P.PORTO
ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

Unidade Curricular de Base de Dados

Ano Letivo de 2021/2022

Michelin Star - Restaurant

Luis Sousa 8200398

Francisco Sousa 8200397

Rómulo Leite 8200593

Resumo

No seguimento do início do novo projeto da gerência da “Michelin Star” para o desenvolvimento de um sistema informático que consiga suportar as necessidades do restaurante, surge a necessidade de implementação de uma base de dados para suportar o funcionamento da aplicação que registe informação sobre a gestão e do tratamento dos processos do restaurante. Este relatório descreve as fases da criação e implementação dessa mesma base de dados, tais como o planeamento, identificação de requisitos, desenho conceptual, desenho lógico para o modelo relacional e a tradução desse modelo lógico global para o SGBD. Este projeto é relativo à componente prática da Unidade Curricular de Base de Dados

Índice

Índice

1. Introdução

1.1 Contextualização

Este projeto é relativo à componente pratica da UC de Base de Dados e tem como objetivo principal o desenvolvimento de uma base de dados para suportar as necessidades de um restaurante, com base nos conhecimentos adquiridos ao longo do semestre nesta UC.

1.2 Apresentação do Caso de Estudo

O restaurante “Michelin Star” procura o desenvolvimento de um sistema informático que permita aos proprietários a elaboração e impressão das ementas diárias, registar os pedidos dos clientes, a confeção dos pratos das ementas do dia seguinte, e por fim a gestão e pagamento das contas.

1.3 Motivação e Objetivos

O objetivo principal deste projeto é a implementação de uma base de dados para suportar o funcionamento da aplicação de gestão do restaurante, aplicando os conhecimentos adquiridos na UC. Esta base de dados deverá permitir as seguintes consultas:

- Qual é a ementa de hoje e quais os pratos que nele figuram;
- Quais os produtos que são necessários para cumprir a ementa de amanhã;
- Quais foram os pratos de carne servidos durante o período de tempo a designar;
- Em que dias do corrente mês é que foi servido o prato de peixe “P” juntamente com o de carne “C”.

1.4 Estrutura do Relatório

A estrutura do relatório é dividia em capítulos e subcapítulos, onde se descreve os passos para a realização do projeto, assim desta forma facilita a leitura/consulta do mesmo.

É apresentado o tema, contextualização e o caso de estudo deste projeto na introdução, de seguida é descrito os processos realizados para o desenvolvimento deste projeto e por fim, uma reflexão sobre o mesmo.

2. Requisitos Gerais

2.1 Requisitos Gerais

Para que seja possível a identificação dos requisitos gerais do problema, além da informação disponibilizada no enunciado, os autores falaram com pessoas da área, para uma melhor compreensão e contextualização deste trabalho.

Foram identificados os seguintes requisitos:

1. Os funcionários registam o pedido do cliente;
2. O cliente efetua um pedido ao funcionário;
3. Uma ementa é constituída por várias secções, tais como, entradas, sopas, pratos de carne e peixe, sobremesas, bebidas…;
4. Cada secção da ementa pode possuir um ou mais produtos.
5. O prato do dia é constituído por produtos que serão diferentes todos os dias, mas com o custo sempre igual.
6. Todos os dias irá existir uma lista de produtos a comprar para reabastecer o stock.
7. Esta lista será gerada automaticamente.
8. As contas são automaticamente calculas o que permite elaborara fatura.
9. Com o pagamento do cliente é emitido o respetivo recibo com a discriminação de todos os artigos.

3. Desenho Conceptual

O primeiro passo para o desenho de uma base de dados é definir o modelo conceptual de dados. Este projeto realizar-se-á ao longo deste capítulo dividido nas seguintes fases:

1. Identificação de tipo de entidades;
2. Relações entre entidades;
3. Atributos para entidades;
4. Documentação de atributos;
5. Atribuição de chaves primarias;
6. Documentação de chaves primarias (PK);
7. Revisão do modelo conceptual de dados com o utilizador (versão final).

3.1 Identificação de tipo de entidades

Tabela 1 - Identificação das entidades.

Entidade	Descrição	Ocorrência
Funcionário	Informação geral dos funcionários.	O funcionário regista pedidos.
Pedido	Informação geral dos clientes.	Os pedidos são efetuados por clientes e registados por funcionários. Um pedido emite uma fatura.
Ingrediente	Informação geral dos ingredientes. Os ingredientes são todos os itens necessários para criar uma receita	Um conjunto de ingredientes forma uma receita.
Produto	Informação geral dos produtos. Um produto tem um tipo, por exemplo: sopas, entradas, sobremesas, bebidas, pratos de carne e peixe.	Um produto tem um tipo e pode ter uma receita.
Ementa	Informação geral das Ementas.	As ementas são o conjunto de produtos com vários tipos, estes tipos formam secções dentro da ementa.

3.2 Relações entre Entidades

3.2.1 Diagrama E-R das relações

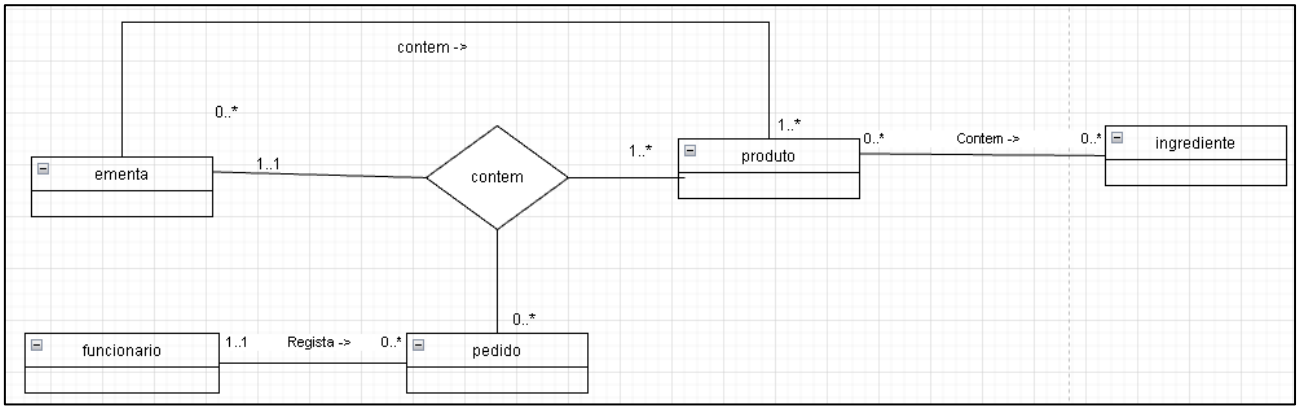

Figura 1 – Diagrama E-R das relações sem atributos.

3.2.2 Multiplicidade

Tabela 2 – Multiplicidade.

Entidade	Multiplicidade	Relação	Multiplicidade	Entidade
Funcionário	1…1	Regista	0…*	**Pedido**
Produto	0...*	Contem	0…*	**Ingrediente**
Ementa	0…1	Contem	1…*	**Produto**

Relacionamento Ternário:

Um pedido contem , 1 ou mais produtos de uma determinada ementa.

3.3 Atributos para entidades

Tabela 3 – Atributos para entidades.

Funcionário	funcionarioID, primeiroNome, ultimoNome, sexo, dataDeNascimento, dataDeModificacao, tipoFuncionario
Pedido	pedidoID, numMesa, data, estado, faturaID, clienteNIF, obs
Ingrediente	ingredienteID, nome, tipoUnidade , obs
Produto	produtoID, nome, tipo, preço, obs
Ementa	ementaID, dataEmenta, obs

3.4 Documentação de atributos

Tabela 4 – Documentação dos atributos.

Entidade	*Atributos*	*Descrições*	*Tipo de dados e tamanho*	*Nulls*	*Multi-Valued*
Funcionário	funcionarioID	Número do funcionário	Até 3 números inteiros	não	não
	pNome	Primeiro nome do funcionário	15 caracteres variáveis	não	não
	uNome	Último nome do funcionário	15 caracteres variáveis	não	não
	sexo	Género do funcionário	1 caracter	não	não
	estado	Estado do funcionário, ativo ou inativo	Caracter (‘Ativo’, ‘Inativo’)	não	não
	dataDeNascimento	Data de nascimento do funcionário	Tipo data	não	não
	tipoFuncionario	Tipo de funcionário da empresa	1 caracter	não	não
	dataDeModificacao	Data de modificação relativamente a alguma informação do funcionário	Tipo data	não	não
Pedido	pedidoID	Número do pedido	Até 3 números inteiros	não	não
	numMesa	Número da mesa	Até 2 números inteiros	não	não
	data	Data de quando foi efetuado o pedido	Tipo data	não	não
	faturaID	Número da fatura	Até 3 números inteiros Unique	sim	não

	estado	Estado do pagamento, pago ou não pago	número inteiro	não	não
	clienteNIF	NIF do cliente	9 dígitos numéricos	sim	não
	valorTotal	Valor Total do Pedido	Até 4 números inteiros e 2 casas decimais		
	obs	Conjunto de observações	250 caracteres variáveis	não	não
Ingrediente	ingredienteID	Número do ingrediente	Até 3 números inteiros	não	não
	nome	Nome do ingrediente	15 caracteres variáveis	não	não
	unit	Tipo de unidade do ingrediente	10 caracteres variáveis		
	obs	Descrição do ingrediente	250 caracteres variáveis	não	não
Produto	produtoID	Número do produto	Até 2 números inteiros	não	não
	nome	Nome do produto	15 caracteres variáveis	não	não
	tipo	Tipo do produto	2 caracteres variáveis	não	não
	preço	Preço do produto	Decimal (6,2)	não	não
	obs	Conjunto de observações	250 caracteres variáveis	não	não
Ementa	ementaID	Número da ementa	Até 2 números inteiros	não	não
	dataEmenta	Data da ementa	Tipo data	não	não
	obs	Conjunto de observações	250 caracteres variáveis	não	não

