UNIVERSIDADE DE TRÁS-OS-MONTES E ALTO DOURO

# TRABALHO EXPERIMENTAL 2

## 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 segunda etapa do mesmo, incide no modelo de distribuição da base de dados desenvolvida, na politica de acesso aos mesmos, na resolução de problemas de concorrência no sistema de base de dados distribuída e na análise de otimização de questões distribuídas. Estas incidências foram aplicadas a uma Base de Dados previamente elaborada, em outra unidade curricular, Laboratório de Aplicações Web e de Base de Dados.

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

Segue em anexo (Anexo A – SQL Local e Anexo B – SQL Remoto) 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]

As bases de dados distribuídas existem para se conseguir regular quantidades enormes de pedidos a que um sistema de gestão de bases de dados tem que dar resposta. Uma base de dados distribuída, tal como é sugerido pelo nome, é um sistema de bases de dados cujos dados se encontram fisicamente dispersos por várias máquinas, ligadas por meios de comunicação, mas integrados logicamente. Isto é feito para se conseguir um equilíbrio dos recursos e poder computacional dos diversos servidores, levando, portanto, a uma maior complexidade do sistema.

Podemos dividir estes sistemas de bases de dados distribuídas em dois modelos:

- ***Homogéneos***: todos os nodos usam o mesmo SGBD, lembrando um único sistema de bases de dados, mas em que, em vez de todos os dados estarem armazenados num único repositório, os dados estão armazenados por vários repositórios ligados por meios de comunicação.

- ***Heterogéneos***: existência de SGBDs diferentes nos vários nodos. As diferenças entre os SGBDs presentes podem colocar-se a vários níveis, desde SGBDs diferentes baseados no mesmo modelo de dados (MS SQL Server, Oracle, MySQL, DB2, Informix, Sybase, etc.), até SGBDs baseados em modelos de dados diferentes (hierárquico, rede, relacional, orientado a objetos, etc.).

Podem ser escolhidos 2 métodos de divisão das bases de dados de forma a se conseguir alcançar uma base de dados distribuída. Estes tratam do esquema lógico distribuído pelos vários nodos que a base de dados irá ter.

- ***Top-down***: Correspondente à divisão de uma base de dados preexistente ou, mais genericamente, de um esquema de base de dados predefinido em várias partes a armazenar em diferentes nodos. Normalmente, o resultado deste processo será um ambiente distribuído homogéneo de bases de dados.
- ***Bottom-up***: Correspondente, não à desagregação, mas sim à integração de várias bases de dados preexistentes numa base de dados global, distribuída por várias máquinas. Dada a mais que provável heterogeneidade das várias bases de dados em presença, esta será a abordagem que mais dificuldades oferece.

# 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.

Relativamente ao relatório do Trabalho Experimental 1, os 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.

Relativamente ao relatório do Trabalho Experimental 2, os objetivos reúnem-se em:

- Relatório detalhado da execução do trabalho;
- Modelo de distribuição da base de dados desenvolvida;
- Políticas de segurança e acesso a dados distribuídos;
- Resolução para os problemas de concorrência no sistema de bases de dados distribuídas;
- Análise de otimização de questões distribuídas.

