Universidade de Trás-os-Montes e Alto Douro

Relatório Detalhado da Execução do Trabalho Prático Nº1

Licenciatura em Engenharia Informática
Técnicas Avançadas de Bases de Dados
Paulo Martins
António Marques

Realizado pelos alunos:

- Hugo Anes 68571
- Vítor Neto 68717
- João Leal 68719
- Leandro Coelho 68541

25 de Abril de 2021

ÍNDICE

INTRODUÇÃO

No âmbito da unidade curricular Técnicas Avançadas de Bases de Dados foi proposto a realização de um projeto em que iremos trabalhar com uma base de dados feita numa outra unidade curricular. Os principais objetivos a realizar nesta primeira fase foi definir politicas de segurança e acesso a dados centralizados, ou seja, definir logins e permissões que cada utilizador da base de dados poderá realizar, e a resolução de problemas de concorrência no sistema de base de dados, ou seja, definir mecanismos responsáveis pelas coordenações dos acessos concorrentes dos diferentes usuários ou processos à nossa base de dados, mecanismos estes muito importantes para termos uma base de dados consistente, segura e coerente.

Segue-se um diagrama da base de dados utilizada:

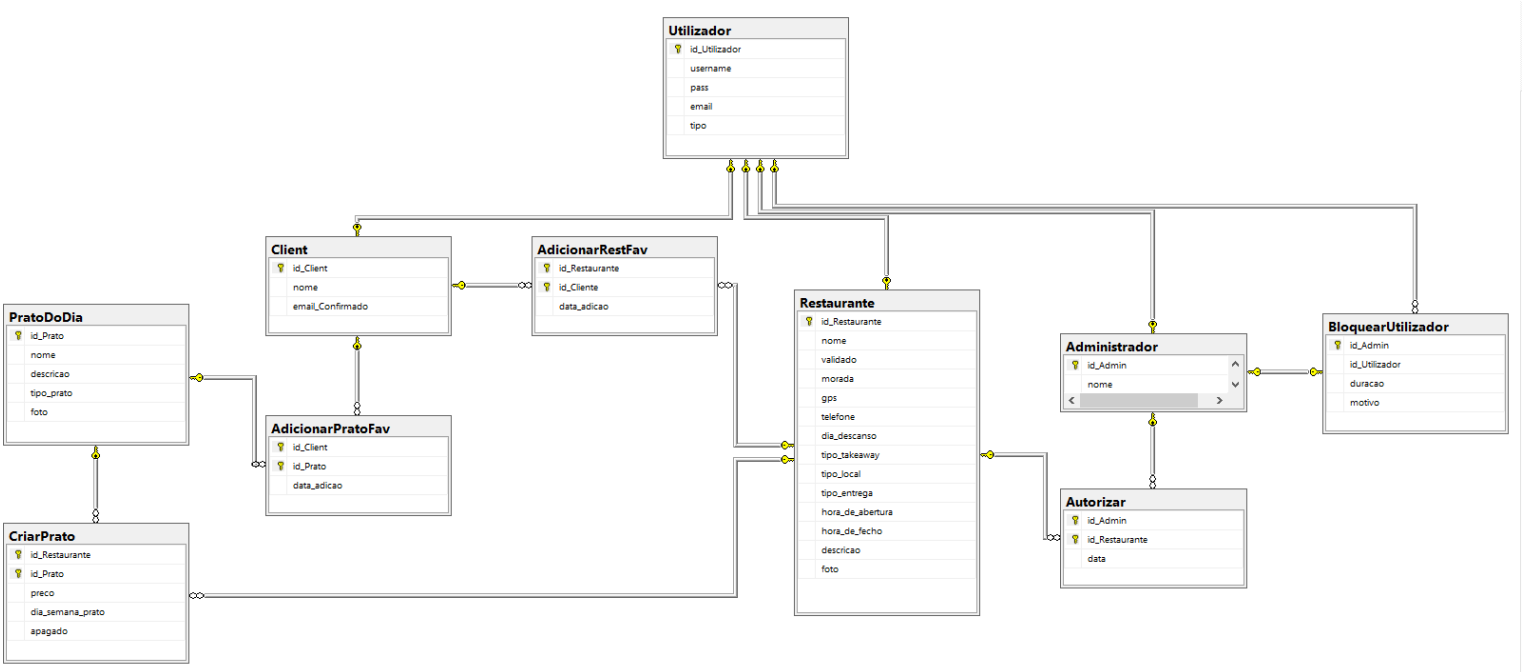

Figura 1 – Diagrama da base de dados

Políticas de Segurança

As políticas de segurança e acesso a dados centralizados serão organizadas num diagrama para melhor compreensão, demonstrado na figura seguinte, para os 3 tipos de contas, Administrador, Cliente e Restaurante:

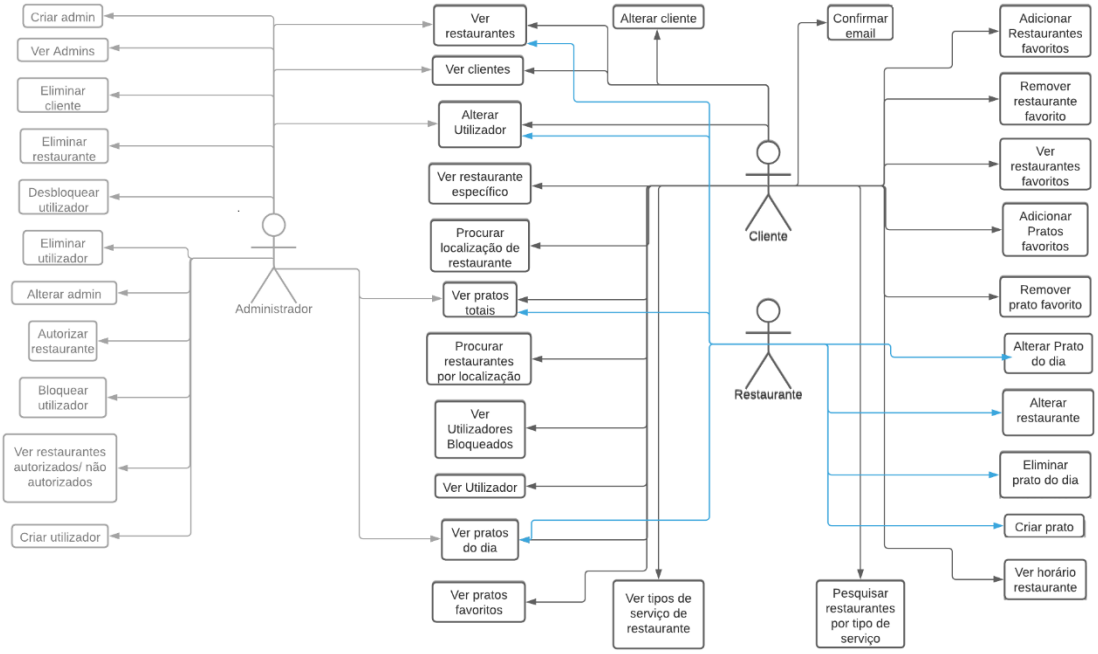

Figura 2.1 –Diagrama de permissões dos restaurantes, clientes e administradores da base de dados

Como se pode verificar no diagrama, as permissões que o administrador tem, baseiam-se mais na administração das informações gerais das outras contas bem como fazer cumprir as regras impostas pelo website, punindo quem não as cumpra. É também do encargo dos administradores criar outros administradores bem como autorizar a criação de outras contas de restaurantes.

Quanto aos restaurantes, estes possuem permissões baseadas na gestão da ementa e das informações do restaurante representado por essa mesma conta. Podem também ver os outros utilizadores do website assim como ver as ementas dos outros restaurantes no site.

Os clientes possuem permissões que lhes permite verificar as informações sobre os restaurantes que pretendem visitar. Permite que os mesmos pesquisem por restaurantes baseados nos parâmetros por eles metidos, como a localização ou o tipo de serviço provido. Os clientes podem também verificar os pratos do dia dos restaurantes registados no website bem como adicionar pratos e restaurantes a uma lista de favoritos pessoal a cada utilizador, esta funcionalidade não está só limitada à adição de novos restaurantes e pratos, permite também a remoção de favoritos não desejados. Os clientes podem também pesquisar outros clientes que estejam registados bem como alterar as suas prórpias informações.

Para além destes três roles existem ainda outra duas roles especiais. Os visitantes cujas permissões estão identificadas na seguinte figura:

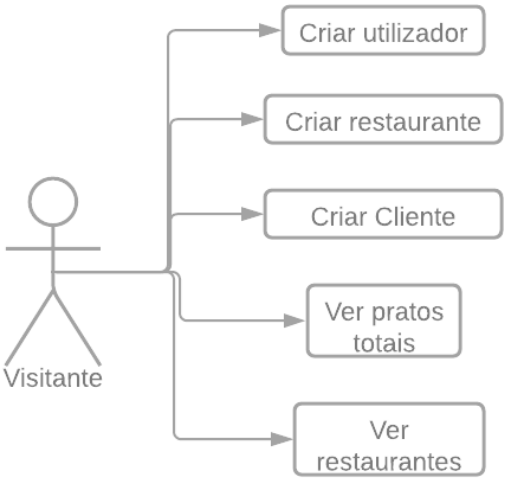

Figura 2.2 – Diagrama de permissões de visitantes da base de dados

Os visitantes apenas podem ver os pratos e restaurantes registados no site. Para poderem desbloquear as restantes funcionalides, devem-se registar no site como cliente ou restaurante, caso queiram representar um restaurante.

A outra role especial é o dono da base de dados, que possui controlo total sobre tudo, ou seja, pode efetuar todas as operações CRUD sobre qualquer um dos elementos presentes na base de dados.