3.5 Atribuição das chaves primárias

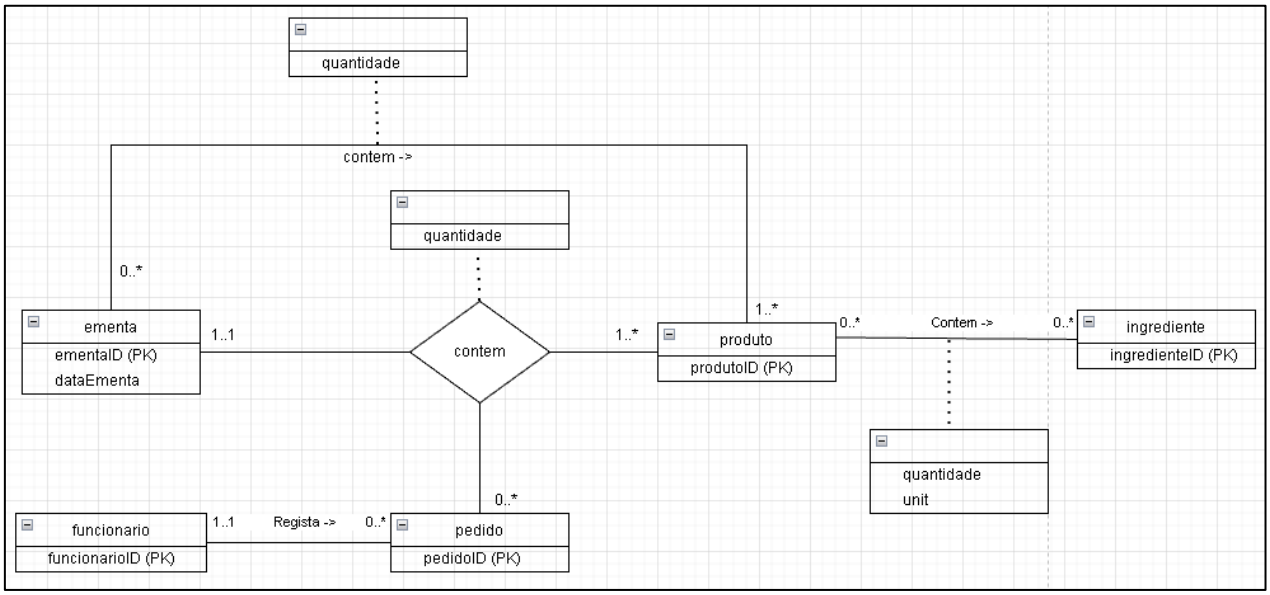

Figura 2 – Diagrama conceptual só com chaves primárias

3.6 Documentação das chaves primárias (PK)

Tabela 5 - Atribuição das chaves primárias

Entidade	**Chave Primária**	**Chaves Candidatas**
Funcionário	funcionarioID	
Pedido	pedidoID	faturaID
Ingrediente	ingredienteID	
Produto	produtoID	
Ementa	EmentaID	dataEmenta

3.7 Modelo conceptual (Versão final)

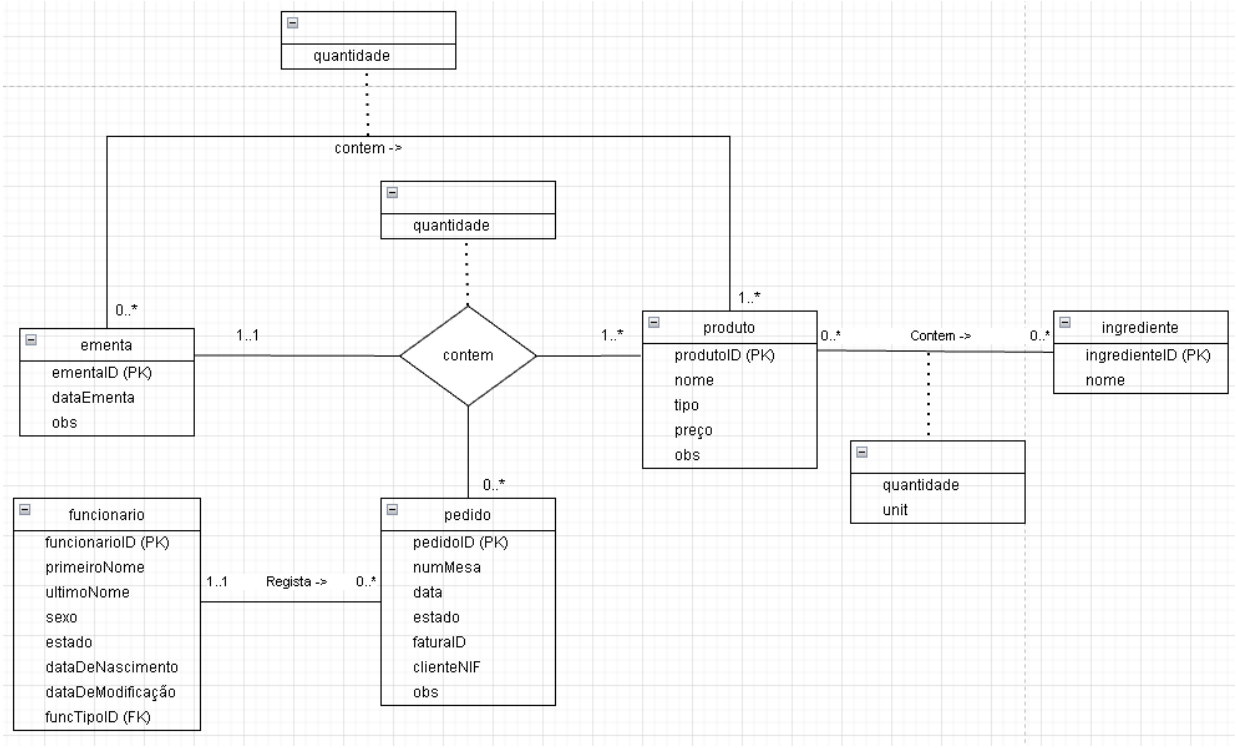

Figura 3 – Diagrama Conceptual ER

4. Desenho Lógico

Nesta fase o modelo conceptual vai ser traduzido para modelo lógico, que será visto pelos utilizadores de forma a verificar se este se encontra estruturalmente correto, as fases desta etapa são:

1. Derivação de relações entre entidades para o modelo de dados lógico e sua documentação;
2. Normalização;
3. Restrições de integridade;
4. Rever o modelo lógico de dados com o utilizador

4.1 Derivação de relações entre entidades para o modelo de dados lógico e sua documentação

4.1.1 Entidades Fortes

Pedido
Produto
Ementa

4.1.2 Entidades Fracas

Funcionário
Ingrediente

4.1.3 Relação de um para muitos (1 : *)

Funcionário 1: Pedido *

Entidade pai:
Funcionário { funcionarioID, primeiroNome, ultimoNome, sexo, dataDeNascimento, dataDeModificacao, tipoFuncionario }
Chave primaria: funcionarioID

Entidade filho:

Pedido { pedidoID, numMesa, data, estado, faturaID, clientNIF, obs, funcionarioID}

Chave primaria: pedidoID

Chave estrangeira: funcionarioID refere Funcionario.

Põe funcionarioID em pedido para modelar uma relação de 1 : *

4.1.5 Relação de um para um (* : *)

Ingrediente * : Produto*

Ingrediente { ingredienteID, nome, obs }

Chave primária: ingredienteID

Produto { produtoID, nome, tipo, preço, obs, ingredienteID}

Chave primária: produtoID

Nova entidade:

Receita { produtoID, ingredienteID, quantidade }

Chave primária: produtoID, ingredienteID

Chave estrangeira: produtoID refere o Produto

ingredienteID refere o Ingrediente

Ementa * : Produto *

Ementa { ementaID, dataEmenta, obs }

Chave primária: ementaID, dataEmenta

Produto { produtoID, nome, tipo, preço, obs, ingredienteID }

Chave primária: produtoID

Nova entidade:

EmentaProduto { ementaID, produtoID, quantidade }

Chave primária: ementaID, produtoID

Chave estrangeira: ementaID refere Ementa

produtoID refere Produto

4.1.6 Relação ternária

Ementa { ementaID, dataEmenta, obs }
Chave primária: ementaID, dataEmenta

Produto { produtoID, nome, tipo, preço, obs, ingredienteID }
Chave primária: produtoID

Nova entidade:
PedidoEmentaProduto { ementaID, produtoID, pedidoID ,quantidade }
Chave primária: ementaID, produtoID, pedidoID
Chave estrangeira: ementaID, produtoID refere EmentaProduto
pedidoID refere Pedido

4.2 Normalização

O processo de normalização surge como uma atividade associada ao Desenho Lógico que visa validar se as relações obtidas anteriormente são válidas.

Este processo de validação assenta no estudo das dependências funcionais e transitivas existentes entre atributos de uma relação.

4.2.1 Primeira Forma Normal (1FN)

A normalização de uma tabela na 1.ª Forma Normal (1FN) exige que a tabela tenha uma estrutura bidimensional correta, ou seja, cada linha deve corresponder a um só registo e cada coluna a um só campo.

O objetivo é eliminar redundância e introduzir dados apropriados nas colunas vazias das linhas que contêm grupos repetidos.

4.2.2 Segunda Forma Normal (2FN)

A 2.ª forma normal diz que a tabela tem de estar na 1FN e que cada atributo não chave tem de ser funcionalmente dependente da totalidade da chave primária e não apenas de uma parte dessa chave.

Assim depois de identificada a chave primária de uma tabela, pode dar-se um dos dois casos:

A chave primária é constituída por um só atributo (chave elementar). Neste caso, a tabela está seguramente na 2FN (nenhum atributo depende de uma parte da chave, visto que a chave não é composta por partes);

A chave primária é constituída por mais que um atributo (chave primária composta). Neste caso, se existe algum ou alguns atributos que dependem de uma parte da chave (ou seja, de algum atributo que constitui a chave), então a tabela não está na 2FN.

4.2.3 Terceira Forma Normal (3FN)

A 3.ª Forma Normal (3FN) diz que a tabela tem de estar na 2FN e que nenhum atributo não chave pode depender funcionalmente de algum outro atributo que não seja a chave primária.

É baseada no conceito de dependência transitiva.

Portanto, para normalizar uma tabela de acordo com a 3FN, deve-se analisar todos os atributos não chave e verificar se existem dependências transitivas sobre a chave primária, removê-las e colocá-las numa nova relação

4.2.4 Mockups

4.2.4.1 Ementa

Figura 4 – Mockup Ementa

Forma não normalizada.

Figura 5 – Forma não normalizada.

1º Forma Normal.