# 4. Desenvolvimento

## 4.1. Diagrama de Base de Dados Local

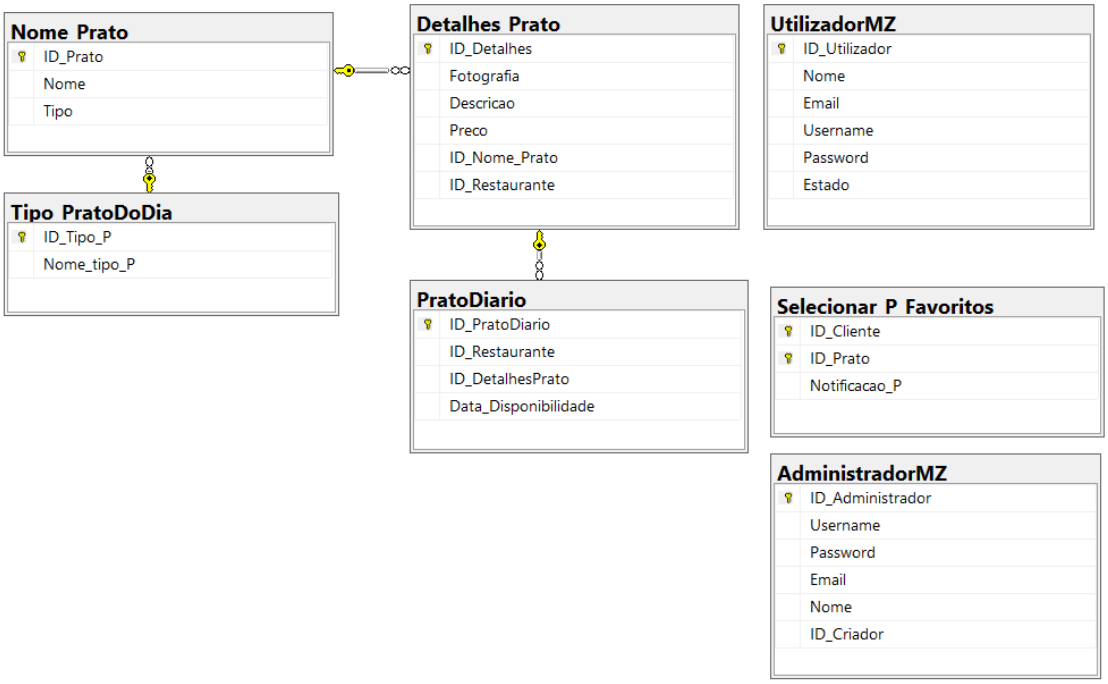


*Figura 1 - Diagrama de Base de Dados Local*

## 4.2.Diagrama de Base de Dados Remoto

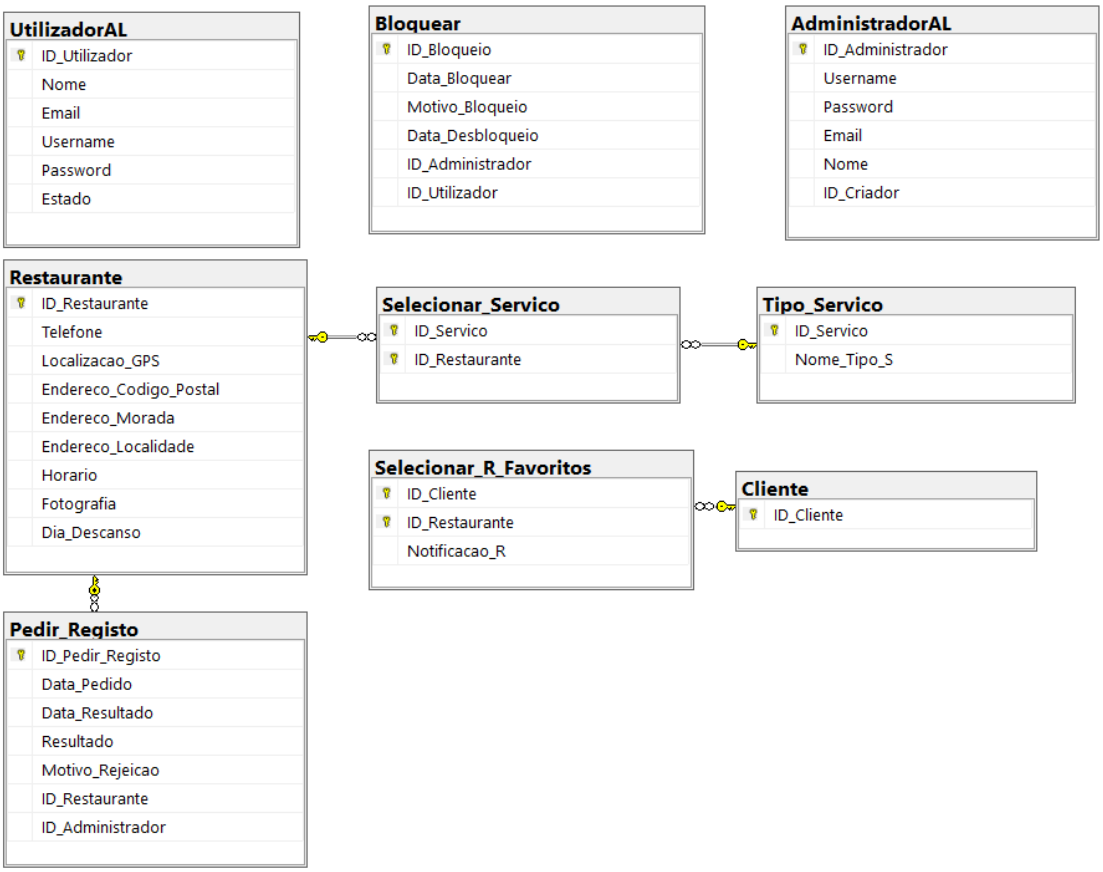


*Figura 2 - Diagrama de Base de Dados Remoto*

## 4.3. 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, 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.

No que toca à distribuição da base de dados, optamos por dividir as tabelas Utilizador e Administrador, pela letra de A-L e da M-Z. Isso foi atingido após a escolha da abordagem de ***Top-Down***, o que resultou num sistema ***Homogéneo***.

O que influenciou a escolha da nossa abordagem foi a obtenção de um sistema com desempenho elevado, ou seja, procuramos distribuir as tabelas pelos vários nodos da rede de maneira a obtermos menor custo de exploração do sistema global.

## 4.4.Código SQL

```
--- CRIAR O LINKED SERVER
exec sp_addlinkedserver
@server='ServerRemoto',
@srvproduct='SQLServer Native Client OLEDB Provider',
@provider='SQLBCLI',
@datasrc='192.168.xxx.xxx'

--- ACESSO AO SERVIDOR REMOTO
exec sp_addlinkedsrvlogin
@rmtsrvname='ServerRemoto',
@useself='false',
@locallogin='sa',
@rmtuser='sa',
@rmtpassword='12345'
```

*Figura 3 - Código referente à Criação do Linked Server*

Para que a conexão seja possível, no servidor local foi necessário implementar uma ligação com o servidor remoto. Assim, os parâmetros usados foram: [@server], que é o nome dado ao server remoto; [@srvproduct], que é o nome da OLE DB data source conectado ao ***linked server***; [@provider], referente ao identificador especificado no [@srvproduct] e [@datasrc], onde se declara o IP onde o servidor remoto se localiza. Desta forma é criado o ***linked server***, e consequentemente, a necessidade da criação de acesso