETURN -1
GO


--PROCEDURES DOS RESTAURANTES

CREATE PROCEDURE Alterar_Restaurante
	@id_utilizador			INTEGER,
	@telefone			NVARCHAR(9),
	@localizacao_GPS		NVARCHAR(100),
	@Codigo_Postal			NVARCHAR(8),
	@Morada				NVARCHAR(50),
	@Localidade			NVARCHAR(50),
	@horario			NVARCHAR(MAX),
	@fotografia			NVARCHAR(MAX),
	@dia_descanso			NVARCHAR(50),
	@id_servico			ServicoType READONLY
AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRANSACTION
	IF (EXISTS (SELECT * FROM Restaurante R WHERE ( R.ID_Restaurante =
@id_utilizador)))
	BEGIN
		UPDATE Restaurante
		SET Telefone = @telefone, Localizacao_GPS = @localizacao_GPS,
Endereco_Codigo_Postal = @Codigo_Postal, Endereco_Morada=@Morada,
Endereco_Localidade=@Localidade,
		Horario=@horario, Fotografia=@fotografia,
Dia_Descanso=@dia_descanso
```

```
		WHERE (ID_Restaurante = @id_utilizador)

		IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
		GOTO ERRO

		--Variável que guarda o número de serviços que um restaurante tem
		DECLARE @nr_servicos INTEGER
		DECLARE @servico INTEGER
		SET @nr_servicos = (SELECT COUNT(*) FROM @id_servico)
		DELETE FROM Selecionar_Servico
		WHERE ID_Restaurante=@id_utilizador
		WHILE @nr_servicos > 0
		BEGIN
			SET @servico = (SELECT ID FROM (SELECT ROW_NUMBER() OVER
(ORDER BY ID ASC) AS RowNum, * FROM @id_servico) T2 WHERE RowNum = @nr_servicos)
			IF (EXISTS (SELECT * FROM Tipo_Servico T WHERE (T.ID_Servico
= @servico)))
			BEGIN
				INSERT INTO Selecionar_Servico(ID_Restaurante,
ID_Servico)
				VALUES (@id_utilizador, @servico)

				IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
					GOTO ERRO
			END
			ELSE
			BEGIN
				PRINT 'Tipo de serviço não existe!'
			END
		SET @nr_servicos = @nr_servicos - 1
		END

		IF (@@ERROR <> 0)
			GOTO ERRO

	END
	ELSE
		PRINT 'Restaurante não existe!'
COMMIT
RETURN 1

ERRO:
	ROLLBACK
	RETURN -1
GO

CREATE PROCEDURE Registar_Novo_Prato
	@nome		NVARCHAR(50),
	@tipo		INTEGER
AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRANSACTION
	IF (EXISTS (SELECT * FROM Tipo_PratoDoDia T WHERE ( T.ID_Tipo_P = @tipo)))
	BEGIN
		IF (NOT EXISTS (SELECT * FROM Nome_Prato N WHERE ( N.Nome =
@nome)))
		BEGIN
			INSERT INTO Nome_Prato(Nome, Tipo)
			VALUES (@nome, @tipo)
		END
		ELSE
			PRINT 'Prato já existe!'
	END
	ELSE
```

```
		PRINT 'Tipo de prato não existe!'

	IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
	GOTO ERRO

COMMIT
RETURN 1

ERRO:
	ROLLBACK
	RETURN -1
GO

CREATE PROCEDURE Criar_Detalhes_PratoDia
	@fotografia			NVARCHAR(MAX),
	@descricao			NVARCHAR(50),
	@preco				MONEY,
	@id_Nome_Prato			INTEGER,
	@id_restaurante			INTEGER
AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRANSACTION
	IF ( (EXISTS (SELECT * FROM Nome_Prato N WHERE ( N.ID_Prato =
@id_Nome_Prato)))
		AND (EXISTS (SELECT * FROM Restaurante R WHERE (R.ID_Restaurante =
@id_restaurante))))
	BEGIN
		INSERT INTO Detalhes_Prato(Fotografia, Descricao, Preco,
ID_Nome_Prato, ID_Restaurante)
		VALUES (@fotografia, @descricao, @preco, @id_Nome_Prato,
@id_restaurante)
	END
	ELSE
		PRINT 'Não existe nenhum restaurante ou nome de prato com os dados
indicados!'

	IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
	GOTO ERRO

COMMIT
RETURN 1

ERRO:
	ROLLBACK
	RETURN -1
GO

CREATE PROCEDURE Alterar_Detalhes_PratoDia
	@id_detalhes		INTEGER,
	@fotografia			NVARCHAR(MAX),
	@descricao			NVARCHAR(50),
	@preco				MONEY,
	@id_Nome_Prato			INTEGER
AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRANSACTION
	IF (EXISTS (SELECT * FROM Detalhes_Prato D WHERE (D.ID_Detalhes =
@id_detalhes)))
	BEGIN
		IF (EXISTS (SELECT * FROM Nome_Prato N WHERE ( N.ID_Prato =
@id_Nome_Prato)))
		BEGIN
			UPDATE Detalhes_Prato
```

```
                        SET Fotografia=@fotografia, Descricao=@descricao,
Preco=@preco, ID_Nome_Prato=@id_Nome_Prato
                        WHERE ID_Detalhes=@id_detalhes
                END
                ELSE
                        PRINT 'Não existe nenhum nome de prato com o ID indicado!'
        END
        ELSE
                PRINT 'Não existe nenhum prato com o ID indicado!'

        IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
        GOTO ERRO

COMMIT
RETURN 1

ERRO:
        ROLLBACK
        RETURN -1
GO

CREATE PROCEDURE Apagar_Detalhes_PratoDia
        @id_detalhes                INTEGER
AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRANSACTION
        IF (EXISTS (SELECT * FROM Detalhes_Prato D WHERE ( D.ID_Detalhes =
@id_detalhes)))
        BEGIN
                DELETE FROM Detalhes_Prato
                WHERE ID_Detalhes=@id_detalhes
        END
        ELSE
                        PRINT 'Não existe nenhum prato com os dados indicados!'

        IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
        GOTO ERRO

COMMIT
RETURN 1

ERRO:
        ROLLBACK
        RETURN -1
GO

CREATE PROCEDURE Criar_PratoDiario
        @id_restaurante                     INTEGER,
        @id_detalhes                INTEGER,
        @data_Disponibilidade       DATE
AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRANSACTION
        IF ( (EXISTS (SELECT * FROM Detalhes_Prato D WHERE ( D.ID_Detalhes =
@id_detalhes)))
                AND (EXISTS (SELECT * FROM Restaurante R WHERE (R.ID_Restaurante =
@id_restaurante))))
        BEGIN
                IF (NOT EXISTS (SELECT * FROM PratoDiario P WHERE (
P.ID_Restaurante = @id_restaurante AND P.ID_DetalhesPrato = @id_detalhes AND
P.Data_Disponibilidade = @data_Disponibilidade)))
                BEGIN
                        INSERT INTO PratoDiario(ID_Restaurante, ID_DetalhesPrato,
Data_Disponibilidade)
```

```
                VALUES (@id_restaurante, @id_detalhes, @data_Disponibilidade)
        END
        ELSE
                PRINT 'Prato já existe no dia indicado!'
    END
    ELSE
                PRINT 'Não existe nenhum restaurante ou prato com os dados
indicados!'

    IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
    GOTO ERRO

COMMIT
RETURN 1

ERRO:
    ROLLBACK
    RETURN -1
GO

CREATE PROCEDURE Alterar_PratoDiario
    @id_pratodiario                 INTEGER,
    @data_Disponibilidade       DATE
AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRANSACTION
    IF (EXISTS (SELECT * FROM PratoDiario P WHERE ( P.ID_PratoDiario =
@id_pratodiario)))
    BEGIN
        UPDATE PratoDiario
            SET Data_Disponibilidade=@data_Disponibilidade
            WHERE ID_PratoDiario=@id_pratodiario
    END
    ELSE
        PRINT 'Não existe nenhum prato diário com o ID indicado!'

    IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
    GOTO ERRO

COMMIT
RETURN 1

ERRO:
    ROLLBACK
    RETURN -1
GO

CREATE PROCEDURE Apagar_PratoDiario
    @id_pratodiario                 INTEGER
AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRANSACTION
    IF (EXISTS (SELECT * FROM PratoDiario P WHERE ( P.ID_PratoDiario =
@id_pratodiario)))
    BEGIN
        DELETE FROM PratoDiario
        WHERE ID_PratoDiario=@id_pratodiario
    END
    ELSE
        PRINT 'Não existe nenhum prato diário com o ID indicado!'

    IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
    GOTO ERRO
```

```
COMMIT
RETURN 1

ERRO:
	ROLLBACK
	RETURN -1

GO
```