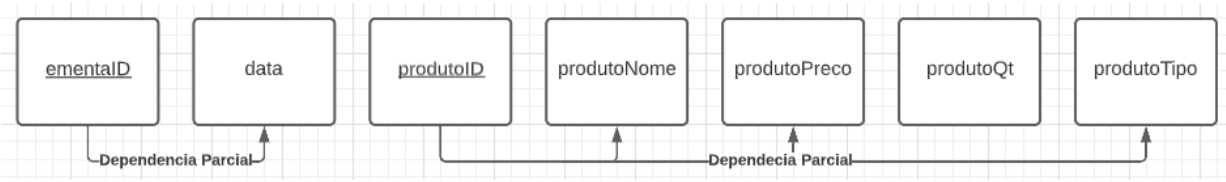

Figura 6 – Primeira forma normal.

Chegamos á conclusão que existe uma repetição do tipo de produto portanto criou-se uma nova relação.

ProdutoTipo { tipoID, tipoNome }
Chave primária: tipoID

2º Forma Normal.

Tabelas:

EmentaProduto:

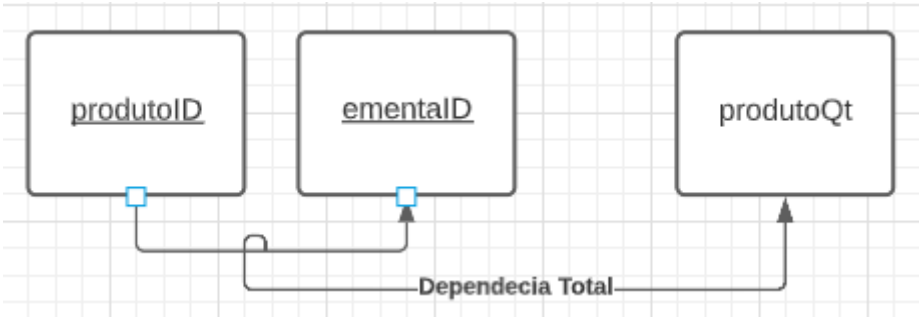

Figura 7 – Tabela EmentaProduto.

Produto:

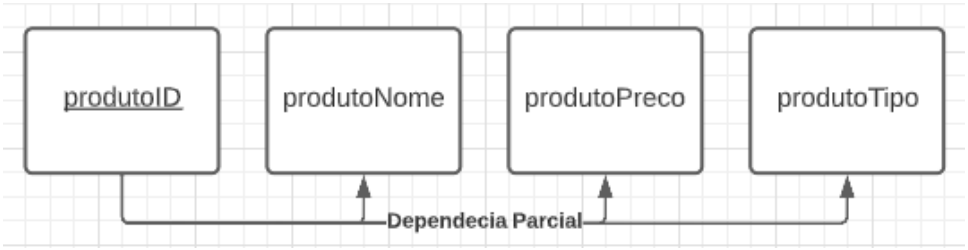

Figura 7 – Tabela Produto.

Ementa:

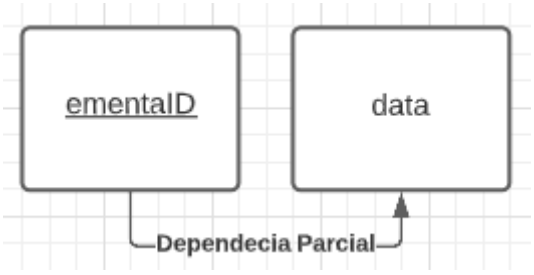

Figura 7 – Tabela Ementa.

3º Forma Normal.

A relação está na 2ª Forma Normal, mas não tem dependências transitivas (não existirem atributos descritores a dependerem funcionalmente de outros atributos descritores (não chaves)), ou seja, assume-se que se encontra na 3ª Forma Normal

Conclusão deste mockup.

Tabelas:

Ementa { ementaID, data }
Chave primária: ementaID.

Produto { produtoID, produtoNome, produtoPreco, tipoID }
Chave primária: produtoID.
Chave estrangeira: tipoID refere o ProdutoTipo

ProdutoTipo { tipoID, tipoNome }
Chave primária: tipoID.

EmentaProduto { ementaID, produtoID, quantidade }
Chave primária: produtoID, ementaID
Chave estrangeira: produtoID refere o Produto
ementaID refere a Ementa

4.2.4.2 Fatura simplificada e pedido

MICHELIN STAR		
Nº Funcionario:		
Nº Pedido:		
Nº Mesa:		
Data:		
ProdutoID	Qtd	preço

MICHELIN STAR		
Nº Fatura:		
NIF:		
Nº Pedido:		
Nº Mesa:		
Data:		
ProdutoID	Qtd	preço
Valor Total:		

Figura 5 e 6 – Mockup pedido e fatura, respetivamente.

Forma não normalizada.

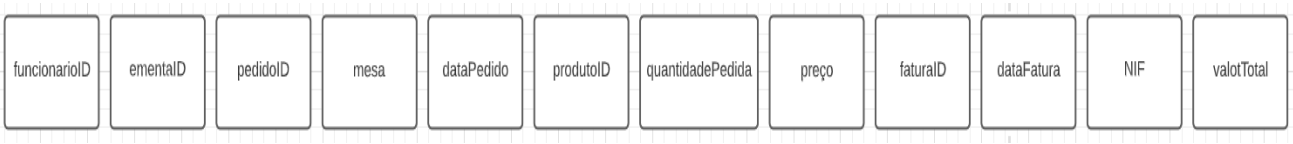

Figura 7 – Forma não normalizada.

1º Forma Normal.

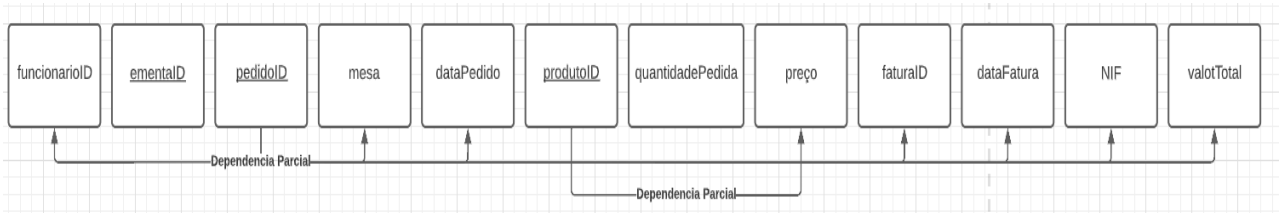

Figura 8 – Primeira forma normal.

2º Forma Normal.

Tabelas:

PedidoEmentaProduto:

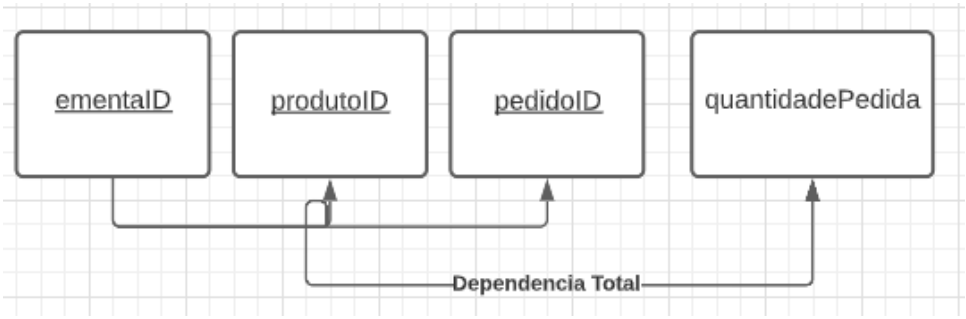

Figura 9 – Tabela PedidoEmentaProduto.

Produto:

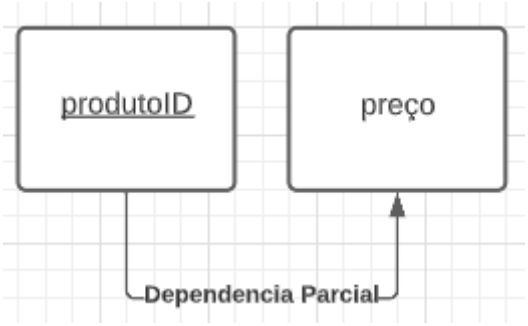

Figura 10 – Tabela Produto.

Pedido:

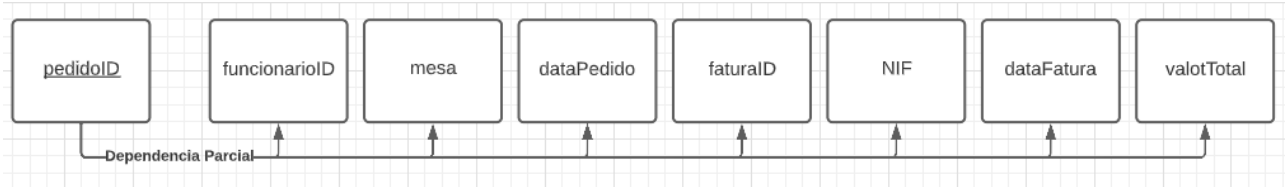

Figura 11 – Tabela Pedido.

3º Forma Normal.

A relação está na 2ª Forma Normal, mas não tem dependências transitivas (não existirem atributos descritores a dependerem funcionalmente de outros atributos descritores (não chaves)), ou seja, assume-se que se encontra na 3ª Forma Normal

Conclusão deste mockup.

Tabelas:

Produto { produtoID, produtoNome, produtoPreco, tipoID }

Chave primária: produtoID.

Chave estrangeira: tipoID.refere o ProdutoTipo

ProdutoTipo { tipoID, tipoNome }

Chave primária: tipoID.

Pedido { pedidoID, funcionarioId, mesa, dataPedido, faturaID, dataFatura, NIF, valorTotal }

Chave primária: pedidoID.

Chave estrangeira: funcionarioID refere Funcionário

PedidoEmentaProduto { pedidoID, ementaID, produtoID, quantidade }

Chave primária: pedidoID, produtoID, ementaID.

Chave estrangeira: produtoID, ementaID refere o EmentaProduto.

4.2.4.3 Receita

MICHELIN STAR			
Nº Produto:			
Nome:			
IngredienteID	nome	Qtd	Unit

Figura 12 – Mockup Receita

Forma não normalizada.

Figura 13 – Forma não normalizada.

1º Forma Normal.