Ordem de Implementação das Políticas de Segurança

Primeiramente foram criados todos os logins, um exemplo de utilizador para cada login e ainda a criação do roles para agrupar esses utilizadores, necessários para a base de dados.

```
--Logins--
CREATE LOGIN Administrador WITH PASSWORD = '1'
CREATE LOGIN Client WITH PASSWORD = '1'
CREATE LOGIN Restaurante WITH PASSWORD = '1'
CREATE LOGIN Dono WITH PASSWORD = '1'

--Users--
CREATE USER Restaurante1 FOR LOGIN Restaurante
CREATE USER Administrador1 FOR LOGIN Administrador
CREATE USER Client1 FOR LOGIN Client
CREATE USER Dono FOR LOGIN Dono
CREATE USER Visitante WITHOUT LOGIN

--Roles--
CREATE ROLE Administradores
CREATE ROLE Clients
CREATE ROLE Restaurantes

--Adicionar membros a cada uma das roles--
ALTER ROLE Clients ADD MEMBER Client1
ALTER ROLE Administradores ADD MEMBER Administrador1
ALTER ROLE Restaurantes ADD MEMBER Restaurante1
```

Figura 3.1 – Criação dos logins e atribuição das roles

Seguidamente foram atribuídas as respetivas permissões para cada tipo de utilizador consoante a necessidade do mesmo.

```
--Permissões para dono BD
GRANT SELECT, INSERT, UPDATE, DELETE TO Dono
GRANT EXECUTE ON Eliminar_Admin TO Dono

--Permissões para os Restaurantes
GRANT SELECT ON Client to Restaurantes
GRANT SELECT, INSERT, UPDATE, DELETE ON CriarPrato TO Restaurantes
GRANT SELECT, INSERT, UPDATE, DELETE ON PratoDoDia TO Restaurantes
GRANT SELECT, UPDATE ON Restaurante TO Restaurantes
GRANT SELECT, UPDATE ON Utilizador TO Restaurantes

--Permissões para os Clients
GRANT SELECT ON Restaurante TO Clients
GRANT SELECT ON CriarPrato TO Clients
GRANT SELECT ON PratoDoDia TO Clients
GRANT SELECT ON BloquearUtilizador TO Clients
GRANT SELECT,UPDATE ON Utilizador TO Clients
GRANT SELECT,INSERT,UPDATE,DELETE ON AdicionarPratoFav TO Clients
GRANT SELECT,INSERT,UPDATE,DELETE ON AdicionarRestFav TO Clients
GRANT SELECT,UPDATE ON Client TO Clients

--Permissões para os Administradores
GRANT SELECT, INSERT, UPDATE, DELETE ON BloquearUtilizador TO Administradores
GRANT SELECT, INSERT, UPDATE ON Autorizar TO Administradores
GRANT SELECT, DELETE ON Utilizador to Administradores
GRANT SELECT, DELETE ON Client TO Administradores
GRANT SELECT, INSERT, UPDATE ON Administrador TO Administradores
GRANT SELECT, DELETE ON Restaurante TO Administradores
GRANT SELECT on CriarPrato TO Administradores
GRANT SELECT on PratoDoDia TO Administradores

--Permissões e procedures para Visitante
GRANT SELECT ON Restaurante TO Visitante
GRANT SELECT ON CriarPrato TO Visitante
GRANT SELECT ON PratoDoDia TO Visitante
GRANT EXECUTE ON Criar_Restaurante TO Visitante
GRANT EXECUTE On Novo_Cliente TO Visitante
GRANT EXECUTE On Novo_Utilizador TO Visitante
```

Figura 3.2 – Atribuição das permissões a cada uma das roles

Foram também atribuídas a cada tipo de utilizador da base de dados as respetivas permissões para executar cada um dos stored procedures necessários a cada role.

```
--Procedures para Todos
GRANT EXECUTE On Alterar_Utilizador TO Clients,Administradores,Restaurantes
--Procedures dos Clientes
GRANT EXECUTE On Ver_Restaurante_Especifico_Cliente TO Clients
GRANT EXECUTE ON Ver_Horario_Restaurantes TO Clients
GRANT EXECUTE On Ver_Todos_Restaurantes_Cliente TO Clients
GRANT EXECUTE ON Ver_Horario_Restaurante_Especifico TO Clients
GRANT EXECUTE On Alterar_Cliente TO Clients
GRANT EXECUTE On Ver_RestaurantesFav TO Clients
GRANT EXECUTE On Adicionar_PratoFavorito TO Clients
GRANT EXECUTE On Adicionar_RestauranteFavorito TO Clients
GRANT EXECUTE On Ver_Pratos_Do_Dia TO Clients
GRANT EXECUTE On Ver_Pratos_Do_Dia_Restaurante TO Clients
GRANT EXECUTE On Ver_Pratos TO Clients
GRANT EXECUTE On Ver_Pratos_Restaurante TO Clients
GRANT EXECUTE ON Ver_Servicos_Rest_Especifico TO Clients
GRANT EXECUTE ON Ver_Servicos TO Clients
GRANT EXECUTE ON Ver_Localizacao_Rest_Especifico TO Clients
GRANT EXECUTE ON Ver_Localizacao_Rest TO Clients
```

```
--Procedures dos Administradores
GRANT EXECUTE ON Novo_Admin TO Administradores
GRANT EXECUTE ON Confirmar_Email TO Administradores
GRANT EXECUTE ON Autorizar_Restaurante TO Administradores
GRANT EXECUTE ON Alterar_Admin TO Administradores
GRANT EXECUTE ON Bloquear_Utilizador TO Administradores
GRANT EXECUTE ON Ver_Clientes TO Administradores
GRANT EXECUTE ON Ver_Restaurantes_Admin TO Administradores
GRANT EXECUTE ON Ver_Clientes_Bloqueados TO Administradores
GRANT EXECUTE ON Ver_Restaurantes_Autorizados TO Administradores
GRANT EXECUTE ON Ver_Restaurantes_Nao_Autorizados TO Administradores
GRANT EXECUTE On Novo_Utilizador TO Administradores
GRANT EXECUTE ON Eliminar_Utilizador TO Administradores
GRANT EXECUTE ON Eliminar_Restaurante TO Administradores
GRANT EXECUTE ON Eliminar_Cliente TO Administradores
--Procedures dos Restaurantes
GRANT EXECUTE ON Criar_Prato_Do_Dia TO Restaurantes
GRANT EXECUTE ON Criar_Prato TO Restaurantes
GRANT EXECUTE ON Alterar_Pratos_Restaurante TO Restaurantes
GRANT EXECUTE ON Eliminar_Pratos_Restaurante TO Restaurantes
GRANT EXECUTE ON Eliminar_Pratos_Do_Dia_Restaurante TO Restaurantes
GRANT EXECUTE ON Alterar_Restaurante TO Restaurantes
```

Figura 3.3 – Atribuição das permissões de execução de cada uma das procedures

Cada um dos stored procedures garantido às roles, tal como é demonstrado nas figuras acima, será explanado nas seguintes páginas deste relatório.

Stored procedures

Em todos os stored procedures verificamos se existe algum tipo de erro, caso não exista nenhum erro será feito um commit, ou seja, a ação que o stored procedure tentou realizar pode ser efetuada na base de dados, no exemplo acima será eliminado o utilizador, até então a ação do stored procedure estava em stand-by. Caso exista algum erro a ação do stored procedure não é realizada, através do comando ROLLBACK. Esta explicação não será mais abordada ao longo deste trabalho, pois funciona de forma semelhante nos restantes procedures.

```
CREATE PROCEDURE Eliminar_Utilizador
    @id_Utilizador INTEGER
AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRANSACTION

IF EXISTS (SELECT Utilizador.id_Utilizador FROM Utilizador WHERE Utilizador.id_Utilizador = @id_Utilizador)
    BEGIN
        DELETE FROM BloquearUtilizador
        WHERE (id_Utilizador = @id_Utilizador)
        DELETE FROM Utilizador
        WHERE (id_Utilizador = @id_Utilizador)
    END
ELSE
    BEGIN
        PRINT 'Utilizador nao existente'
    END

    IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
    GOTO ERRO
    COMMIT
return 1
    ERRO:
        ROLLBACK
        RETURN -1
GO
```

Figura 4.1 – Eliminar_Utilizador

Neste stored procedure teremos como parâmetro de entrada um id de um certo utilizador onde vai ser preciso para apagar das tabelas BloquearUtilizador e Utilizador o utilizador que lhe pertença esse id, para isso verificamos sempre nas duas tabelas se existe esse utilizador.

Procedures criados com a mesma funcionalidade:

- Eliminar_Cliente;
- Eliminar_Admin;
- Eliminar_Restaurante;
- Eliminar_Pratos_Do_Dia_Restaurante;
- Eliminar_Pratos_Restaurante;

Estes dois últimos deram mais trabalho, pois para eliminar um prato, teríamos de o eliminar das outras tabelas em que ele já se encontrava (efeito cascade).

```
CREATE PROCEDURE Ver_Pratos_do_Dia
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

    IF EXISTS (SELECT PratoDoDia.id_Prato FROM PratoDoDia)
        BEGIN
            SELECT * FROM PratoDoDia
        END
    ELSE
        BEGIN
            RAISERROR('Nao ha pratos do dia', 16, 1)
        END

    if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
    GOTO ERRO
    COMMIT
return 1
    ERRO:
        ROLLBACK
        RETURN -1
GO
```

Figura 4.2 – Ver_Pratos_do_Dia

Este stored procedure é bastante simples, o seu objetivo é simplesmente mostrar todos os pratos do dia. Para isso fazemos um select em toda tabela PratoDoDia.