ao mesmo. Para esse efeito, é executado 'sp_addlinkedsrvlogin', que inclui os parâmetros [@rmtsrvname], nome do ***linked server***, [@useself], onde é determinado a conexão, ou seja, caso esteja a ***false***, é necessário utilizar as credenciais do servidor remoto, que estão definidas em [@locallogin], [@rmtuser] e [@rmtpassword]. O código apresentado é referente ao server local, sendo que no server remoto o mesmo é implementado, apenas com a mudança do parâmetro de [@server], [@datasrc] e [@rmtsrvname].

```
CREATE ROLE VisitanteRole
EXECUTE sp_addrolemember 'VisitanteRole', 'VISITANTE'
CREATE ROLE AdministradorRole
EXECUTE sp_addrolemember 'AdministradorRole', 'ADMINISTRADOR'
CREATE ROLE ClienteRole
EXECUTE sp_addrolemember 'ClienteRole', 'CLIENTE'
CREATE ROLE RestauranteRole
EXECUTE sp_addrolemember 'RestauranteRole', 'RESTAURANTE'
```

*Figura 4 - Código referente à Criação de Roles*

Para que a distribuição seja o mais simples possível, foi necessário a implementação de Roles, associados aos Logins previamente definidos.

```
CREATE VIEW Utilizador
AS
	SELECT * FROM UtilizadorAL
	UNION ALL
	SELECT * FROM ServerRemoto.TABD_RISTORANTIS_REMOTO.dbo.UtilizadorMZ

GO
```

*Figura 5 - Código referente à Views 1*

```
CREATE VIEW Nome_Prato
AS

	SELECT * FROM ServerRemoto.TABD_RISTORANTIS_REMOTO.dbo.Nome_Prato

GO
```

*Figura 6 - Código referente à Views 2*

Para simplificar e para que o acesso seja possível, é necessário dentro da Base de Dados aceder ao ***linked server*** através de ***Views*** que interligam as duas Base de Dados. No código apresentado, este refere-se apenas às Views criadas no Server Local, sendo que o mesmo tem que ser aplicado ao Server Remoto, alterando o caminho de acesso para o nome do server, 'ServerLocal' e o nome da Base de Dados para 'TABD_RISTORANTIS_LOCAL'.