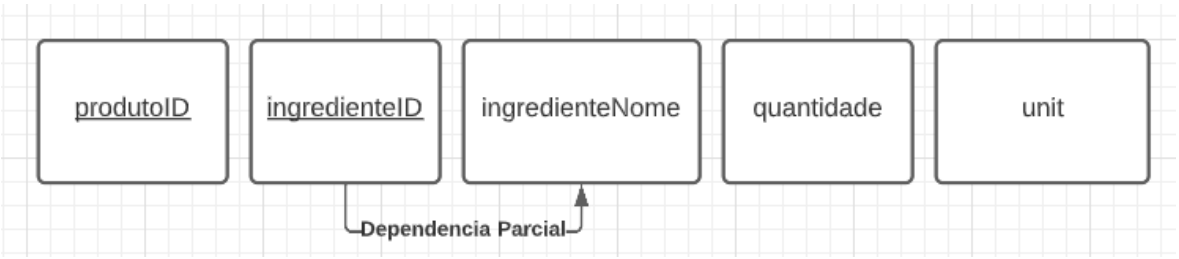

Figura 14 – Primeira forma normal.

Chegamos á conclusão que existe uma repetição do unit do ingrediente portanto criou-se uma nova relação.

Unit { unitID, unitNome}
Chave primária: unitID.

2º Forma Normal.

Tabelas:

Receita:

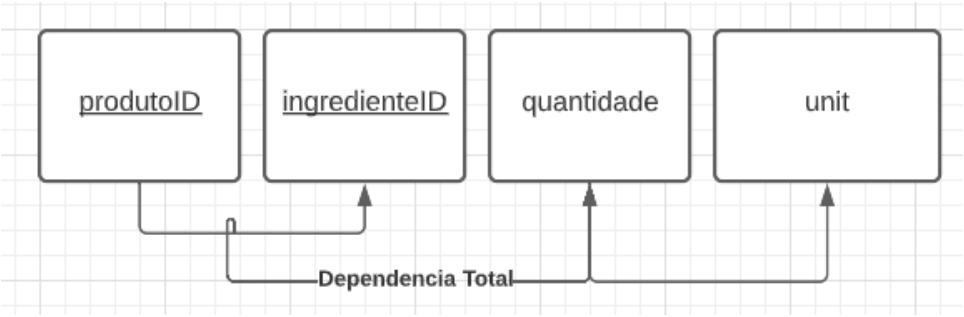

Figura 15 – Tabela Receita.

Ingrediente:

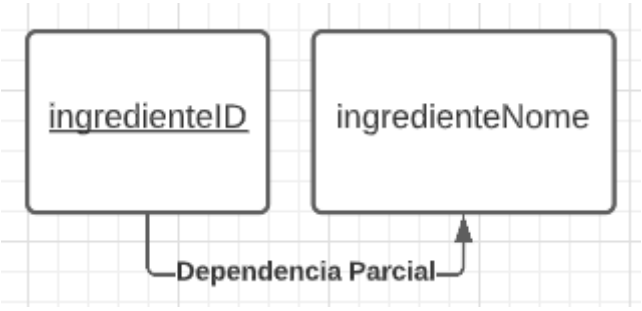

Figura 16 – Tabela ingrediente.

3º Forma Normal.

A relação está na 2ª Forma Normal, mas não tem dependências transitivas (não existirem atributos descritores a dependerem funcionalmente de outros atributos descritores (não chaves)), ou seja, assume-se que se encontra na 3ª Forma Normal

Conclusão deste mockup.

Tabelas:

Receita { produtoID, ingredienteID, quantidade, unitID }

Chave primária: produtoID, ingredienteID

Chave estrangeira: produtoID refere o Produto

ingredienteID refere o ingrediente

unitID refere Unit

Ingrediente{ ingredienteID, nome}

Chave primária: ingredienteID.

Unit { unitID, unitNome}

Chave primária: unitID.

4.2.4.4 Registar Funcionário

Figura 17 – Mockup registo funcionário.

Forma não normalizada.

Figura 18 – Forma não normalizada.

1º Forma Normal.

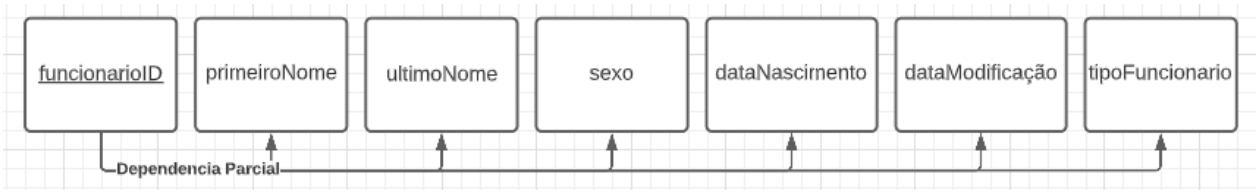

Figura 19 – Primeira forma normal.

Chegamos á conclusão que existe uma repetição do tipo do funcionário portanto criou-se uma nova relação.

FuncionárioTipo{ funcTipoID, funcTipoNome}
Chave primária: funcTipoID.

2º Forma Normal.

Tabelas:

Funcionário:

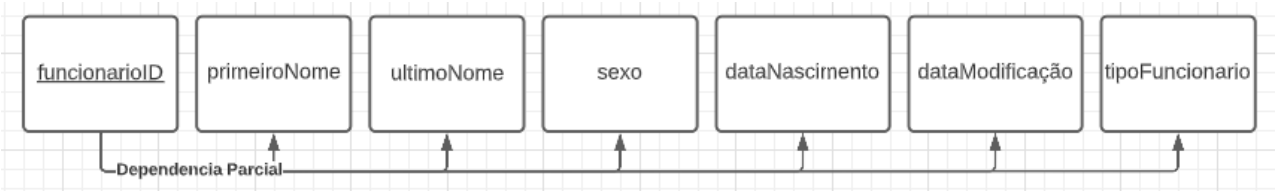

Figura 20 – Tabela Funcioário.

3º Forma Normal.

A relação está na 2ª Forma Normal, mas não tem dependências transitivas (não existirem atributos descritores a dependerem funcionalmente de outros atributos descritores (não chaves)), ou seja, assume-se que se encontra na 3ª Forma Normal

Conclusão deste mockup.

Tabelas:

Funcionário { funcionarioID, primeiroNome, ultimoNome, sexo, dataDeNascimento, dataDeModificacao, funcTipoID}

Chave primaria: funcionarioID

Chave estrangeira: funcTipoID refere FuncionárioTipo

FuncionárioTipo{ funcTipoID, funcTipoNome}

Chave primária: funcTipoID.

4.2.5 Conclusão do desenho lógico

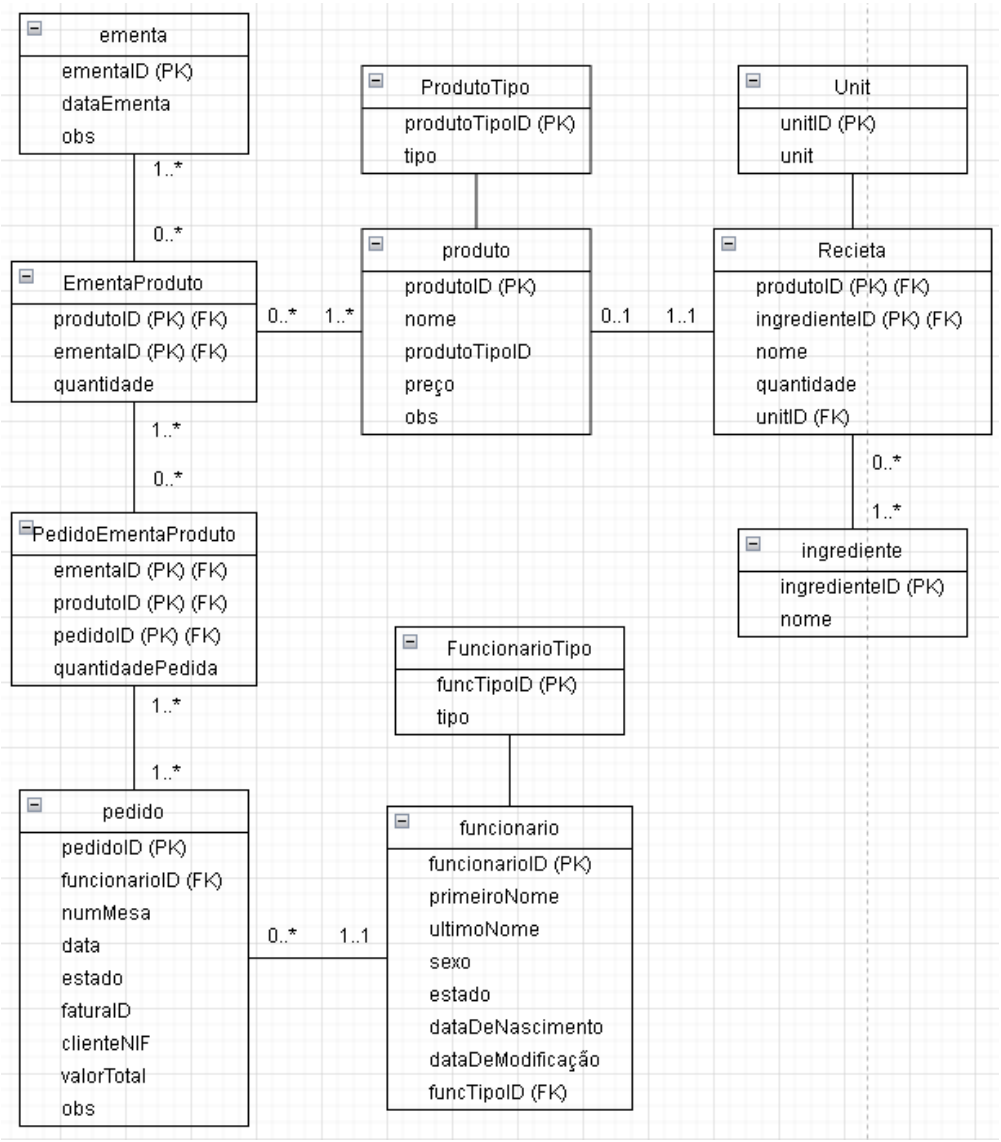

Figura 21 – Desenho lógico.

4.3 Restrições de integridade

4.3.1 Restrições

Funcionário

- primeiroNome e ultimoNome, funcionarioID, sexo, dataDeNascimento, dataDeModificacao, tipoFuncionario, estado são campos de preenchimento obrigatório.
- O sexo é definido por M – Masculino, F – Feminino ou O – outro.
- A dataDeNascimento deve ter o formato XX-XX-XXXX.
- O funcionarioID deve ter até 3 números inteiros.
- O estado é ativo (1) ou inativo (0).
- A dataDeModificacao deve ter o formato XX-XX-XXXX.