Procedures criados com a mesma funcionalidade:

- Ver_Pratos;
- Ver_Horario_Restaurantes;
- Ver_Todos_Restaurantes_Cliente;
- Ver_Clientes;
- Ver_Restaurantes_Admin;
- Ver_Restaurantes_Autorizados;
- Ver_Restaurantes_Nao_Autorizados;
- Ver_RestaurantesFav;
- Ver_Pratos_Do_Dia;
- Ver_Servicos;
- Ver_Clientes_Bloqueados;
- Ver_Localizacao_Rest;
- Ver_Pratos_Restaurante;
- Ver_Restaurante_Especifico_Cliente;
- Ver_Horario_Restaurante_Especifico;
- Ver_Localizacao_Rest_Especifico;
- Ver_Servicos_Rest_Especifico.

Estes quatro últimos deram mais trabalho, isto porque, apenas aparecerá coisas relacionadas com o id desejado.

```
CREATE PROCEDURE Novo_Admin
    @id_Admin INTEGER, @nome VARCHAR(20)
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

    IF EXISTS (SELECT Utilizador.id_Utilizador FROM Utilizador WHERE Utilizador.id_Utilizador = @id_Admin)
        BEGIN
            INSERT INTO Administrador
            VALUES(@id_Admin, @nome)
        END
    ELSE
        BEGIN
            RAISERROR('Utilizador invalido', 16, 1)
        END

    if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
    GOTO ERRO
    COMMIT
return 1
    ERRO:
        ROLLBACK
        RETURN -1
GO
```

Figura 4.3 – Novo_Admin

Neste caso iremos criar um administrador, através de um utilizador já criado. Para isso utilizamos o id e damos um novo nome para assim criar um novo utilizador como administrador.

Procedures criados com a mesma funcionalidade:

- Novo_Cliente;
- Novo_Utilizador;
- Novo_Admin;
- Criar_Restaurante;
- Criar_Prato;
- Criar_Prato_Do_Dia;
- Adicionar_PratoFavorito;
- Adicionar_RestauranteFavorito;

```
CREATE PROCEDURE Confirmar_Email
    @id_Utilizador VARCHAR(50)
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

    IF EXISTS (SELECT Client.id_Client FROM Client WHERE Client.id_Client = @id_Utilizador)
        BEGIN
            IF EXISTS(SELECT Client.id_Client FROM Client WHERE Client.id_Client = @id_Utilizador AND Client.email_Confirmado = 1)
                BEGIN
                    RAISERROR('Email ja confirmado', 16, 1)
                END
            ELSE
                BEGIN
                    UPDATE Client
                    SET Email_Confirmado=1
                    WHERE id_Client=@id_Utilizador
                END
        END
    ELSE
        BEGIN
            RAISERROR('Cliente inexistente', 16, 1)
        END

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

ERRO:
    ROLLBACK
    RETURN -1
GO
```

Figura 4.4 – Confirmar_Email

Neste store procedure é atualizado o valor de email_confirmado na tabela Client, para isso utilizamos o id do utilizador, fazemos um update no valor de email_confirmado para esse mesmo utilizador.

```
CREATE PROCEDURE Alterar_Admin
     @nome VARCHAR(50), @id_Admin INTEGER
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

    IF EXISTS (SELECT Administrador.id_Admin FROM Administrador WHERE Administrador.id_Admin = @id_Admin)
        BEGIN
            UPDATE Administrador
            SET nome = @nome
            WHERE id_Admin = @id_Admin
        END
    ELSE
        BEGIN
            RAISERROR('Admin inexistente', 16, 1)
        END

    if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
    GOTO ERRO
    COMMIT
return 1
    ERRO:
        ROLLBACK
        RETURN -1
GO
```

Figura 4.5 – Alterar_Admin

Este stored procedure serve para alterar o nome de um Administrador, para isso, uma vez mais teremos como parâmetros um id de administrador já existente e um novo nome que irá substituir o existente. Assim iremos fazer um update na tabela Administrador alterando o seu nome.

Procedures criados com a mesma funcionalidade:

- Alterar_Cliente;
- Alterar_Utilizador;
- Alterar_Restaurante;
- Alterar_Pratos_Restaurante;

```
CREATE PROCEDURE Autorizar_Restaurante
    @id_A INTEGER, @id_R INTEGER
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

    IF EXISTS (SELECT Administrador.id_Admin FROM Administrador WHERE Administrador.id_Admin = @id_A)
        BEGIN
            IF EXISTS (SELECT Restaurante.id_Restaurante FROM Restaurante WHERE Restaurante.id_Restaurante = @id_R)
                BEGIN
                    IF EXISTS (SELECT Autorizar.id_Restaurante FROM Autorizar WHERE Autorizar.id_Restaurante = @id_R)
                        BEGIN
                            RAISERROR('Restaurante ja autorizado', 16, 1)
                        END
                    ELSE
                        BEGIN
                            INSERT INTO Autorizar(id_Admin,id_Restaurante,data)
                            VALUES (@id_A,@id_R,GETDATE())
                            UPDATE Restaurante
                            SET validado = 1
                        WHERE @id_R = id_Restaurante
                    END
                END
            ELSE
                BEGIN
                    RAISERROR('Restaurante inexistente', 16, 1)
                END
        END
    ELSE
        BEGIN
            RAISERROR('Admin inexistente', 16, 1)
        END

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

COMMIT
RETURN 1

ERRO:
    ROLLBACK
    RETURN -1
GO
```

Figura 4.6 – Autorizar_Restaurante

Este stored procedure tem como função validar o pedido de um restaurante (para se tornar um restaurante). Primeiramente, precisamos de dois parâmetros, o id do Admin e o id do Restaurante, porque os pedidos feitos por restaurantes serão enviados para uma tabela chamada "Autorizar", em que aparecerá o id do admin que aceitou, o restaurante que fez o pedido e a data que foi aceite esse mesmo restaurante. De seguida, iremos atualizar a tabela dos restaurantes, através do id do restaurante, e pôr a coluna "validado" com valor 1 (antes encontrava-se a 0, pois por default o restaurante não se encontra validado).

```
CREATE PROCEDURE Bloquear_Utilizador
    @id_Admin INTEGER, @id_Utilizador INTEGER, @duracao INTEGER, @motivo VARCHAR(50)
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

    IF EXISTS (SELECT Administrador.id_Admin FROM Administrador WHERE Administrador.id_Admin = @id_Admin)
        BEGIN
            IF EXISTS (SELECT Utilizador.id_Utilizador FROM Utilizador WHERE Utilizador.id_Utilizador = @id_Utilizador)
                BEGIN
                    IF EXISTS (SELECT BloquearUtilizador.id_Utilizador FROM BloquearUtilizador WHERE BloquearUtilizador.id_Utilizador = @id_Utilizador)
                        BEGIN
                            RAISERROR('Utilizador ja bloqueado', 16, 1)
                        END
                    ELSE
                        BEGIN
                            INSERT INTO BloquearUtilizador(id_Admin,id_Utilizador,duracao,motivo)
                            VALUES (@id_Admin,@id_Utilizador,@duracao,@motivo)
                        END
                END
            ELSE
                BEGIN
                    RAISERROR('Utilizador inexistente', 16, 1)
                END
        END
    ELSE
        BEGIN
            RAISERROR('Admin inexistente', 16, 1)
        END

    if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
    GOTO ERRO
    COMMIT
return 1
    ERRO:
        ROLLBACK
        RETURN -1
GO
```

Figura 4.7 – Bloquear_Utilizador

Este stored procedure tem como função bloquear um utilizador com o respetivo motivo e duração. Primeiramente, precisamos de quatro parâmetros, id do Admin, id do Utilizador, duração e motivo, porque os bloqueios de utilizadores serão enviados para a tabela “BloquearUtilizador”, em que aparecerá o id do Admin que bloqueou, o id do Utilizador bloqueado, a duração do bloqueio e o respetivo motivo.

Conclusão

As políticas de segurança e acesso aos dados são um aspeto fundamental em qualquer base de dados centralizada, senão seria uma anarquia total e qualquer um poderia manipular os dados conforme quisesse, o que não é o pretendido em nenhuma ocasião. Para resolver esta situação, foram utilizadas as permissões e estabelecida uma estrita política de acesso aos dados. Para além deste problema, outra situação que foi resolvida neste trabalho, foram os problemas de concorrência, visto que é um tema central no desenvolvimento das bases de dados, pois qualquer transação de uma base de dados tem de ser ACID, ou seja, atómica, consistente/coerente, isolada e durável, o que isto significa é que, quando um utilizador inicia uma transação, esta, ao ser aplicada, tem de ser também aplicada para todos os outros utilizadores, ou seja, deve ser coerente para toda a gente. O uso das transações desta forma, permite também que duas transações que estejam a aceder aos mesmos dados, ao mesmo tempo, não entrem em conflito, devido à possibilidade de rollback.