# 5. Conclusão

Após a finalização da segunda 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 2, 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 distribuída. Assim, na Base de Dados utilizada foram criados Logins, Roles, Users e Procedures, bem como todo o código necessário para a ligação entre os dois servidores.

Para a resolução de todos os objetivos pretendidos nesta segunda 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 2 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
- sp_addrolemember (Transact-SQL), Microsoft. https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/sp-addrolemember-transact-sql?view=sql-server-ver15
- sp_addlinkedsrvlogin (Transact-SQL), Microsoft. https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/sp-addlinkedsrvlogin-transact-sql?view=sql-server-ver15
- sp_addlinkedserver (Transact-SQL), Microsoft. https://docs.microsoft.com/pt-pt/sql/relational-databases/system-stored-procedures/sp-addlinkedserver-transact-sql?view=sql-server-ver15

# 7. Anexo A – SQL Local

```
USE MASTER
GO

CREATE DATABASE TABD_RISTORANTIS_LOCAL
GO

USE TABD_RISTORANTIS_LOCAL
GO

--Ana Dias al69691
--Diana Alves al68557
--Diana Ferreira al68938
--Rui Vaz al68565

--- CRIAR O LINKED SERVER
exec sp_addlinkedserver
@server='ServerRemoto',
@srvproduct='SQLServer Native Client OLEDB Provider',
@provider='SQLBCLI',
@datasrc='192.168.xxx.xxx'

--- ACESSO AO SERVIDOR REMOTO
exec sp_addlinkedsrvlogin
@rmtsrvname='ServerRemoto',
@useself='false',
@locallogin='sa',
@rmtuser='sa',
@rmtpassword='12345'


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

	Check(Nome<='L'),

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

CREATE TABLE UtilizadorAL(
	ID_Utilizador INTEGER IDENTITY(1,1) NOT NULL,
	Nome			NVARCHAR(150) NOT NULL,
	Email			NVARCHAR(200) UNIQUE NOT NULL,
	Username		NVARCHAR(10) UNIQUE NOT NULL,
	Password		NVARCHAR(10) NOT NULL,
	Estado			NVARCHAR(10) DEFAULT 'Registado' NOT NULL,
	Check(Nome<='L'),

	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

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 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)
)



USE TABD_RISTORANTIS_LOCAL
GO

CREATE VIEW Utilizador
AS
SELECT * FROM UtilizadorAL
UNION ALL
SELECT * FROM ServerRemoto.TABD_RISTORANTIS_REMOTO.dbo.UtilizadorMZ
GO

CREATE VIEW Administrador
AS
SELECT * FROM AdministradorAL
UNION ALL
SELECT * FROM ServerRemoto.TABD_RISTORANTIS_REMOTO.dbo.AdministradorMZ
GO

CREATE VIEW Tipo_PratoDoDia
AS
SELECT * FROM ServerRemoto.TABD_RISTORANTIS_REMOTO.dbo.Tipo_PratoDoDia
GO

CREATE VIEW Nome_Prato
AS
SELECT * FROM ServerRemoto.TABD_RISTORANTIS_REMOTO.dbo.Nome_Prato
GO

CREATE VIEW Detalhes_Prato
AS
SELECT * FROM ServerRemoto.TABD_RISTORANTIS_REMOTO.dbo.Detalhes_Prato
GO

CREATE VIEW PratoDiario
AS
SELECT * FROM ServerRemoto.TABD_RISTORANTIS_REMOTO.dbo.PratoDiario
GO

CREATE VIEW Selecionar_P_Favoritos
AS
SELECT * FROM ServerRemoto.TABD_RISTORANTIS_REMOTO.dbo.Selecionar_P_Favoritos
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'
CREATE LOGIN VISITANTE WITH PASSWORD= '12345'

-- CRIAÇÃO DE USERS
USE TABD_RISTORANTIS_LOCAL
GO

exec sp_addlinkedsrvlogin
@rmtsrvname='ServerRemoto',
@useself='true'

CREATE USER ADMINISTRADOR FOR LOGIN ADMINISTRADOR
CREATE USER CLIENTE FOR LOGIN CLIENTE
CREATE USER RESTAURANTE FOR LOGIN RESTAURANTE
CREATE USER VISITANTE FOR LOGIN VISITANTE

--CRIAÇÃO DE ROLES
CREATE ROLE VisitanteRole
EXECUTE sp_addrolemember 'VisitanteRole', 'VISITANTE'
CREATE ROLE AdministradorRole
EXECUTE sp_addrolemember 'AdministradorRole', 'ADMINISTRADOR'
CREATE ROLE ClienteRole
EXECUTE sp_addrolemember 'ClienteRole', 'CLIENTE'
CREATE ROLE RestauranteRole
EXECUTE sp_addrolemember 'RestauranteRole', 'RESTAURANTE'


--- ATRIBUIÇÃO DE PERMISSÕES NAS TABELAS
--- PERMISSÕES DOS VISITANTE
USE TABD_RISTORANTIS_LOCAL
GO

GRANT SELECT, INSERT ON Restaurante TO VisitanteRole
GRANT SELECT ON UtilizadorAL(ID_Utilizador, Nome, Email, Estado) TO VisitanteRole