TipoFuncionario

- Todos os campos são obrigatórios

Pedido

- pedidoIDF, numMesa, data, estado, faturaID, clienteNIF são campos de preenchimento obrigatório.
- O numMesa deve estar entre os valores 1-20.
- O pedidoID deve ter até 3 números inteiros.
- A data, do pedido, deve ter o formato XX-XX-XXXX.
- A faturaID, deve ter até 3 números inteiros.
- O estado, deve ser pago (1) ou não pago (0).
- O clienteNIF deve ter exatamente 9 dígitos numéricos.
- Obs (opcional)

Ingrediente

- ingredienteID, nome, são campos de preenchimento obrigatório.
- O nome, deve ter até 30 caracteres variáveis.

Unit

- Todos os campos são obrigatórios

Produto

- produtoID, nome, tipo, preço são campos de preenchimento obrigatório.
- O produtoID deve ter até 2 números inteiros.
- O nome deve ter até 15 caracteres variáveis.
- O preço, do produto, deve ser do tipo decimal,
- Obs (opcional)

TipoProduto

- Todos os campos são obrigatórios

Ementa

- ementaID, dataEmenta, são campos de preenchimento obrigatório.
- A ementaID, deve ter até 3 números inteiros.
- Obs (opcional)

PedidoEmentaProduto

- Todos os campos são obrigatórios

EmentaProduto

- Todos os campos são obrigatórios

Receita

- Todos os campos são obrigatórios

4.3.2 Integridade referencial

EmentaProduto { ementaID, produtoID, quantidade}

Chave estrangeira	UPDATE	DELETE
productID	NO ACTION	NO ACTION
ementaID	NO ACTION	NO ACTION

Produto { produtoID, produtoNome, produtoPreco, tipoID }

Chave estrangeira	UPDATE	DELETE
tipoID	CASCADE	NO ACTION

PedidoEmentaProduto { pedidoID, ementaID, produtoID, quantidade}

Chave estrangeira	UPDATE	DELETE
pedidoID	NO ACTION	NO ACTION
ementaID, productID	NO ACTION	NO ACTION

Receita { produtoID, ingredienteID, quantidade}

Chave estrangeira	UPDATE	DELETE
productID	NO ACTION	NO ACTION
ingredienteID	NO ACTION	NO ACTION
unitID	CASCADE	NO ACTION

Pedido { pedidoID, funcionarioId, mesa, dataPedido, faturaID, dataFatura, NIF, valorTotal }

Chave estrangeira	**UPDATE**	**DELETE**
funcionarioID	CASCADE	NO ACTION

Funcionário { funcionarioID, primeiroNome, ultimoNome, sexo, dataDeNascimento, estado dataDeModificacao, funcTipoID}

Chave estrangeira	**UPDATE**	**DELETE**
funcTipoID	CASCADE	NO ACTION

6 Desenho Físico

Nesta Etapa, o modelo logico será traduzido para um modelo físico através de um SGBD, que neste caso foi usado o Microsoft SQL Server, e relatado o processo nas fases:

1. Desenho das relações e restrições
2. Representação de Dados Derivados
3. Desenho das Restrições Gerais
4. Desenho das Vistas de Utilizador

Criação de tipos de dados

Tabela 13 - Tipos de dados

Nome	Tipo de dado	Tamanho | precisão | escala
tipoFuncionarioID	Numeric	(1)
funcionarioID	Numeric	(3)
NIF	Numeric	(9)
pedidoID	Numeric	(3)
faturaID	Numeric (Unique)	(3)
ingredienteID	Numeric	(3)
unitID	Numeric	(1)
ementaID	Numeric	(2)
produtoID	Numeric	(2)
tipoProdutoID	Caracter	Tamanho: 2

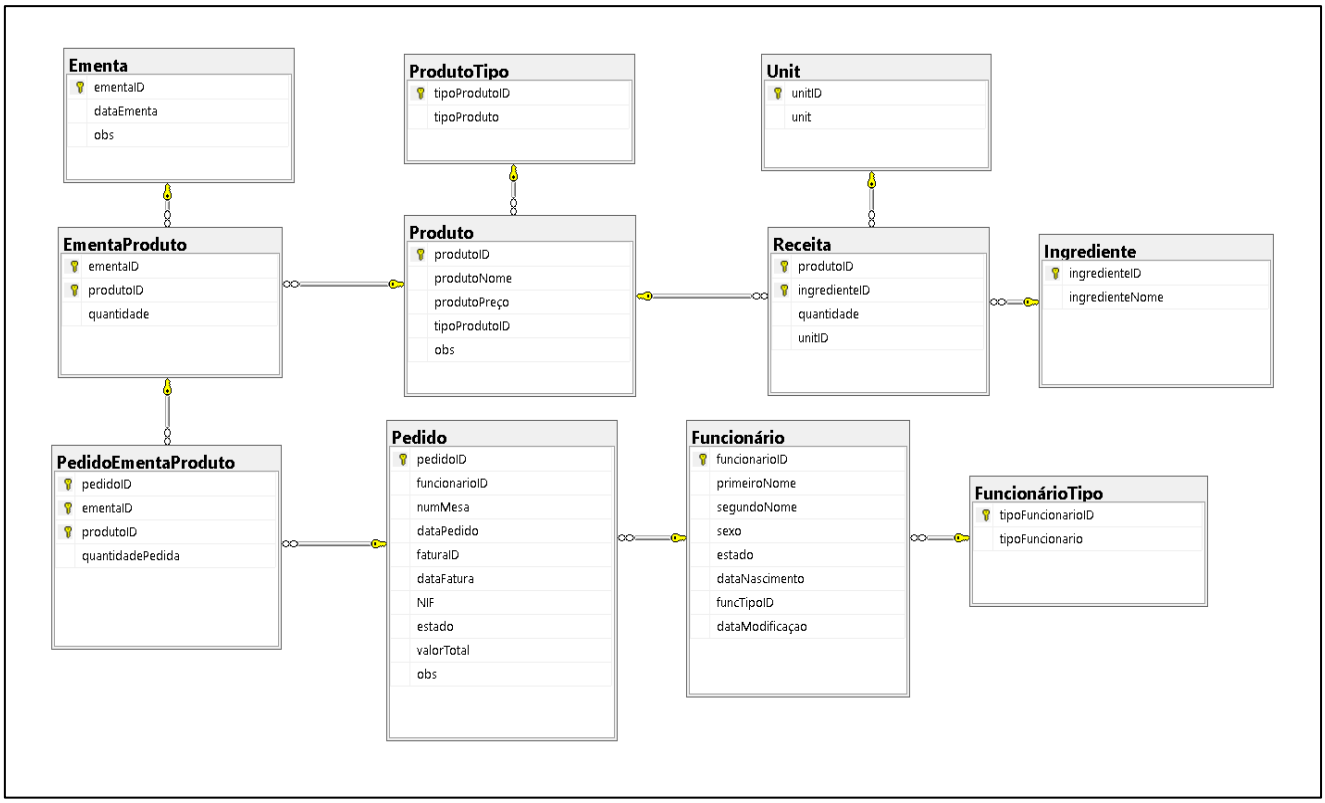

Figura 22 – Diagrama ER do SQL Management Studio

6.1 Criação da Tabelas, relacionamentos e restrições aos atributos

6.1.1 Criação Tabela Unit

```
USE [Restaurant]
GO

/****** Object:  Table [dbo].[Unit]    Script Date: 14/06/2022 17:45:23 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Unit](
    [unitID] [dbo].[unitID] NOT NULL,
    [unit] [varchar](30) NOT NULL,
 CONSTRAINT [PK_Unit] PRIMARY KEY CLUSTERED
(
    [unitID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Unit]  WITH CHECK ADD  CONSTRAINT [CK_Unit] CHECK  (([unitID]>(0)))
GO

ALTER TABLE [dbo].[Unit] CHECK CONSTRAINT [CK_Unit]
GO
```

6.1.2 Criação Tabela ProdutoTipo

```
USE [Restaurant]
GO

/****** Object:  Table [dbo].[ProdutoTipo]    Script Date: 14/06/2022 20:29:03 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[ProdutoTipo](
    [tipoProdutoID] [dbo].[tipoProdutoID] NOT NULL,
    [tipoProduto] [varchar](20) NOT NULL,
 CONSTRAINT [PK_ProdutoTipo] PRIMARY KEY CLUSTERED
(
    [tipoProdutoID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[ProdutoTipo]  WITH CHECK ADD  CONSTRAINT [CK_ProdutoTipo] CHECK  (([tipoProdutoID]='P' OR
    [tipoProdutoID]='C'OR [tipoProdutoID]='S' OR [tipoProdutoID]='B' OR [tipoProdutoID]='SO' OR [tipoProdutoID]='E'))
GO

ALTER TABLE [dbo].[ProdutoTipo] CHECK CONSTRAINT [CK_ProdutoTipo]
GO
```

6.1.3 Criação Tabela FuncionarioTipo

```
USE [Restaurant]
GO

/****** Object:  Table [dbo].[FuncionárioTipo]    Script Date: 14/06/2022 17:41:00 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[FuncionárioTipo](
    [tipoFuncionarioID] [dbo].[tipoFuncionarioID] NOT NULL,
    [tipoFuncionario] [varchar](30) NOT NULL,
 CONSTRAINT [PK_FuncionárioTipo] PRIMARY KEY CLUSTERED
(
    [tipoFuncionarioID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[FuncionárioTipo]  WITH CHECK ADD  CONSTRAINT [CK_FuncionárioTipo] CHECK  (([tipoFuncionarioID]>(0)))
GO

ALTER TABLE [dbo].[FuncionárioTipo] CHECK CONSTRAINT [CK_FuncionárioTipo]
GO
```