Anexo – Scripts Utilizados

```
--Trabalho realizado por Vítor Neto 68717, João Leal 68719, Hugo Anes 68571 e Leandro Coelho 68541--
--Baseado na base de dados desenvolvida a LAWBD--
--Base de dados--
USE master
GO
CREATE DATABASE TABD_TP1
--DROP DATABASE TABD_TP1
USE TABD_TP1
CREATE TABLE Utilizador(
	id_Utilizador INTEGER PRIMARY KEY,
	username VARCHAR(50) UNIQUE NOT NULL,
	pass VARCHAR(50) NOT NULL,
	email VARCHAR(50) NOT NULL,
	tipo INTEGER,
	CHECK(tipo LIKE '0' OR tipo LIKE '1' OR tipo LIKE '2'), --0 = Admin, 1 = Client, 2 = Restaurante
	CHECK(email LIKE '%@%.%')
);

CREATE TABLE Client(
	id_Client INTEGER,
	nome VARCHAR(50) NOT NULL,
	email_Confirmado BIT DEFAULT 0,
	FOREIGN KEY(id_Client) REFERENCES Utilizador(id_Utilizador),
	PRIMARY KEY(id_Client)
);

CREATE TABLE Administrador(
	id_Admin INTEGER,
	nome VARCHAR(50) NOT NULL,
	FOREIGN KEY(id_Admin) REFERENCES Utilizador(id_Utilizador),
	PRIMARY KEY(id_Admin)
);

CREATE TABLE Restaurante(
	id_Restaurante INTEGER,
	nome VARCHAR(50) NOT NULL,
	validado BIT DEFAULT 0 NOT NULL,
	morada VARCHAR(50) NOT NULL,
	gps VARCHAR(50) NOT NULL,
	telefone NUMERIC(9) UNIQUE NOT NULL,
	dia_descanso VARCHAR(50) NOT NULL,
	tipo_takeaway BIT DEFAULT 0,
	tipo_local BIT DEFAULT 0,
	tipo_entrega BIT DEFAULT 0,
	hora_de_abertura CHAR(5) NOT NULL,
	hora_de_fecho CHAR(5) NOT NULL,
	descricao VARCHAR(250),
	foto VARCHAR(300),
	FOREIGN KEY(id_Restaurante) REFERENCES Utilizador(id_Utilizador),
	PRIMARY KEY(id_Restaurante),
	CHECK (hora_de_abertura LIKE '[0-1][0-9]:[0-5][0-9]' OR 	hora_de_abertura Like '[2][0-4]:[0-5][0-9]'),
	CHECK (hora_de_fecho LIKE '[0-1][0-9]:[0-5][0-9]' OR hora_de_fecho Like '[2][0-4]:[0-5][0-9]'),
	CHECK (telefone BETWEEN 100000000 AND 999999999),
	CHECK (tipo_takeaway LIKE '1' OR tipo_local LIKE '1' OR tipo_entrega LIKE '1')
);

CREATE TABLE PratoDoDia(
```

```
	id_Prato INTEGER PRIMARY KEY,
	nome VARCHAR(250) NOT NULL,
	descricao VARCHAR(250),
	tipo_prato VARCHAR(250) NOT NULL,
	foto VARCHAR(300) NOT NULL,
	CHECK(tipo_prato LIKE 'Vegan' OR tipo_prato LIKE 'Peixe' or tipo_prato LIKE 'Carne')
);

CREATE TABLE CriarPrato(
	id_Restaurante INTEGER NOT NULL,
	id_Prato INTEGER NOT NULL,
	preco MONEY NOT NULL,
	dia_semana_prato VARCHAR(30) NOT NULL,
	apagado BIT NOT NULL DEFAULT 0, --0 não está apagado, 1 está
	PRIMARY KEY(id_Restaurante, id_Prato),
	FOREIGN KEY(id_Restaurante) REFERENCES Restaurante(id_Restaurante),
	FOREIGN KEY(id_Prato) REFERENCES PratoDoDia(id_Prato),
	CHECK(dia_semana_prato LIKE 'Segunda feira' OR dia_semana_prato LIKE 'Terça feira' OR
dia_semana_prato LIKE 'Quarta feira' or dia_semana_prato LIKE 'Quinta feira' OR
dia_semana_prato LIKE 'Sexta feira' OR dia_semana_prato LIKE 'Sabado' OR dia_semana_prato
LIKE 'Domingo')
	);

CREATE TABLE AdicionarPratoFav(
	id_Client INTEGER NOT NULL,
	id_Prato INTEGER NOT NULL,
	data_adicao DATE NOT NULL,
	PRIMARY KEY(id_Client, id_Prato),
	FOREIGN KEY(id_Client) REFERENCES Client(id_Client),
	FOREIGN KEY(id_Prato) REFERENCES PratoDoDia(id_Prato)
);

CREATE TABLE AdicionarRestFav(
	id_Restaurante integer not null,
	id_Cliente integer not null,
	data_adicao DATE NOT NULL,
	Primary key(id_Restaurante, id_Cliente),
	FOREIGN KEY(id_Restaurante) REFERENCES Restaurante(id_Restaurante),
	FOREIGN KEY(id_Cliente) REFERENCES Client(id_Client)
);

CREATE TABLE BloquearUtilizador(
	id_Admin INTEGER PRIMARY KEY,
	id_Utilizador INTEGER,
	duracao INTEGER NOT NULL,
	motivo VARCHAR(250) NOT NULL,
	FOREIGN KEY(id_Utilizador) REFERENCES Utilizador(id_Utilizador),
	FOREIGN KEY(id_Admin) REFERENCES Administrador(id_Admin)
);

CREATE TABLE Autorizar(
	id_Admin INTEGER,
	id_Restaurante INTEGER,
	data DATETIME NOT NULL DEFAULT GETDATE(),
	PRIMARY KEY(id_Admin, id_Restaurante),
	FOREIGN KEY(id_Admin) REFERENCES Administrador(id_Admin),
	FOREIGN KEY(id_Restaurante) REFERENCES Restaurante(id_Restaurante)
	);
--Trabalho realizado por Vítor Neto 68717, João Leal 68719, Hugo Anes 68571 e Leandro Coelho
68541--
--Políticas de segurança e acesso a dados centralizados--

USE TABD_TP1
GO
```

```
--Logins--
CREATE LOGIN Administrador WITH PASSWORD = '1'
CREATE LOGIN Client WITH PASSWORD = '1'
CREATE LOGIN Restaurante WITH PASSWORD = '1'
CREATE LOGIN Dono WITH PASSWORD = '1'

--Users--
CREATE USER Restaurante1 FOR LOGIN Restaurante
CREATE USER Administrador1 FOR LOGIN Administrador
CREATE USER Client1 FOR LOGIN Client
CREATE USER Dono FOR LOGIN Dono
CREATE USER Visitante WITHOUT LOGIN

--Roles--
CREATE ROLE Administradores
CREATE ROLE Clients
CREATE ROLE Restaurantes

--Adicionar membros a cada uma das roles--
ALTER ROLE Clients ADD MEMBER Client1
ALTER ROLE Administradores ADD MEMBER Administrador1
ALTER ROLE Restaurantes ADD MEMBER Restaurante1

--Atribuição de Permissões nas Tabelas

--Permissões para dono BD
GRANT SELECT, INSERT, UPDATE, DELETE TO Dono
GRANT EXECUTE ON Eliminar_Admin TO Dono

--Permissões para os Restaurantes
GRANT SELECT ON Client to Restaurantes
GRANT SELECT, INSERT, UPDATE, DELETE ON CriarPrato TO Restaurantes
GRANT SELECT, INSERT, UPDATE, DELETE ON PratoDoDia TO Restaurantes
GRANT SELECT, UPDATE ON Restaurante TO Restaurantes
GRANT SELECT, UPDATE ON Utilizador TO Restaurantes

--Permissões para os Clients
GRANT SELECT ON Restaurante TO Clients
GRANT SELECT ON CriarPrato TO Clients
GRANT SELECT ON PratoDoDia TO Clients
GRANT SELECT ON BloquearUtilizador TO Clients
GRANT SELECT,UPDATE ON Utilizador TO Clients
GRANT SELECT,INSERT,UPDATE,DELETE ON AdicionarPratoFav TO Clients
GRANT SELECT,INSERT,UPDATE,DELETE ON AdicionarRestFav TO Clients
GRANT SELECT,UPDATE ON Client TO Clients

--Permissões para os Administradores
GRANT SELECT, INSERT, UPDATE, DELETE ON BloquearUtilizador TO Administradores
GRANT SELECT, INSERT, UPDATE ON Autorizar TO Administradores
GRANT SELECT, DELETE ON Utilizador to Administradores
GRANT SELECT, DELETE ON Client TO Administradores
GRANT SELECT, INSERT, UPDATE ON Administrador TO Administradores
GRANT SELECT, DELETE ON Restaurante TO Administradores
GRANT SELECT on CriarPrato TO Administradores
GRANT SELECT on PratoDoDia TO Administradores

--Permissões e procedures para Visitante
GRANT SELECT ON Restaurante TO Visitante
GRANT SELECT ON CriarPrato TO Visitante
GRANT SELECT ON PratoDoDia TO Visitante
GRANT EXECUTE ON Criar_Restaurante TO Visitante
GRANT EXECUTE On Novo_Cliente TO Visitante
GRANT EXECUTE On Novo_Utilizador TO Visitante
```

```
--Procedures para Todos
GRANT EXECUTE On Alterar_Utilizador TO Clients,Administradores,Restaurantes

--Procedures dos Clientes
GRANT EXECUTE On Ver_Restaurante_Especifico_Cliente TO Clients
GRANT EXECUTE ON Ver_Horario_Restaurantes TO Clients
GRANT EXECUTE On Ver_Todos_Restaurantes_Cliente TO Clients
GRANT EXECUTE ON Ver_Horario_Restaurante_Especifico TO Clients
GRANT EXECUTE On Alterar_Cliente TO Clients
GRANT EXECUTE On Ver_RestaurantesFav TO Clients
GRANT EXECUTE On Adicionar_PratoFavorito TO Clients
GRANT EXECUTE On Adicionar_RestauranteFavorito TO Clients
GRANT EXECUTE On Ver_Pratos_Do_Dia TO Clients
GRANT EXECUTE On Ver_Pratos_Do_Dia_Restaurante TO Clients
GRANT EXECUTE On Ver_Pratos TO Clients
GRANT EXECUTE On Ver_Pratos_Restaurante TO Clients
GRANT EXECUTE ON Ver_Servicos_Rest_Especifico TO Clients
GRANT EXECUTE ON Ver_Servicos TO Clients
GRANT EXECUTE ON Ver_Localizacao_Rest_Especifico TO Clients
GRANT EXECUTE ON Ver_Localizacao_Rest TO Clients

--Procedures dos Administradores
GRANT EXECUTE ON Novo_Admin TO Administradores
GRANT EXECUTE ON Confirmar_Email TO Administradores
GRANT EXECUTE ON Autorizar_Restaurante TO Administradores
GRANT EXECUTE ON Alterar_Admin TO Administradores
GRANT EXECUTE ON Bloquear_Utilizador TO Administradores
GRANT EXECUTE ON Ver_Clientes TO Administradores
GRANT EXECUTE ON Ver_Restaurantes_Admin TO Administradores
GRANT EXECUTE ON Ver_Clientes_Bloqueados TO Administradores
GRANT EXECUTE ON Ver_Restaurantes_Autorizados TO Administradores
GRANT EXECUTE ON Ver_Restaurantes_Nao_Autorizados TO Administradores
GRANT EXECUTE On Novo_Utilizador TO Administradores
GRANT EXECUTE ON Eliminar_Utilizador TO Administradores
GRANT EXECUTE ON Eliminar_Restaurante TO Administradores
GRANT EXECUTE ON Eliminar_Cliente TO Administradores

--Procedures dos Restaurantes
GRANT EXECUTE ON Criar_Prato_Do_Dia TO Restaurantes
GRANT EXECUTE ON Criar_Prato TO Restaurantes
GRANT EXECUTE ON Alterar_Pratos_Restaurante TO Restaurantes
GRANT EXECUTE ON Eliminar_Pratos_Restaurante TO Restaurantes
GRANT EXECUTE ON Eliminar_Pratos_Do_Dia_Restaurante TO Restaurantes
GRANT EXECUTE ON Alterar_Restaurante TO Restaurantes

--Trabalho realizado por Vítor Neto 68717, João Leal 68719, Hugo Anes 68571 e Leandro Coelho
68541--
--Procedures--

USE TABD_TP1
GO

--Procedures Necessários aos Utilizadores/Visitantes-----------------------------------------
-------------------------------------------------------------------------------------------
-----------------------------------------------------
CREATE PROCEDURE Novo_Utilizador
       @id_Utilizador INTEGER, @username VARCHAR(50), @pass VARCHAR(50), @email
VARCHAR(50) ,@tipo INTEGER
AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRANSACTION

       INSERT INTO Utilizador
```

```
	VALUES (@id_Utilizador, @username, @pass, @email, @tipo)

	IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
	GOTO ERRO
	COMMIT
RETURN 1
	ERRO:
		ROLLBACK
		RETURN -1
GO
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----------------------------------------
CREATE PROCEDURE Eliminar_Utilizador
    @id_Utilizador INTEGER
AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRANSACTION

IF EXISTS (SELECT Utilizador.id_Utilizador FROM Utilizador WHERE Utilizador.id_Utilizador =
@id_Utilizador)
	BEGIN
		DELETE FROM BloquearUtilizador
		WHERE (id_Utilizador = @id_Utilizador)
		DELETE FROM Utilizador
		WHERE (id_Utilizador = @id_Utilizador)
	END
ELSE
	BEGIN
		PRINT 'Utilizador nao existente'
	END

    IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
    GOTO ERRO
    COMMIT
return 1
    ERRO:
        ROLLBACK
        RETURN -1
GO
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----------------------------------------
CREATE PROCEDURE Novo_Cliente
	@id_Cliente INTEGER, @nome VARCHAR(20), @emailConf BIT
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

	IF EXISTS (SELECT Utilizador.id_Utilizador FROM Utilizador WHERE
Utilizador.id_Utilizador = @id_Cliente)
		BEGIN
			INSERT INTO Client
			VALUES(@id_Cliente, @nome, '0')
		END
	ELSE
		BEGIN
			PRINT 'Utilizador invalido'
		END