GRANT SELECT, INSERT ON Selecionar_Servico TO VisitanteRole
GRANT SELECT ON Tipo_Servico TO VisitanteRole

GRANT INSERT ON UtilizadorAL TO VisitanteRole
GRANT INSERT ON Cliente TO VisitanteRole
GRANT INSERT ON Pedir_Registo TO VisitanteRole

--- PERMISSÕES DOS CLIENTE
USE TABD_RISTORANTIS_LOCAL
GO

GRANT SELECT ON Cliente TO ClienteRole
GRANT SELECT ON Restaurante TO ClienteRole
GRANT SELECT ON Selecionar_Servico TO ClienteRole
GRANT SELECT ON Tipo_Servico TO ClienteRole
GRANT SELECT ON UtilizadorAL TO ClienteRole
GRANT UPDATE ON UtilizadorAL(ID_Utilizador, Nome, Email, Username, Password) TO
ClienteRole
GRANT SELECT ON Bloquear TO ClienteRole
GRANT SELECT, INSERT, UPDATE, DELETE ON Selecionar_R_Favoritos TO ClienteRole

--- PERMISSÕES DOS RESTAURANTE
USE TABD_RISTORANTIS_LOCAL
GO
```

```
GRANT SELECT ON UtilizadorAL TO RestauranteRole
GRANT UPDATE ON UtilizadorAL(Nome, Email, Username, Password) TO RestauranteRole
GRANT SELECT, UPDATE ON Restaurante TO RestauranteRole
GRANT SELECT, INSERT, DELETE ON Selecionar_Servico TO RestauranteRole
GRANT SELECT ON Tipo_Servico TO RestauranteRole
GRANT SELECT ON Pedir_Registo TO RestauranteRole
GRANT SELECT ON Bloquear TO RESTAURANTE

--- PERMISSÕES DOS ADMINISTRADOR
USE TABD_RISTORANTIS_LOCAL
GO

GRANT SELECT, INSERT, UPDATE, DELETE ON AdministradorAL TO AdministradorRole
GRANT SELECT, UPDATE ON Pedir_Registo TO AdministradorRole
GRANT SELECT ON Restaurante TO AdministradorRole
GRANT SELECT ON Selecionar_Servico TO AdministradorRole
GRANT SELECT ON Tipo_Servico TO AdministradorRole
GRANT SELECT ON UtilizadorAL TO AdministradorRole
GRANT SELECT, INSERT, UPDATE ON Bloquear TO AdministradorRole
GRANT UPDATE ON UtilizadorAL(Estado) TO AdministradorRole
GRANT SELECT ON Cliente TO AdministradorRole


--PROCEDURES
--VISITANTE
GRANT EXECUTE ON Criar_Utilizador TO VisitanteRole
GRANT EXECUTE ON Criar_Cliente TO VisitanteRole
GRANT EXECUTE ON Criar_Restaurante TO VisitanteRole

--CLIENTE
GRANT EXECUTE ON Alterar_Utilizador TO ClienteRole
GRANT EXECUTE ON Selecionar_R_Favorito TO ClienteRole
GRANT EXECUTE ON Eliminar_Selecionar_R_Favoritos TO ClienteRole
GRANT EXECUTE ON Selecionar_P_Favorito TO ClienteRole
GRANT EXECUTE ON Eliminar_Selecionar_P_Favoritos TO ClienteRole

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

--RESTAURANTE
GRANT EXECUTE ON Alterar_Utilizador TO RestauranteRole
GRANT EXECUTE ON Alterar_Restaurante TO RestauranteRole
GRANT EXECUTE ON Registar_Novo_Prato TO RestauranteRole
GRANT EXECUTE ON Criar_Detalhes_PratoDia TO RestauranteRole
GRANT EXECUTE ON Alterar_Detalhes_PratoDia TO RestauranteRole
GRANT EXECUTE ON Apagar_Detalhes_PratoDia TO RestauranteRole
GRANT EXECUTE ON Criar_PratoDiario TO RestauranteRole
GRANT EXECUTE ON Alterar_PratoDiario TO RestauranteRole
GRANT EXECUTE ON Apagar_PratoDiario TO RestauranteRole
```

```
--PROCEDURES DOS VISITANTE
USE TABD_RISTORANTIS_LOCAL
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 DISTRIBUTED TRANSACTION Novo_Utilizador
	IF ( (EXISTS (SELECT * FROM Administrador A WHERE (A.Username=@username)))
OR (EXISTS (SELECT * FROM Administrador A WHERE (A.Email=@email))) )
		GOTO ERRO
	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 DISTRIBUTED TRANSACTION
		IF ((SELECT U.Estado FROM Utilizador U WHERE
(U.ID_Utilizador=@id_utilizador)) = 'Registado')
		BEGIN
			INSERT INTO Cliente(ID_Cliente)
			VALUES (@id_utilizador)
			IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
				GOTO ERRO

			UPDATE Utilizador
			SET Estado = 'Ativo'
			WHERE ID_Utilizador=@id_utilizador
			IF (@@ERROR <> 0) OR (@@ROWCOUNT = 0)
				GOTO ERRO
		END

COMMIT TRANSACTION
RETURN 1

ERRO:
	ROLLBACK TRANSACTION
	RETURN -1
GO

CREATE TYPE ServicoType AS TABLE(ID INTEGER)
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 DISTRIBUTED TRANSACTION Novo_Restaurante
	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)