6.1.4 Criação Tabela Funcionario

```
USE [Restaurant]
GO

/****** Object:  Table [dbo].[Funcionário]    Script Date: 14/06/2022 17:39:55 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Funcionário](
    [funcionarioID] [dbo].[funcionarioID] NOT NULL,
    [primeiroNome] [varchar](30) NOT NULL,
    [segundoNome] [varchar](30) NOT NULL,
    [sexo] [varchar](1) NOT NULL,
    [estado] [numeric](1, 0) NOT NULL,
    [dataNascimento] [date] NOT NULL,
    [funcTipoID] [dbo].[tipoFuncionarioID] NOT NULL,
    [dataModificaçao] [date] NOT NULL,
 CONSTRAINT [PK_Funcionário] PRIMARY KEY CLUSTERED
(
    [funcionarioID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Funcionário]  WITH CHECK ADD  CONSTRAINT [FK_Funcionário_FuncionárioTipo] FOREIGN KEY([funcTipoID])
REFERENCES [dbo].[FuncionárioTipo] ([tipoFuncionarioID])
ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[Funcionário] CHECK CONSTRAINT [FK_Funcionário_FuncionárioTipo]
GO

ALTER TABLE [dbo].[Funcionário]  WITH CHECK ADD  CONSTRAINT [CK_Funcionário] CHECK  (([funcionarioID]>(0)))
GO

ALTER TABLE [dbo].[Funcionário] CHECK CONSTRAINT [CK_Funcionário]
GO

ALTER TABLE [dbo].[Funcionário]  WITH CHECK ADD  CONSTRAINT [CK_Funcionário_Estado] CHECK  (([estado]=(0) OR [estado]=(1)))
GO

ALTER TABLE [dbo].[Funcionário] CHECK CONSTRAINT [CK_Funcionário_Estado]
GO

ALTER TABLE [dbo].[Funcionário]  WITH CHECK ADD  CONSTRAINT [Sexo] CHECK  (([sexo]='M' OR [sexo]='F' OR [sexo]='O'))
GO

ALTER TABLE [dbo].[Funcionário] CHECK CONSTRAINT [Sexo]
GO
```

6.1.5 Criação Tabela Ementa

```
USE [Restaurant]
GO

/****** Object:  Table [dbo].[Ementa]    Script Date: 14/06/2022 17:37:22 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Ementa](
    [ementaID] [dbo].[ementaID] NOT NULL,
    [dataEmenta] [date] NOT NULL,
    [obs] [varchar](250) NULL,
 CONSTRAINT [PK_Ementa] PRIMARY KEY CLUSTERED
(
    [ementaID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Ementa]  WITH CHECK ADD  CONSTRAINT [CK_Ementa] CHECK  (([ementaID]>(0)))
GO

ALTER TABLE [dbo].[Ementa] CHECK CONSTRAINT [CK_Ementa]
GO
```

6.1.6 Criação Tabela Pedido

```
USE [Restaurant]
GO

/****** Object:  Table [dbo].[Pedido]    Script Date: 14/06/2022 17:46:44 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Pedido](
    [pedidoID] [dbo].[pedidoID] NOT NULL,
    [funcionarioID] [dbo].[funcionarioID] NOT NULL,
    [numMesa] [numeric](2, 0) NOT NULL,
    [dataPedido] [date] NOT NULL,
    [faturaID] [dbo].[faturaID] NULL,
    [dataFatura] [date] NULL,
    [NIF] [dbo].[NIF] NULL,
    [estado] [numeric](1, 0) NOT NULL,
    [valorTotal] [numeric](6, 2) NOT NULL,
    [obs] [varchar](250) NULL,
 CONSTRAINT [PK_Pedido] PRIMARY KEY CLUSTERED
(
    [pedidoID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Pedido] ADD  CONSTRAINT [DF_Pedido_estado]  DEFAULT ((0)) FOR [estado]
GO

ALTER TABLE [dbo].[Pedido]  WITH CHECK ADD  CONSTRAINT [FK_Pedido_Funcionário] FOREIGN KEY([funcionarioID])
REFERENCES [dbo].[Funcionário] ([funcionarioID])
ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[Pedido] CHECK CONSTRAINT [FK_Pedido_Funcionário]
GO

ALTER TABLE [dbo].[Pedido]  WITH CHECK ADD  CONSTRAINT [CK_NIF] CHECK  (([NIF]>(0)))
GO

ALTER TABLE [dbo].[Pedido] CHECK CONSTRAINT [CK_NIF]
GO

ALTER TABLE [dbo].[Pedido]  WITH CHECK ADD  CONSTRAINT [CK_Pedido] CHECK  (([pedidoID]>(0)))
GO

ALTER TABLE [dbo].[Pedido] CHECK CONSTRAINT [CK_Pedido]
GO

ALTER TABLE [dbo].[Pedido]  WITH CHECK ADD  CONSTRAINT [CK_Pedido_Estado] CHECK  (([estado]=(1) OR [estado]=(0)))
GO

ALTER TABLE [dbo].[Pedido] CHECK CONSTRAINT [CK_Pedido_Estado]
GO

ALTER TABLE [dbo].[Pedido]  WITH CHECK ADD  CONSTRAINT [CK_Pedido_Mesa] CHECK  (([numMesa]>(0) AND [numMesa]<(21)))
GO

ALTER TABLE [dbo].[Pedido] CHECK CONSTRAINT [CK_Pedido_Mesa]
GO
```

6.1.7 Criação Tabela EmentaProduto

```
USE [Restaurant]
GO

/****** Object:  Table [dbo].[EmentaProduto]    Script Date: 16/06/2022 14:47:50 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[EmentaProduto](
    [ementaID] [dbo].[ementaID] NOT NULL,
    [produtoID] [dbo].[produtoID] NOT NULL,
    [quantidade] [numeric](2, 0) NOT NULL,
 CONSTRAINT [PK_EmentaProduto] PRIMARY KEY CLUSTERED
(
    [ementaID] ASC,
    [produtoID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[EmentaProduto]  WITH CHECK ADD  CONSTRAINT [FK_EmentaProduto_Ementa] FOREIGN KEY([ementaID])
REFERENCES [dbo].[Ementa] ([ementaID])
ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[EmentaProduto] CHECK CONSTRAINT [FK_EmentaProduto_Ementa]
GO

ALTER TABLE [dbo].[EmentaProduto]  WITH CHECK ADD  CONSTRAINT [FK_EmentaProduto_Produto] FOREIGN KEY([produtoID])
REFERENCES [dbo].[Produto] ([produtoID])
ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[EmentaProduto] CHECK CONSTRAINT [FK_EmentaProduto_Produto]
GO

ALTER TABLE [dbo].[EmentaProduto]  WITH CHECK ADD  CONSTRAINT [CK_EmentaProduto] CHECK  (([quantidade]>(0)))
GO

ALTER TABLE [dbo].[EmentaProduto] CHECK CONSTRAINT [CK_EmentaProduto]
GO
```

6.1.8 Criação Tabela Ingrediente

```
USE [Restaurant]
GO

/****** Object:  Table [dbo].[Ingrediente]    Script Date: 14/06/2022 17:41:34 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Ingrediente](
    [ingredienteID] [dbo].[ingredienteID] NOT NULL,
    [ingredienteNome] [varchar](30) NOT NULL,
 CONSTRAINT [PK_Ingrediente] PRIMARY KEY CLUSTERED
(
    [ingredienteID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Ingrediente]  WITH CHECK ADD  CONSTRAINT [CK_Ingrediente] CHECK  (([ingredienteID]>(0)))
GO

ALTER TABLE [dbo].[Ingrediente] CHECK CONSTRAINT [CK_Ingrediente]
GO
```

6.1.9 Criação Tabela Produto

```
USE [Restaurant]
GO

/****** Object:  Table [dbo].[Produto]    Script Date: 16/06/2022 14:48:37 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Produto](
    [produtoID] [dbo].[produtoID] NOT NULL,
    [produtoNome] [varchar](30) NOT NULL,
    [produtoPreço] [decimal](6, 2) NOT NULL,
    [tipoProdutoID] [dbo].[tipoProdutoID] NOT NULL,
    [obs] [varchar](250) NULL,
 CONSTRAINT [PK_Produto] PRIMARY KEY CLUSTERED
(
    [produtoID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Produto]  WITH CHECK ADD  CONSTRAINT [FK_Produto_ProdutoTipo] FOREIGN KEY([tipoProdutoID])
REFERENCES [dbo].[ProdutoTipo] ([tipoProdutoID])
ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[Produto] CHECK CONSTRAINT [FK_Produto_ProdutoTipo]
GO

ALTER TABLE [dbo].[Produto]  WITH CHECK ADD  CONSTRAINT [CK_Produto] CHECK  (([produtoID]>(0)))
GO

ALTER TABLE [dbo].[Produto] CHECK CONSTRAINT [CK_Produto]
GO

ALTER TABLE [dbo].[Produto]  WITH CHECK ADD  CONSTRAINT [CK_Produto_Preço] CHECK  (([produtoPreço]>(0)))
GO

ALTER TABLE [dbo].[Produto] CHECK CONSTRAINT [CK_Produto_Preço]
GO
```

6.1.10 Criação Tabela Receita

```
USE [Restaurant]
GO

/****** Object:  Table [dbo].[Receita]    Script Date: 16/06/2022 14:49:16 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Receita](
    [produtoID] [dbo].[produtoID] NOT NULL,
    [ingredienteID] [dbo].[ingredienteID] NOT NULL,
    [quantidade] [numeric](3, 0) NOT NULL,
    [unitID] [dbo].[unitID] NOT NULL,
 CONSTRAINT [PK_Receita] PRIMARY KEY CLUSTERED
(
    [produtoID] ASC,
    [ingredienteID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Receita]  WITH CHECK ADD  CONSTRAINT [FK_Receita_Ingrediente] FOREIGN KEY([ingredienteID])
REFERENCES [dbo].[Ingrediente] ([ingredienteID])
GO

ALTER TABLE [dbo].[Receita] CHECK CONSTRAINT [FK_Receita_Ingrediente]
GO

ALTER TABLE [dbo].[Receita]  WITH CHECK ADD  CONSTRAINT [FK_Receita_Produto] FOREIGN KEY([produtoID])
REFERENCES [dbo].[Produto] ([produtoID])
GO

ALTER TABLE [dbo].[Receita] CHECK CONSTRAINT [FK_Receita_Produto]
GO

ALTER TABLE [dbo].[Receita]  WITH CHECK ADD  CONSTRAINT [FK_Receita_Unit] FOREIGN KEY([unitID])
REFERENCES [dbo].[Unit] ([unitID])
ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[Receita] CHECK CONSTRAINT [FK_Receita_Unit]
GO

ALTER TABLE [dbo].[Receita]  WITH CHECK ADD  CONSTRAINT [CK_Receita_Quantidade] CHECK  (([quantidade]>(0)))
GO

ALTER TABLE [dbo].[Receita] CHECK CONSTRAINT [CK_Receita_Quantidade]
GO
```