	if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
	GOTO ERRO
	COMMIT
return 1
```

```
	ERRO:
		ROLLBACK
		RETURN -1
GO
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------
CREATE PROCEDURE Alterar_Utilizador
	@username VARCHAR(50), @pass VARCHAR(50), @email VARCHAR(50), @tipo INTEGER
AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRANSACTION

IF EXISTS (SELECT Utilizador.id_Utilizador FROM Utilizador WHERE Utilizador.username =
@username)
	BEGIN
		UPDATE Utilizador
		SET Email = @email, pass = @pass,tipo = @tipo
		WHERE Username=@username
	END
ELSE
	BEGIN
		PRINT 'Utilizador nao existente'
	END

	IF(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
	GOTO ERRO
	COMMIT
return 1
	ERRO:
		ROLLBACK
		RETURN -1
GO
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------
--Procedures Necessários aos Clientes-----------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------
CREATE PROCEDURE Ver_Restaurante_Especifico_Cliente
	@id_R INTEGER
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

	IF EXISTS (SELECT Restaurante.id_Restaurante FROM Restaurante WHERE
Restaurante.id_Restaurante = @id_R)
		BEGIN
			SELECT R.nome AS 'Nome', R.morada AS 'Morada', R.gps AS 'GPS',
R.telefone AS 'Telefone', R.dia_descanso 'Dia de descanso', R.descricao AS 'Descricao',
R.foto AS 'Foto', R.hora_de_abertura AS 'Hora de abertura', R.hora_de_fecho AS 'Hora de
fecho', R.tipo_entrega 'Entrega', R.tipo_local 'Local', R.tipo_takeaway 'Takeaway'
			FROM Restaurante R
			WHERE R.id_Restaurante = @id_R
		END
	ELSE
		BEGIN
			PRINT 'Restaurante nao existente'
		END
	if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
	GOTO ERRO
```

```
	COMMIT
return 1
	ERRO:
		ROLLBACK
		RETURN -1
GO
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------
CREATE PROCEDURE Eliminar_Restaurante_Fav
     @id_Cliente integer,@id_Restaurante integer
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

	IF EXISTS (SELECT Client.id_Client FROM Client WHERE Client.id_Client = @id_Cliente)
		BEGIN
			IF EXISTS (SELECT AdicionarRestFav.id_Restaurante FROM AdicionarRestFav
WHERE AdicionarRestFav.id_Restaurante = @id_Restaurante)
				BEGIN
					DELETE FROM AdicionarRestFav
					WHERE(id_Cliente=@id_Cliente AND
id_Restaurante=@id_Restaurante)
				END
			ELSE
				BEGIN
					RAISERROR('Restaurante nao esta na lista de favoritos do
cliente', 16, 1)
				END
		END
	ELSE
		BEGIN
			RAISERROR('Cliente nao existente', 16, 1)
		END

    if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
    GOTO ERRO
    COMMIT
return 1
    ERRO:
        ROLLBACK
        RETURN -1
GO
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------
CREATE PROCEDURE Eliminar_Prato_Fav
    @id_Cliente integer,@id_Prato integer
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

	IF EXISTS (SELECT Client.id_Client FROM Client WHERE Client.id_Client = @id_Cliente)
		BEGIN
			IF EXISTS (SELECT AdicionarPratoFav.id_Prato FROM AdicionarPratoFav
WHERE AdicionarPratoFav.id_Prato = @id_Prato)
				BEGIN
					DELETE FROM AdicionarPratoFav
					WHERE(id_Client=@id_Cliente and id_Prato=@id_Prato)
				END
			ELSE
				BEGIN
					RAISERROR('Prato nao esta na lista de favoritos do
cliente', 16, 1)
```

```
                        END
                END
        ELSE
                BEGIN
                        RAISERROR('Cliente nao existente', 16, 1)
                END

    if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
    GOTO ERRO
    COMMIT
return 1
    ERRO:
        ROLLBACK
        RETURN -1
GO
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
----------------------------------------
CREATE PROCEDURE Ver_Todos_Restaurantes_Cliente
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

                SELECT R.nome AS 'Nome', R.morada AS 'Morada', R.gps AS 'GPS', R.telefone AS
'Telefone', R.dia_descanso 'Dia de descanso', R.descricao AS 'Descricao', R.foto AS 'Foto',
R.hora_de_abertura AS 'Hora de abertura', R.hora_de_fecho AS 'Hora de fecho', R.tipo_entrega
'Entrega', R.tipo_local 'Local', R.tipo_takeaway 'Takeaway'
                FROM Restaurante R

        if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
        GOTO ERRO
        COMMIT
return 1
        ERRO:
                ROLLBACK
                RETURN -1
GO
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
----------------------------------------
CREATE PROCEDURE Alterar_Cliente
        @id_Cliente INTEGER, @nome varchar(50)
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

IF EXISTS (SELECT Client.id_Client FROM Client WHERE Client.id_Client = @id_Cliente)
                BEGIN
                        UPDATE Client
                        SET  nome = @nome
                        WHERE id_Client = @id_Cliente
                END
                ELSE
                        BEGIN
                                PRINT 'Cliente nao existente'
                        END

        if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
        GOTO ERRO
        COMMIT
return 1
        ERRO:
                ROLLBACK
                RETURN -1
```

```
GO
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
-----------------------------------------
CREATE PROCEDURE Ver_Horario_Restaurantes

AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

	SELECT R.nome AS 'Nome', R.hora_de_abertura AS 'Hora de Abertura', R.hora_de_fecho AS
'Hora de Fecho', R.dia_descanso AS 'Dia de Descanso'
	FROM Restaurante R

	if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
	GOTO ERRO
	COMMIT
return 1
	ERRO:
		ROLLBACK
		RETURN -1
GO
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
-----------------------------------------
CREATE PROCEDURE Ver_Horario_Restaurante_Especifico
	@id_Restaurante INTEGER
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

	IF EXISTS (SELECT Restaurante.id_Restaurante FROM Restaurante WHERE
Restaurante.id_Restaurante = @id_R)
		BEGIN
			SELECT R.nome AS Nome, R.hora_de_abertura AS 'Hora de
Abertura',R.hora_de_fecho AS 'Hora de Fecho',R.dia_descanso AS 'Dia de Descanso'
			FROM Restaurante R
			WHERE R.id_Restaurante = @id_Restaurante
		END
		ELSE
			BEGIN
				PRINT 'Restaurante nao existente'
			END

	if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
	GOTO ERRO
	COMMIT
return 1
	ERRO:
		ROLLBACK
		RETURN -1
GO
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
-----------------------------------------
CREATE PROCEDURE Ver_RestaurantesFav--de certo cliente
	@Id_C INTEGER
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

	IF EXISTS (SELECT Client.id_Client FROM Client WHERE Client.id_Client = @Id_C)
		BEGIN
```

```
                        SELECT R.nome AS 'Nome', R.dia_descanso AS 'Dia de descanso',
R.telefone AS 'Telefone', R.tipo_takeaway AS 'Takeaway', R.tipo_local AS 'Local',
R.tipo_entrega AS 'Entrega', R.hora_de_abertura AS 'Hora Abertura', R.hora_de_fecho AS 'Hora
Fecho', R.descricao AS 'Descricao', R.foto AS 'Foto', R.validado AS 'Validado'
                        FROM Restaurante R, AdicionarRestFav F
                        WHERE F.Id_Restaurante = R.id_Restaurante and Id_Cliente = F.Id_Cliente
                END
        ELSE
                BEGIN
                        PRINT 'Cliente nao existente'
                END
        if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
        GOTO ERRO
        COMMIT
return 1
        ERRO:
                ROLLBACK
                RETURN -1
GO
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
-----------------------------------------
CREATE PROCEDURE Ver_PratosFav--de certo cliente
        @Id_C integer
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

        IF EXISTS (SELECT Client.id_Client FROM Client WHERE Client.id_Client = @Id_C)
                BEGIN
                        SELECT PD.nome AS 'Nome', PD.descricao 'Descricao', PD.tipo_prato AS
'Tipo de prato', PD.foto AS 'Foto'
                        FROM AdicionarPratoFav PF, PratoDoDia PD
                        WHERE PF.id_Prato=PD.id_Prato and PF.id_Client=@Id_C;
                END
        ELSE
                BEGIN
                        RAISERROR('Cliente nao existente', 16, 1)
                END

        if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
        GOTO ERRO
        COMMIT
return 1
        ERRO:
                ROLLBACK
                RETURN -1
GO
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
-----------------------------------------
CREATE PROCEDURE Adicionar_PratoFavorito
        @ID_C INTEGER,
        @ID_P INTEGER,
        @data DATETIME
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

        IF EXISTS (SELECT Client.id_Client FROM Client WHERE Client.id_Client = @ID_C)
                BEGIN
                        IF EXISTS (SELECT PratoDoDia.id_Prato FROM PratoDoDia WHERE
PratoDoDia.id_Prato = @ID_P)
                                BEGIN
```

```
                                        IF EXISTS (SELECT AdicionarPratoFav.id_Prato FROM
AdicionarPratoFav WHERE AdicionarPratoFav.id_Prato = @ID_P AND AdicionarPratoFav.id_Client =
@ID_C)
                                                BEGIN
                                                        RAISERROR('Prato ja esta nos favoritos deste
utilizador', 16, 1)
                                                END
                                        ELSE
                                                BEGIN
                                                        INSERT INTO AdicionarPratoFav(id_Client,
id_Prato, data_adicao)
                                                        VALUES (@ID_C, @ID_P, GETDATE())
                                                END
                                END
                        ELSE
                                BEGIN
                                        RAISERROR('Prato nao existente', 16, 1)
                                END
                END
        ELSE
                BEGIN
                        RAISERROR('Cliente nao existente', 16, 1)
                END

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

ERRO:
        ROLLBACK
        RETURN -1
GO
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------
CREATE PROCEDURE Adicionar_RestauranteFavorito
        @ID_C INTEGER,
        @ID_R INTEGER,
        @data DATETIME
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

        IF EXISTS (SELECT Client.id_Client FROM Client WHERE Client.id_Client = @ID_C)
                BEGIN
                        IF EXISTS (SELECT Restaurante.id_Restaurante FROM Restaurante WHERE