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

			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
				SET @nr_servicos = @nr_servicos - 1
			END

			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

    COMMIT TRANSACTION Novo_Restaurante
    RETURN 1

ERRO:
    ROLLBACK TRANSACTION Novo_Restaurante
    RETURN -1

GO

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

--PROCEDURES DOS CLIENTE
USE TABD_RISTORANTIS_LOCAL
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 DISTRIBUTED TRANSACTION
    IF (EXISTS (SELECT * FROM Administrador A WHERE (A.Username=@username)) OR
EXISTS (SELECT * FROM Administrador A WHERE (A.Email=@email)) )
        GOTO ERRO
    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 DISTRIBUTED 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)
	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 DISTRIBUTED 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)

	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 DISTRIBUTED 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)
	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 DISTRIBUTED 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)

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

ERRO:
	ROLLBACK
	RETURN -1
GO

--PROCEDURES DOS ADMINISTRADOR
USE TABD_RISTORANTIS_LOCAL
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 DISTRIBUTED TRANSACTION
	IF (NOT EXISTS (SELECT * FROM Utilizador U WHERE (U.Username=@username))
OR (NOT EXISTS (SELECT * FROM Utilizador U WHERE (U.Email=@email))))
	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

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 DISTRIBUTED TRANSACTION
	IF ( NOT EXISTS (SELECT * FROM Utilizador u WHERE (u.Username=@username))
OR ( NOT EXISTS (SELECT * FROM Utilizador u WHERE (u.Email=@email))))
	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 DISTRIBUTED TRANSACTION

	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 DISTRIBUTED TRANSACTION
	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)))
		END

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

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 DISTRIBUTED TRANSACTION
	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

COMMIT
RETURN 1

ERRO:
	ROLLBACK
	RETURN -1
GO
```

```
--PROCEDURES DOS RESTAURANTE

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 DISTRIBUTED TRANSACTION
	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

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

	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
		SET @nr_servicos = @nr_servicos - 1
	END

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 DISTRIBUTED TRANSACTION
	IF (EXISTS (SELECT * FROM Tipo_PratoDoDia T WHERE ( T.ID_Tipo_P = @tipo)))
	BEGIN
		INSERT INTO Nome_Prato(Nome, Tipo)
		VALUES (@nome, @tipo)
	END
	ELSE
		GOTO ERRO

	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 DISTRIBUTED 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

	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 DISTRIBUTED TRANSACTION
	UPDATE Detalhes_Prato
	SET Fotografia=@fotografia, Descricao=@descricao, Preco=@preco,
ID_Nome_Prato=@id_Nome_Prato
	WHERE ID_Detalhes=@id_detalhes

	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 DISTRIBUTED TRANSACTION
	DELETE FROM Detalhes_Prato
	WHERE ID_Detalhes=@id_detalhes

	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 DISTRIBUTED 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
			GOTO ERRO
	END
	ELSE
		GOTO ERRO
```

```
	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 DISTRIBUTED TRANSACTION

	UPDATE PratoDiario
	SET Data_Disponibilidade=@data_Disponibilidade
	WHERE ID_PratoDiario=@id_pratodiario

	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 DISTRIBUTED TRANSACTION
	DELETE FROM PratoDiario
	WHERE ID_PratoDiario=@id_pratodiario

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

COMMIT
RETURN 1

ERRO:
	ROLLBACK
	RETURN -1

GO
```