6.1.11 Criação Tabela PedidoEmentaProduto

```
USE [Restaurant]
GO

/****** Object:  Table [dbo].[PedidoEmentaProduto]    Script Date: 16/06/2022 14:49:55 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[PedidoEmentaProduto](
    [pedidoID] [dbo].[pedidoID] NOT NULL,
    [ementaID] [dbo].[ementaID] NOT NULL,
    [produtoID] [dbo].[produtoID] NOT NULL,
    [quantidadePedida] [numeric](3, 0) NOT NULL,
 CONSTRAINT [PK_PedidoEmentaProduto] PRIMARY KEY CLUSTERED
(
    [pedidoID] ASC,
    [ementaID] ASC,
    [produtoID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[PedidoEmentaProduto]  WITH CHECK ADD  CONSTRAINT [FK_PedidoEmentaProduto_EmentaProduto] FOREIGN KEY([ementaID], [produtoID])
REFERENCES [dbo].[EmentaProduto] ([ementaID], [produtoID])
GO

ALTER TABLE [dbo].[PedidoEmentaProduto] CHECK CONSTRAINT [FK_PedidoEmentaProduto_EmentaProduto]
GO

ALTER TABLE [dbo].[PedidoEmentaProduto]  WITH CHECK ADD  CONSTRAINT [FK_PedidoEmentaProduto_Pedido] FOREIGN KEY([pedidoID])
REFERENCES [dbo].[Pedido] ([pedidoID])
GO

ALTER TABLE [dbo].[PedidoEmentaProduto] CHECK CONSTRAINT [FK_PedidoEmentaProduto_Pedido]
GO

ALTER TABLE [dbo].[PedidoEmentaProduto]  WITH CHECK ADD  CONSTRAINT [CK_PedidoEmentaProduto_Quantidade] CHECK  (([quantidadePedida]>(0)))
GO

ALTER TABLE [dbo].[PedidoEmentaProduto] CHECK CONSTRAINT [CK_PedidoEmentaProduto_Quantidade]
GO
```

6.2 T-SQL

6.2.1 Triggers

Nesta Base de dados foram criados 4 triggers de forma a validar algumas das restrições mais importantes do negócio.

Os triggers criados para a base ed dados foram:

-PedidoEmentaProduto_DeleteLinha

Este trigger consiste em que sempre um produto por eliminado de um pedido, o valor total do pedido diminuia e a quantidade daquele produto que está para ser servido naquela ementa aumente.

-PedidoEmentaProduto_UpdateLinha

Este trigger consiste em que sempre for atualizado uma linha do pedido, isto é alterar a quantidade pedida ou mesmo o produto, o valor total do pedido atualize também assim como a quantidade daquele produto que está para ser servido naquela ementa.

-PedidoEmentaProduto_InsertLinha

Este trigger consiste em que sempre for inserido um novo produto ao pedido, verifique se o produto está na ementa do dia do pedido, se existe quantidade suficiente para satisfazer a quantidade pedida, se isto estiver de acordo como o previsto o produto é inserido e o valor total do pedido é incrementado e a quantidade daquele produto diminui.

-Funcionário_Delete

Este trigger consiste em que sempre um funcionario for eliminado, ele não ser eliminado da base de dados mais sim passar para o estado Inativo (0).

6.2.1.1 Trigger PedidoEmentaProduto_DeleteLinha

```
USE [Restaurant]
GO

/****** Object:  Trigger [dbo].[PedidoEmentaProduto_DeleteLinha]    Script Date: 16/06/2022 19:57:34 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

-- =============================================
-- Author:      <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Description,,>
-- =============================================
CREATE TRIGGER [dbo].[PedidoEmentaProduto_DeleteLinha]
   ON   [dbo].[PedidoEmentaProduto]
   INSTEAD OF DELETE
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    DECLARE @pedidoAntigoID NUMERIC (3)
    DECLARE @ementaAntigoID NUMERIC(3)
    DECLARE @produtoAntigoID NUMERIC(3)
    DECLARE @dataEmenta DATE
    DECLARE @dataPedido DATE
    DECLARE @antigaQuantidade NUMERIC(2)
    DECLARE @antigaQuantidadePedida NUMERIC (2)
    DECLARE @antigoProdutoPreco NUMERIC (6,2)

    SELECT @produtoAntigoID = deleted.produtoID,
           @pedidoAntigoID = deleted.pedidoID,
           @ementaAntigoID = deleted.ementaID,
           @antigaQuantidadePedida  = deleted.quantidadePedida
    FROM deleted

    SET @dataEmenta = (SELECT dataEmenta FROM Ementa WHERE ementaID = @ementaAntigoID)
    SET @dataPedido = (SELECT dataPedido FROM Pedido WHERE pedidoID = @pedidoAntigoID)
    SET @antigaQuantidade = (SELECT quantidade FROM EmentaProduto WHERE ementaID = @ementaAntigoID AND produtoID = @produtoAntigoID)

    BEGIN
        SELECT @antigoProdutoPreco = produtoPreço
        FROM Produto
        WHERE produtoID = @produtoAntigoID

        UPDATE EmentaProduto
        SET quantidade = quantidade + @antigaQuantidadePedida
        WHERE ementaID = @ementaAntigoID AND produtoID = @produtoAntigoID

        -- Aumentar o valor total do pedido
        UPDATE Pedido
        SET valorTotal = valorTotal - (@antigoProdutoPreco * @antigaQuantidadePedida)
        WHERE pedidoID = @pedidoAntigoID

        DELETE PedidoEmentaProduto
        WHERE pedidoID = @pedidoAntigoID AND ementaID = @ementaAntigoID AND produtoID = @produtoAntigoID

    END
END

ALTER TABLE [dbo].[PedidoEmentaProduto] ENABLE TRIGGER [PedidoEmentaProduto_DeleteLinha]

GO
```

6.2.1.2 Trigger PedidoEmentaProduto_UpdateLinha

```
USE [Restaurant]
GO

/****** Object:  Trigger [dbo].[PedidoEmentaProduto_UpdateLinha]    Script Date: 16/06/2022 20:00:43 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

-- =============================================
-- Author:      <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Description,,>
-- =============================================
CREATE TRIGGER [dbo].[PedidoEmentaProduto_UpdateLinha]
   ON   [dbo].[PedidoEmentaProduto]
   INSTEAD OF UPDATE
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    DECLARE @pedidoID NUMERIC(3)
    DECLARE @pedidoAntigoID NUMERIC (3)
    DECLARE @ementaID NUMERIC(3)
    DECLARE @ementaAntigoID NUMERIC(3)
    DECLARE @produtoID NUMERIC(2)
    DECLARE @produtoAntigoID NUMERIC(3)
    DECLARE @dataEmenta DATE
    DECLARE @dataPedido DATE
    DECLARE @quantidade NUMERIC(2)
    DECLARE @antigaQuantidade NUMERIC(2)
    DECLARE @produtoPreco NUMERIC (6,2)
    DECLARE @quantidadePedida NUMERIC(2)
    DECLARE @antigaQuantidadePedida NUMERIC (2)
    DECLARE @antigoProdutoPreco NUMERIC (6,2)
    DECLARE @pedidoEstado NUMERIC(1)

    SELECT @produtoID = inserted.produtoID,
           @pedidoID = inserted.pedidoID,
           @ementaID = inserted.ementaID,
           @quantidadePedida  = inserted.quantidadePedida
    FROM inserted

    SELECT @produtoAntigoID = deleted.produtoID,
           @pedidoAntigoID = deleted.pedidoID,
           @ementaAntigoID = deleted.ementaID,
           @antigaQuantidadePedida  = deleted.quantidadePedida
    FROM deleted

    SELECT @pedidoEstado = estado
    FROM Pedido
    WHERE pedidoID = @pedidoID

    SET @dataEmenta = (SELECT dataEmenta FROM Ementa WHERE ementaID = @ementaID)
    SET @dataPedido = (SELECT dataPedido FROM Pedido WHERE pedidoID = @pedidoID)
    SET @antigaQuantidade = (SELECT quantidade FROM EmentaProduto WHERE ementaID = @ementaAntigoID AND produtoID = @produtoAntigoID)

    IF (@pedidoEstado = 1)
    BEGIN
        RAISERROR('O produto nao pode ser adicionado a um pedido já pago!!',16,1)
    END
```

```
    IF (@dataEmenta != @dataPedido)
    BEGIN
        RAISERROR('O produto tem de estar na ementa do mesmo dia do pedido!!',16,1);
    END
    ELSE
    IF (@antigaQuantidade + @antigaQuantidadePedida < @quantidadePedida)
    BEGIN
        RAISERROR('Não ha quantidade suficiente para satisfazer o pedido!!',16,1);
    END
    ELSE
    BEGIN
        SELECT @produtoPreco = produtoPreço
        FROM Produto
        WHERE produtoID = @produtoID

        SELECT @antigoProdutoPreco = produtoPreço
        FROM Produto
        WHERE produtoID = @produtoAntigoID

        UPDATE EmentaProduto
        SET quantidade = quantidade + @antigaQuantidadePedida
        WHERE ementaID = @ementaAntigoID AND produtoID = @produtoAntigoID

        UPDATE EmentaProduto
        SET quantidade = quantidade - @quantidadePedida
        WHERE ementaID = @ementaID AND produtoID = @produtoID

        -- Aumentar o valor total do pedido
        UPDATE Pedido
        SET valorTotal = valorTotal - (@antigoProdutoPreco * @antigaQuantidadePedida)
        WHERE pedidoID = @pedidoAntigoID

        UPDATE Pedido
        SET valorTotal = valorTotal + (@produtoPreco * @quantidadePedida)
        WHERE pedidoID = @pedidoID

        UPDATE EmentaProduto
        SET quantidade = quantidade - @quantidadePedida
        WHERE ementaID = @ementaID AND produtoID = @produtoID

        -- Aumentar o valor total do pedido
        UPDATE Pedido
        SET valorTotal = valorTotal - (@antigoProdutoPreco * @antigaQuantidadePedida)
        WHERE pedidoID = @pedidoAntigoID

        UPDATE Pedido
        SET valorTotal = valorTotal + (@produtoPreco * @quantidadePedida)
        WHERE pedidoID = @pedidoID

        UPDATE PedidoEmentaProduto
        SET pedidoID = @pedidoID, produtoID = @produtoID, quantidadePedida = @quantidadePedida
        WHERE pedidoID = @pedidoAntigoID AND ementaID = @ementaAntigoID AND produtoID = @produtoAntigoID

    END
END

ALTER TABLE [dbo].[PedidoEmentaProduto] ENABLE TRIGGER [PedidoEmentaProduto_UpdateLinha]

GO
```