Restaurante.id_Restaurante = @ID_R)
                                BEGIN
                                        IF EXISTS (SELECT AdicionarRestFav.id_Restaurante FROM
AdicionarRestFav WHERE AdicionarRestFav.id_Restaurante = @ID_R AND
AdicionarRestFav.id_Cliente = @ID_C)
                                                BEGIN
                                                        RAISERROR('Restaurante ja esta nos favoritos
deste utilizador', 16, 1)
                                                END
                                        ELSE
                                                BEGIN
                                                        INSERT INTO
AdicionarRestFav(id_Cliente,id_Restaurante,data_adicao)
                                                        VALUES (@ID_C,@ID_R,GETDATE())
                                                END
                                END
                        ELSE
```

```
                        BEGIN
                                RAISERROR('Restaurante nao existente', 16, 1)
                        END
                END
        ELSE
                BEGIN
                        RAISERROR('Cliente nao existente', 16, 1)
                END


        IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
                GOTO ERRO
COMMIT
RETURN 1
S
ERRO:
        ROLLBACK
        RETURN -1
GO
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
-----------------------------------------
CREATE PROCEDURE Ver_Pratos_do_Dia
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

        IF EXISTS (SELECT PratoDoDia.id_Prato FROM PratoDoDia)
                BEGIN
                        SELECT * FROM PratoDoDia
                END
        ELSE
                BEGIN
                        RAISERROR('Nao ha pratos do dia', 16, 1)
                END

        if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
        GOTO ERRO
        COMMIT
return 1
        ERRO:
                ROLLBACK
                RETURN -1
GO
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
-----------------------------------------
CREATE PROCEDURE Ver_Pratos_do_Dia_Restaurante
        @id_Restaurante INTEGER
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

        IF EXISTS (SELECT Restaurante.id_Restaurante FROM Restaurante WHERE
Restaurante.id_Restaurante = @id_Restaurante)
                BEGIN
                        SELECT R.nome AS Restaurante, PD.nome AS Prato, PD.descricao AS
Descrição,PD.tipo_prato AS Tipo
                        FROM Restaurante R, PratoDoDia PD, CriarPrato CP
                        WHERE R.id_Restaurante=@id_Restaurante and PD.id_Prato=CP.id_Prato and
R.id_Restaurante=CP.id_Restaurante
                END
        ELSE
                BEGIN
```

```
                RAISERROR('Restaurante nao existente', 16, 1)
        END

    if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
    GOTO ERRO
    COMMIT
return 1
    ERRO:
        ROLLBACK
        RETURN -1
GO
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
----------------------------------------
CREATE PROCEDURE Ver_Pratos
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

    IF EXISTS (SELECT PratoDoDia.id_Prato FROM PratoDoDia)
        BEGIN
            SELECT * FROM CriarPrato
        END
    ELSE
        BEGIN
            RAISERROR('Nao existem pratos', 16, 1)
        END

    if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
    GOTO ERRO
    COMMIT
return 1
    ERRO:
        ROLLBACK
        RETURN -1
GO
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
----------------------------------------
CREATE PROCEDURE Ver_Pratos_Restaurante
    @id_Restaurante INTEGER
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

    IF EXISTS (SELECT Restaurante.id_Restaurante FROM Restaurante WHERE
Restaurante.id_Restaurante = @id_Restaurante)
        BEGIN
            SELECT R.nome AS Nome, PD.nome AS 'Prato do dia', PD.descricao AS
Descrição,PD.tipo_prato AS Tipo, CP.preco AS Preço,CP.dia_semana_prato AS 'Dia da Semana'
            FROM Restaurante R, CriarPrato CP, PratoDoDia PD
            WHERE R.id_Restaurante = @id_Restaurante and CP.id_Prato = PD.id_Prato
        END
    ELSE
        BEGIN
            RAISERROR('Restaurante nao existente', 16, 1)
        END

    if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
    GOTO ERRO
    COMMIT
return 1
    ERRO:
        ROLLBACK
```

```
		RETURN -1
GO
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------
CREATE PROCEDURE Ver_Servicos_Rest_Especifico
	@id_Rest INTEGER
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

	IF EXISTS (SELECT Restaurante.id_Restaurante FROM Restaurante WHERE
Restaurante.id_Restaurante = @id_Rest)
		BEGIN
			SELECT tipo_entrega, tipo_local, tipo_takeaway FROM Restaurante
			WHERE id_Restaurante = @id_Rest
		END
	ELSE
		BEGIN
			RAISERROR('Restaurante nao existente', 16, 1)
		END

	if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
	GOTO ERRO
	COMMIT
return 1
	ERRO:
		ROLLBACK
		RETURN -1
GO
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------
CREATE PROCEDURE Ver_Servicos
	@tt BIT, @te BIT, @tl BIT
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

	SELECT * FROM Restaurante
	WHERE tipo_entrega = @te OR tipo_local = @tl OR tipo_takeaway = @tt

	if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
	GOTO ERRO
	COMMIT
return 1
	ERRO:
		ROLLBACK
		RETURN -1
GO
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------
CREATE PROCEDURE Ver_Localizacao_Rest_Especifico
	@id_Rest INTEGER
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

	IF EXISTS (SELECT Restaurante.id_Restaurante FROM Restaurante WHERE
Restaurante.id_Restaurante = @id_Rest)
		BEGIN
			SELECT morada AS Morada FROM Restaurante
			WHERE id_Restaurante = @id_Rest
```

```
		END
	ELSE
		BEGIN
			RAISERROR('Restaurante nao existente', 16, 1)
		END

	if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
	GOTO ERRO
	COMMIT
return 1
	ERRO:
		ROLLBACK
		RETURN -1
GO
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
----------------------------------------
CREATE PROCEDURE Ver_Localizacao_Rest
	@morada VARCHAR(50)
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

	IF EXISTS (SELECT Restaurante.id_Restaurante FROM Restaurante WHERE
Restaurante.morada = @morada)
		BEGIN
			SELECT * FROM Restaurante
			WHERE morada = @morada
		END
	ELSE
		BEGIN
			RAISERROR('Morada nao corresponde a nenhum restaurante existente', 16,
1)
		END

	if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
	GOTO ERRO
	COMMIT
return 1
	ERRO:
		ROLLBACK
		RETURN -1
GO
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
----------------------------------------
--Procedures Necessários aos Administradores-----------------------------------------------
-----------------------------------------------------------------------------------------
----------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
----------------------------------------
CREATE PROCEDURE Novo_Admin
	@id_Admin INTEGER, @nome VARCHAR(20)
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

	IF EXISTS (SELECT Utilizador.id_Utilizador FROM Utilizador WHERE
Utilizador.id_Utilizador = @id_Admin)
		BEGIN
			INSERT INTO Administrador
			VALUES(@id_Admin, @nome)
		END
```

```
	ELSE
		BEGIN
			RAISERROR('Utilizador invalido', 16, 1)
		END

	if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
	GOTO ERRO
	COMMIT
return 1
	ERRO:
		ROLLBACK
		RETURN -1
GO
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
----------------------------------------
CREATE PROCEDURE Eliminar_Utilizador
    @id_Utilizador INTEGER
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

	IF EXISTS (SELECT Utilizador.id_Utilizador FROM Utilizador WHERE
Utilizador.id_Utilizador = @id_Utilizador)
		BEGIN
			DELETE FROM BloquearUtilizador
			WHERE(id_Utilizador=@id_Utilizador)
			DELETE FROM Utilizador
			WHERE(id_Utilizador=@id_Utilizador)
		END
	ELSE
		BEGIN
			RAISERROR('Utilizador inexistente', 16, 1)
		END

    if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
    GOTO ERRO
    COMMIT
return 1
    ERRO:
        ROLLBACK
        RETURN -1
GO
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
----------------------------------------
CREATE PROCEDURE Eliminar_Restaurante
    @id_Restaurante INTEGER
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

	IF EXISTS (SELECT Restaurante.id_Restaurante FROM Restaurante WHERE
Restaurante.id_Restaurante = @id_Restaurante)
		BEGIN
			DELETE FROM AdicionarRestFav
			WHERE(id_Restaurante=@id_Restaurante)
			DELETE FROM Restaurante
			WHERE(id_Restaurante=@id_Restaurante)
		END
	ELSE
		BEGIN
			RAISERROR('Restaurante inexistente', 16, 1)
		END
```

```
    if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
    GOTO ERRO
    COMMIT
return 1
    ERRO:
        ROLLBACK
        RETURN -1
GO
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
----------------------------------------
CREATE PROCEDURE Eliminar_Admin
    @id_Admin INTEGER
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

       IF EXISTS (SELECT Administrador.id_Admin FROM Administrador WHERE
Administrador.id_Admin = @id_Admin)
              BEGIN
                     DELETE FROM BloquearUtilizador
                     WHERE(id_Admin=@id_Admin)
                     DELETE FROM Administrador
                     WHERE(id_Admin=@id_Admin)
              END
       ELSE
              BEGIN
                     RAISERROR('Admin inexistente', 16, 1)
              END

    if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
    GOTO ERRO
    COMMIT
return 1
    ERRO:
        ROLLBACK
        RETURN -1
GO
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
----------------------------------------
CREATE PROCEDURE Eliminar_Cliente
    @id_Cliente INTEGER
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

       IF EXISTS (SELECT Client.id_Client FROM Client WHERE Client.id_Client = @id_Cliente)
              BEGIN
                     DELETE FROM AdicionarPratoFav
                     WHERE(id_Client=@id_Cliente)
                     DELETE FROM AdicionarRestFav
                     WHERE(id_Cliente=@id_Cliente)
                     DELETE FROM Client
                     WHERE(id_Client=@id_Cliente)
              END
       ELSE
              BEGIN
                     RAISERROR('Cliente inexistente', 16, 1)
              END

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