# 8. Anexo B – SQL Remoto

```
CREATE DATABASE TABD_RISTORANTIS_REMOTO
GO

--Ana Dias al69691
--Diana Alves al68557
--Diana Ferreira al68938
--Rui Vaz al68565

USE TABD_RISTORANTIS_REMOTO
GO

exec sp_addlinkedserver
@server='ServerLocal',
@srvproduct='SQLServer Native Client OLEDB Provider',
@provider='SQLBCLI',
@datasrc='192.168.xxx.xxx'

--- ACESSO AO SERVIDOR REMOTO
exec sp_addlinkedsrvlogin
@rmtsrvname='ServerLocal',
@useself='false',
@locallogin='sa',
@rmtuser='sa',
@rmtpassword='12345'

---ENTIDADES---

CREATE TABLE AdministradorMZ(
	ID_Administrador	INTEGER IDENTITY (1,1) NOT NULL,
	Username			NVARCHAR(10) UNIQUE NOT NULL,
	Password			NVARCHAR(10) NOT NULL,
	Email				NVARCHAR(200) UNIQUE NOT NULL,
	Nome				NVARCHAR(150) NOT NULL,
	ID_Criador			INTEGER,
	Check(Nome>='M'),

	PRIMARY KEY (ID_Administrador)
)

CREATE TABLE UtilizadorMZ(
	ID_Utilizador INTEGER IDENTITY(1,1) NOT NULL,
	Nome			NVARCHAR(150) NOT NULL,
	Email			NVARCHAR(200) UNIQUE NOT NULL,
	Username		NVARCHAR(10) UNIQUE NOT NULL,
	Password		NVARCHAR(10) NOT NULL,
	Estado			NVARCHAR(10) DEFAULT 'Registado' NOT NULL,
	Check(Nome>='M'),

	PRIMARY KEY (ID_Utilizador)
)

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