```
    COMMIT
return 1
    ERRO:
        ROLLBACK
        RETURN -1
GO
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------
CREATE PROCEDURE Confirmar_Email
      @id_Utilizador VARCHAR(50)
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

      IF EXISTS (SELECT Client.id_Client FROM Client WHERE Client.id_Client =
@id_Utilizador)
            BEGIN
                  IF EXISTS(SELECT Client.id_Client FROM Client WHERE Client.id_Client =
@id_Utilizador AND Client.email_Confirmado = 1)
                        BEGIN
                              RAISERROR('Email ja confirmado', 16, 1)
                        END
                  ELSE
                        BEGIN
                              UPDATE Client
                              SET Email_Confirmado=1
                              WHERE id_Client=@id_Utilizador
                        END
            END
      ELSE
            BEGIN
                  RAISERROR('Cliente inexistente', 16, 1)
            END

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

ERRO:
      ROLLBACK
      RETURN -1
GO
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------
CREATE PROCEDURE Autorizar_Restaurante
    @id_A INTEGER, @id_R INTEGER
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

      IF EXISTS (SELECT Administrador.id_Admin FROM Administrador WHERE
Administrador.id_Admin = @id_A)
            BEGIN
                  IF EXISTS (SELECT Restaurante.id_Restaurante FROM Restaurante WHERE
Restaurante.id_Restaurante = @id_R)
                        BEGIN
                              IF EXISTS (SELECT Autorizar.id_Restaurante FROM Autorizar
WHERE Autorizar.id_Restaurante = @id_R)
                                    BEGIN
                                          RAISERROR('Restaurante ja autorizado', 16,
1)
```

```
                                        END
                                ELSE
                                        BEGIN
                                                INSERT INTO
Autorizar(id_Admin,id_Restaurante,data)
                                                VALUES (@id_A,@id_R,GETDATE())
                                                UPDATE Restaurante
                                                SET validado = 1
                                        WHERE @id_R = id_Restaurante
                                END
                        END
                ELSE
                        BEGIN
                                RAISERROR('Restaurante inexistente', 16, 1)
                        END
        END
    ELSE
        BEGIN
                RAISERROR('Admin inexistente', 16, 1)
        END

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

COMMIT
RETURN 1

ERRO:
    ROLLBACK
    RETURN -1
GO
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----------------------------------------
CREATE PROCEDURE Alterar_Admin
        @nome VARCHAR(50), @id_Admin INTEGER
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

    IF EXISTS (SELECT Administrador.id_Admin FROM Administrador WHERE
Administrador.id_Admin = @id_Admin)
        BEGIN
                UPDATE Administrador
                SET nome = @nome
                WHERE id_Admin = @id_Admin
        END
    ELSE
        BEGIN
                RAISERROR('Admin inexistente', 16, 1)
        END

    if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
    GOTO ERRO
    COMMIT
return 1
    ERRO:
        ROLLBACK
        RETURN -1
GO
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----------------------------------------
CREATE PROCEDURE Bloquear_Utilizador
```

```
	@id_Admin INTEGER, @id_Utilizador INTEGER, @duracao INTEGER, @motivo VARCHAR(50)
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

	IF EXISTS (SELECT Administrador.id_Admin FROM Administrador WHERE
Administrador.id_Admin = @id_Admin)
		BEGIN
			IF EXISTS (SELECT Utilizador.id_Utilizador FROM Utilizador WHERE
Utilizador.id_Utilizador = @id_Utilizador)
				BEGIN
					IF EXISTS (SELECT BloquearUtilizador.id_Utilizador FROM
BloquearUtilizador WHERE BloquearUtilizador.id_Utilizador = @id_Utilizador)
						BEGIN
							RAISERROR('Utilizador ja bloqueado', 16, 1)
						END
					ELSE
						BEGIN
							INSERT INTO
BloquearUtilizador(id_Admin,id_Utilizador,duracao,motivo)
							VALUES
(@id_Admin,@id_Utilizador,@duracao,@motivo)
						END
				END
			ELSE
				BEGIN
					RAISERROR('Utilizador inexistente', 16, 1)
				END
		END
	ELSE
		BEGIN
			RAISERROR('Admin inexistente', 16, 1)
		END

	if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
	GOTO ERRO
	COMMIT
return 1
	ERRO:
		ROLLBACK
		RETURN -1
GO
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
----------------------------------------
CREATE PROCEDURE Ver_Clientes
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

	IF EXISTS (SELECT Client.id_Client FROM Client)
		BEGIN
			SELECT * FROM Client
		END
	ELSE
		BEGIN
			RAISERROR('Nao existem clientes', 16, 1)
		END

	if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
	GOTO ERRO
	COMMIT
return 1
	ERRO:
```

```
		ROLLBACK
		RETURN -1
GO
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------
CREATE PROCEDURE Ver_Restaurantes_Admin
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

	IF EXISTS (SELECT Restaurante.id_Restaurante FROM Restaurante)
		BEGIN
			SELECT * FROM Restaurante
		END
	ELSE
		BEGIN
			RAISERROR('Nao existem restaurantes', 16, 1)
		END

	if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
	GOTO ERRO
	COMMIT
return 1
	ERRO:
		ROLLBACK
		RETURN -1
GO
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------
CREATE PROCEDURE Ver_Clientes_Bloqueados
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

	IF EXISTS (SELECT BloquearUtilizador.id_Utilizador FROM BloquearUtilizador)
		BEGIN
			SELECT C.nome AS Nome, B.motivo AS Motivo, B.duracao AS Duração
			FROM Client C, BloquearUtilizador B
			WHERE B.id_Utilizador=C.id_Client
		END
	ELSE
		BEGIN
			RAISERROR('Nao existem clientes bloqueados', 16, 1)
		END

	if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
	GOTO ERRO
	COMMIT
return 1
	ERRO:
		ROLLBACK
		RETURN -1
GO
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------
CREATE PROCEDURE Ver_Restaurantes_Autorizados
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

	IF EXISTS (SELECT Autorizar.id_Restaurante FROM Autorizar)
```

```
		BEGIN
			SELECT R.nome AS Nome
			FROM Autorizar A, Restaurante R
			WHERE A.id_Restaurante=R.id_Restaurante and R.validado=1
		END
	ELSE
		BEGIN
			RAISERROR('Nao existem restaurantes autorizados', 16, 1)
		END

	if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
	GOTO ERRO
	COMMIT
return 1
	ERRO:
		ROLLBACK
		RETURN -1
GO
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
----------------------------------------
CREATE PROCEDURE Ver_Restaurantes_Nao_Autorizados
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

	IF EXISTS (SELECT Restaurante.id_Restaurante FROM Restaurante WHERE
Restaurante.validado = 0)
		BEGIN
			SELECT R.nome as Nome
			FROM  Restaurante R
			WHERE R.validado=0
		END
	ELSE
		BEGIN
			RAISERROR('Nao existem restaurantes nao autorizados', 16, 1)
		END

    if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
    GOTO ERRO
    COMMIT
return 1
    ERRO:
        ROLLBACK
        RETURN -1
GO
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
----------------------------------------
--Procedures Necessários aos Restaurantes-----------------------------------------------------
-----------------------------------------------------------------------------------------------
----------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
----------------------------------------
CREATE PROCEDURE Criar_Prato
	@id_R INTEGER, @id_P INTEGER, @preco MONEY, @dia_semana VARCHAR(30), @apagado BIT
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

	IF EXISTS (SELECT CriarPrato.id_Prato FROM CriarPrato WHERE CriarPrato.id_Prato =
@id_P)
		BEGIN
```

```
                        RAISERROR('Prato ja existente', 16, 1)
                END
        ELSE
                BEGIN
                        INSERT INTO CriarPrato
                        VALUES (@id_R, @id_P, @preco, @dia_semana, '0')
                END

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

ERRO:
        ROLLBACK
        RETURN -1
GO
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
----------------------------------------
CREATE PROCEDURE Criar_Prato_Do_Dia
        @id_P integer, @nome varchar(25), @descricao varchar(200),@tipo_prato
varchar(20),@foto varchar(300)
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

        IF EXISTS (SELECT CriarPrato.id_Prato FROM CriarPrato WHERE CriarPrato.id_Prato =
@id_P)
                BEGIN
                        RAISERROR('Prato ja existente', 16, 1)
                END
        ELSE
                BEGIN
                        insert into PratoDoDia(id_Prato, nome,descricao, tipo_prato, foto)
                        values (@id_P, @nome, @descricao, @tipo_prato, @foto)
                END

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

ERRO:
        ROLLBACK
        RETURN -1
GO
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
----------------------------------------
CREATE PROCEDURE Alterar_Pratos_Restaurante
        @id_Restaurante INTEGER, @id_Prato INTEGER, @nome VARCHAR(50), @preco
MONEY,@dia_semana_prato VARCHAR(50), @descricao VARCHAR(250), @tipo_prato VARCHAR(50)
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

        IF EXISTS (SELECT CriarPrato.id_Prato FROM CriarPrato WHERE CriarPrato.id_Prato =
@id_Prato)
                BEGIN
                        IF EXISTS (SELECT CriarPrato.id_Restaurante FROM CriarPrato WHERE
CriarPrato.id_Restaurante = @id_Restaurante)
                                BEGIN
                                        UPDATE CriarPrato
```

```
                                SET preco=@preco,dia_semana_prato=@dia_semana_prato
                                WHERE id_Prato=@id_Prato
                                UPDATE PratoDoDia
                                SET nome=@nome, descricao=@descricao,
tipo_prato=@tipo_prato
                                WHERE id_Prato=@id_Prato
                        END
                ELSE
                        Begin
                                RAISERROR('Restaurante inexistente', 16, 1)
                        END
        END
    ELSE
        BEGIN
                RAISERROR('Prato inexistente', 16, 1)
        END

    if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
    GOTO ERRO
    COMMIT
return 1
    ERRO:
        ROLLBACK
        RETURN -1
GO
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
----------------------------------------
CREATE PROCEDURE Eliminar_Pratos_Restaurante
    @id_Restaurante INTEGER, @id_Prato INTEGER
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

    IF EXISTS (SELECT CriarPrato.id_Prato FROM CriarPrato WHERE CriarPrato.id_Prato =
@id_Prato)
        BEGIN
                IF EXISTS (SELECT Restaurante.id_Restaurante FROM Restaurante WHERE
Restaurante.id_Restaurante = @id_Restaurante)
                        BEGIN
                                DELETE FROM AdicionarPratoFav
                                WHERE(id_Prato=@id_Prato)
                                DELETE FROM CriarPrato
                                WHERE(id_Restaurante=@id_Restaurante and
id_Prato=@id_Prato)
                                DELETE FROM PratoDoDia
                                WHERE(id_Prato=@id_Prato)
                        END
                ELSE
                        BEGIN
                                RAISERROR('Restaurante inexistente', 16, 1)
                        END
        END
    ELSE
        BEGIN
                RAISERROR('Prato inexistente', 16, 1)
        END

    if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
    GOTO ERRO
    COMMIT
return 1
    ERRO:
        ROLLBACK
```

```
        RETURN -1
GO
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------
CREATE PROCEDURE Eliminar_Pratos_Do_Dia_Restaurante
    @id_Restaurante INTEGER, @id_Prato INTEGER, @dia_da_semana VARCHAR(50)
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

       IF EXISTS (SELECT PratoDoDia.id_Prato FROM PratoDoDia WHERE PratoDoDia.id_Prato =
@id_Prato)
              BEGIN
                     IF EXISTS (SELECT Restaurante.id_Restaurante FROM Restaurante WHERE
Restaurante.id_Restaurante = @id_Restaurante)
                           BEGIN
                                  DELETE FROM AdicionarPratoFav
                                  WHERE(id_Prato=@id_Prato)
                                  DELETE FROM CriarPrato
                                  WHERE(id_Prato=@id_Prato and
dia_semana_prato=@dia_da_semana)
                                  DELETE FROM PratoDoDia
                                  WHERE(id_Prato=@id_Prato)
                           END
                     ELSE
                           BEGIN
                                  RAISERROR('Restaurante inexistente', 16, 1)
                           END
              END
       ELSE
              BEGIN
                     RAISERROR('Prato inexistente', 16, 1)
              END

    if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
    GOTO ERRO
    COMMIT
return 1
    ERRO:
        ROLLBACK
        RETURN -1
GO
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------
CREATE PROCEDURE Criar_Restaurante
       @id_Rest INTEGER, @nome VARCHAR(20), @morada VARCHAR(50), @gps VARCHAR(50), @telefone
NUMERIC(9), @dia_desc VARCHAR(50), @tt BIT, @tl BIT, @te BIT, @hora_aber CHAR(5),
@hora_fecho CHAR(5), @desc VARCHAR(250), @foto VARCHAR(300)
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

       IF EXISTS (SELECT Utilizador.id_Utilizador FROM Utilizador WHERE
Utilizador.id_Utilizador = @id_Rest)
              BEGIN
                     INSERT INTO Restaurante
                     VALUES(@id_Rest, @nome, 0, @morada, @gps, @telefone, @dia_desc, @tt,
@tl, @te, @hora_aber, @hora_fecho, @desc, @foto)
              END
       ELSE
              BEGIN
                     RAISERROR('Utillizador invalido', 16, 1)
```

```
		END

	if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
	GOTO ERRO
	COMMIT
return 1
	ERRO:
		ROLLBACK
		RETURN -1
GO
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
----------------------------------------
CREATE PROCEDURE Alterar_Restaurante
	 @id_Rest INTEGER, @nome VARCHAR(50), @morada VARCHAR(50), @gps VARCHAR(50),
@telefone NUMERIC(9), @dia_desc VARCHAR(50), @tt BIT, @tl BIT, @te BIT, @hora_aber CHAR(5),
@hora_fecho CHAR(5), @desc VARCHAR(250), @foto VARCHAR(300)
AS
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

	IF EXISTS (SELECT Restaurante.id_Restaurante FROM Restaurante WHERE
Restaurante.id_Restaurante = @id_Rest)
		BEGIN
			UPDATE Restaurante
			SET  nome = @nome, morada = @morada, gps = @gps, telefone = @telefone,
dia_descanso = @dia_desc, tipo_takeaway = @tt, tipo_local = @tl, tipo_entrega = @te,
hora_de_abertura = @hora_aber, hora_de_fecho = @hora_fecho, descricao = @desc, foto = @foto
			WHERE id_Restaurante = @id_Rest
		END
	ELSE
		BEGIN
			RAISERROR('Restaurante inexistente', 16, 1)
		END

	if(@@ERROR <> 0) OR (@@ROWCOUNT = 0)
	GOTO ERRO
	COMMIT
return 1
	ERRO:
		ROLLBACK
		RETURN -1
GO
```