```
GO

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

USE TABD_RISTORANTIS_REMOTO

GO


CREATE VIEW Utilizador
AS
SELECT * FROM UtilizadorMZ
UNION ALL
SELECT * FROM ServerLocal.TABD_RISTORANTIS_LOCAL.dbo.UtilizadorAL
GO

CREATE VIEW Administrador
AS
SELECT * FROM AdministradorMZ
UNION ALL
```

```
SELECT * FROM ServerLocal.TABD_RISTORANTIS_LOCAL.dbo.AdministradorAL
GO

CREATE VIEW Cliente
AS
SELECT * FROM ServerLocal.TABD_RISTORANTIS_LOCAL.dbo.Cliente
GO

CREATE VIEW Restaurante
AS
SELECT * FROM ServerLocal.TABD_RISTORANTIS_LOCAL.dbo.Restaurante
GO

CREATE VIEW Tipo_Servico
AS
SELECT * FROM ServerLocal.TABD_RISTORANTIS_LOCAL.dbo.Tipo_Servico
GO

CREATE VIEW Selecionar_Servico
AS
SELECT * FROM ServerLocal.TABD_RISTORANTIS_LOCAL.dbo.Selecionar_Servico
GO

CREATE VIEW Pedir_Registo
AS
SELECT * FROM ServerLocal.TABD_RISTORANTIS_LOCAL.dbo.Pedir_Registo
GO

CREATE VIEW Selecionar_R_Favoritos
AS
SELECT * FROM ServerLocal.TABD_RISTORANTIS_LOCAL.dbo.Selecionar_R_Favoritos
GO

CREATE VIEW Bloquear
AS
SELECT * FROM ServerLocal.TABD_RISTORANTIS_LOCAL.dbo.Bloquear
GO


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

exec sp_addlinkedsrvlogin
@rmtsrvname='ServerLocal',
@useself='true'

CREATE USER ADMINISTRADOR FOR LOGIN ADMINISTRADOR
CREATE USER CLIENTE FOR LOGIN CLIENTE
CREATE USER RESTAURANTE FOR LOGIN RESTAURANTE
CREATE USER VISITANTE FOR LOGIN VISITANTE

--CRIAÇÃO DE ROLES
CREATE ROLE VisitanteRole
EXECUTE sp_addrolemember 'VisitanteRole', 'VISITANTE'
CREATE ROLE AdministradorRole
EXECUTE sp_addrolemember 'AdministradorRole', 'ADMINISTRADOR'
CREATE ROLE ClienteRole
EXECUTE sp_addrolemember 'ClienteRole', 'CLIENTE'
CREATE ROLE RestauranteRole
EXECUTE sp_addrolemember 'RestauranteRole', 'RESTAURANTE'

USE TABD_RISTORANTIS_REMOTO
```

```
GO

GRANT SELECT ON UtilizadorMZ(ID_Utilizador, Nome, Email, Estado) TO VisitanteRole
GRANT SELECT ON Nome_Prato TO VisitanteRole
GRANT SELECT ON Tipo_PratoDoDia TO VisitanteRole
GRANT SELECT ON Detalhes_Prato TO VisitanteRole
GRANT SELECT ON PratoDiario TO VisitanteRole
GRANT INSERT ON UtilizadorMZ TO VisitanteRole


USE TABD_RISTORANTIS_REMOTO
GO

GRANT SELECT ON UtilizadorMZ TO ClienteRole
GRANT UPDATE ON UtilizadorMZ(ID_Utilizador, Nome, Email, Username, Password) TO
ClienteRole
GRANT SELECT ON Nome_Prato TO ClienteRole
GRANT SELECT ON Tipo_PratoDoDia TO ClienteRole
GRANT SELECT ON Detalhes_Prato TO ClienteRole
GRANT SELECT ON PratoDiario TO ClienteRole
GRANT SELECT, INSERT, UPDATE, DELETE ON Selecionar_P_Favoritos TO ClienteRole


--- PERMISSÕES DOS RESTAURANTES
USE TABD_RISTORANTIS_REMOTO
GO
GRANT SELECT ON UtilizadorMZ TO RestauranteRole
GRANT UPDATE ON UtilizadorMZ(Nome, Email, Username, Password) TO RestauranteRole
GRANT SELECT, INSERT ON Nome_Prato TO RestauranteRole
GRANT SELECT ON Tipo_PratoDoDia TO RestauranteRole
GRANT SELECT, INSERT, UPDATE, DELETE ON Detalhes_Prato TO RestauranteRole
GRANT SELECT, INSERT, UPDATE, DELETE ON PratoDiario TO RestauranteRole


--- PERMISSÕES DOS ADMINISTRADORES
USE TABD_RISTORANTIS_REMOTO
GO
GRANT SELECT, INSERT, UPDATE, DELETE ON AdministradorMZ TO AdministradorRole
GRANT SELECT ON UtilizadorMZ TO AdministradorRole
GRANT UPDATE ON UtilizadorMZ(Estado) TO AdministradorRole
```