6.2.1.3 Trigger PedidoEmentaProduto_InsertLinha

```
USE [Restaurant]
GO

/****** Object:  Trigger [dbo].[PedidoEmentaProduto_InsertLinha]    Script Date: 16/06/2022 19:53:36 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

-- =============================================
-- Author:      <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Description,,>
-- =============================================
CREATE TRIGGER [dbo].[PedidoEmentaProduto_InsertLinha]
   ON   [dbo].[PedidoEmentaProduto]
   INSTEAD OF INSERT
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    DECLARE @pedidoID NUMERIC(3)
    DECLARE @ementaID NUMERIC(3)
    DECLARE @produtoID NUMERIC(2)
    DECLARE @dataEmenta DATE
    DECLARE @dataPedido DATE
    DECLARE @quantidade NUMERIC(2)
    DECLARE @produtoPreco NUMERIC (6,2)
    DECLARE @quantidadePedida NUMERIC(2)
    DECLARE @pedidoEstado NUMERIC(1)

    SELECT @produtoID = inserted.produtoID,
           @pedidoID = inserted.pedidoID,
           @ementaID = inserted.ementaID,
           @quantidadePedida  = inserted.quantidadePedida
    FROM inserted

    SELECT @pedidoEstado = estado
    FROM Pedido
    WHERE pedidoID = @pedidoID

    SET @dataEmenta = (SELECT dataEmenta FROM Ementa WHERE ementaID = @ementaID)
    SET @dataPedido = (SELECT dataPedido FROM Pedido WHERE pedidoID = @pedidoID)
    SET @quantidade = (SELECT quantidade FROM EmentaProduto WHERE ementaID = @ementaID AND produtoID = @produtoID)
```

```
    IF (@dataEmenta != @dataPedido)
    BEGIN
        RAISERROR('O produto tem de estar na ementa do mesmo dia do pedido!!',16,1);
    END

    IF (@quantidade < @quantidadePedida)
    BEGIN
        RAISERROR('Não ha quantidade suficiente para satisfazer o pedido!!',16,1);
    END
    ELSE
    BEGIN
        SELECT @produtoPreco = produtoPreço
        FROM Produto
        WHERE produtoID = @produtoID

        UPDATE EmentaProduto
        SET quantidade = quantidade - @quantidadePedida
        WHERE ementaID = @ementaID AND produtoID = @produtoID

        -- Aumentar o valor total do pedido
        UPDATE Pedido
        SET valorTotal = valorTotal + (@produtoPreco * @quantidadePedida)
        WHERE pedidoID = @pedidoID

        INSERT INTO PedidoEmentaProduto
        SELECT * FROM inserted
    END
END

ALTER TABLE [dbo].[PedidoEmentaProduto] ENABLE TRIGGER [PedidoEmentaProduto_InsertLinha]
GO
```

6.2.1.4 Trigger Funcionário_Delete

```
USE [Restaurant]
GO
/****** Object:  Trigger [dbo].[Funcionario_Delete]    Script Date: 16/06/2022 15:17:04 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:      <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Description,,>
-- =============================================
ALTER TRIGGER [dbo].[Funcionario_Delete]
   ON [dbo].[Funcionário]
   INSTEAD OF DELETE
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    DECLARE @funcionarioID NUMERIC (3)

    SELECT @funcionarioID = d.funcionarioID
    FROM deleted d

    UPDATE Funcionário
    SET estado = 0
    WHERE funcionarioID = @funcionarioID

END
```

6.2.2 Stored Procedures

Para cumprir as restrições foram criados quatro procedimentos:

-EfectuarPagamento

Este stored procedure é responsavel por validar se o pedido já foi pago ou não, se o pedido que vai efectuar o pagamento tem produtos inseridos, com estes parametros validos é inserido no pedido o NIF do cliente, a data da fatura, e o estado do pedido fica como pago.

-AddProdutoToPedido

Este stored procedure é responsavel por validar se o pedido já está pago ou não, se o produto está na ementa do dia do pedido e se existe quantidade suficiente do produto para satisfazer o cliente, com estes parametros validos o produto é adicionado com sucesso.

-DespedirFuncionario

Este stored procedure é responsavel por validar se o funcionario está ativo ou inativo, se o mesmo estiver ativo o funcionario é despedido, isto é o estado do funcionario passa para inativo (0), se não tiver é porque já tinha sido despedido.

-ReadmitirFuncionario

Este stored procedure é responsavel por validar se o funcionario está ativo ou inativo, se o mesmo estiver inativo o funcionario é readmitido, isto é o estado do funcionario passa para ativo (1), se não tiver é porque nunca foi despedido.

6.2.2.1 Efectuar Pagamento

```
USE [Restaurant]
GO
/****** Object:  StoredProcedure [dbo].[spPedido_EfetuarPagamento]    Script Date: 16/06/2022 15:47:19 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:      <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Description,,>
-- =============================================
ALTER PROCEDURE [dbo].[spPedido_EfetuarPagamento]
    @pedidoID NUMERIC (3),
    @NIF NUMERIC (9)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    DECLARE @pedidoEstado NUMERIC(1)
    DECLARE @dataPedido DATE
    DECLARE @dataFatura DATE
    DECLARE @valorTotal NUMERIC (6,2)
    DECLARE @faturaID NUMERIC (3)

    SET @faturaID = @pedidoID

    SELECT @pedidoEstado = estado, @dataPedido = dataPedido, @valorTotal = valorTotal
    FROM Pedido
    WHERE pedidoID = @pedidoID

    SET @dataFatura = @dataPedido

    IF (@pedidoEstado = 1)
    BEGIN
        RAISERROR('Este pedido já foi pago!!',16,1)
        RETURN
    END

    IF (@valorTotal = 0)
    BEGIN
        RAISERROR('Este pedido ainda não tem produtos!!',16,1)
        RETURN
    END

    SET @pedidoEstado = 1

    BEGIN TRY
        UPDATE Pedido
        SET NIF = @NIF, dataFatura = @dataFatura, faturaID = @faturaID, estado = @pedidoEstado
        WHERE pedidoID = @pedidoID
        PRINT 'O pagamento foi efetuado com sucesso.'
    END TRY
    BEGIN CATCH
        DECLARE @ErrorMessage NVARCHAR(4000)
        DECLARE @ErrorSeverity INT
        DECLARE @ErrorState INT

        SELECT @ErrorMessage = ERROR_MESSAGE(), @ErrorSeverity = ERROR_SEVERITY(), @ErrorState = ERROR_STATE()

        RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState)
    END CATCH

END
```

6.2.2.2 AddProdutoToPedido

```
USE [Restaurant]
GO
/****** Object:  StoredProcedure [dbo].[spPedidoEmentaProduto_AddProdutoToPedido]    Script Date: 16/06/2022 15:45:48 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:      <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Description,,>
-- =============================================
ALTER PROCEDURE [dbo].[spPedidoEmentaProduto_AddProdutoToPedido]
    @pedidoID NUMERIC (3),
    @ementaID NUMERIC (3),
    @produtoID NUMERIC (2),
    @quantidadePedida NUMERIC (3)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    DECLARE @pedidoEstado NUMERIC(1)
    DECLARE @dataEmenta DATE
    DECLARE @dataPedido DATE
    DECLARE @quantidade NUMERIC(2)

    SELECT @pedidoEstado = estado
    FROM Pedido
    WHERE pedidoID = @pedidoID

    SET @dataEmenta = (SELECT dataEmenta FROM Ementa WHERE ementaID = @ementaID)
    SET @dataPedido = (SELECT dataPedido FROM Pedido WHERE pedidoID = @pedidoID)
    SET @quantidade = (SELECT quantidade FROM EmentaProduto WHERE ementaID = @ementaID AND produtoID = @produtoID)

    IF (@pedidoEstado = 1)
    BEGIN
        RAISERROR('O produto nao pode ser adicionado a um pedido já pago!!',16,1)
        RETURN
    END

    IF (@dataEmenta != @dataPedido)
    BEGIN
        RAISERROR('O produto tem de estar na ementa do mesmo dia do pedido!!',16,1);
        RETURN
    END

    IF (@quantidade < @quantidadePedida)
    BEGIN
        RAISERROR('Não ha quantidade suficiente para satisfazer o pedido!!',16,1);
        RETURN
    END

    BEGIN TRY
        INSERT INTO PedidoEmentaProduto(pedidoID, ementaID, produtoID,quantidadePedida)
        VALUES (@pedidoID, @ementaID, @produtoID, @quantidadePedida)
        PRINT 'O produto foi inserido com sucesso.'
    END TRY
    BEGIN CATCH
        DECLARE @ErrorMessage NVARCHAR(4000)
        DECLARE @ErrorSeverity INT
        DECLARE @ErrorState INT

        SELECT @ErrorMessage = ERROR_MESSAGE(), @ErrorSeverity = ERROR_SEVERITY(), @ErrorState = ERROR_STATE()

        RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState)
    END CATCH

END
```

6.2.2.3 DespedirFuncionario

```
USE [Restaurant]
GO

/****** Object:  StoredProcedure [dbo].[spFuncionario_DespedirFuncionario]    Script Date: 17/06/2022 12:19:14
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

-- =============================================
-- Author:      <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Description,,>
-- =============================================
CREATE PROCEDURE [dbo].[spFuncionario_DespedirFuncionario]
    @funcionarioID NUMERIC (3)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    DECLARE @antigoEstado NUMERIC(1)

    SET @antigoEstado = (SELECT estado FROM Funcionário WHERE funcionarioID = @funcionarioID)

    IF (@antigoEstado = 0)
    BEGIN
        RAISERROR( 'O funcionário já foi eliminado anteriormente!!' ,16,1)
        RETURN
    END

    BEGIN TRY
    UPDATE Funcionário
    SET estado = 0
    WHERE funcionarioID = @funcionarioID
    END TRY

    BEGIN CATCH
        DECLARE @ErrorMessage NVARCHAR(4000)
        DECLARE @ErrorSeverity INT
        DECLARE @ErrorState INT

        SELECT @ErrorMessage = ERROR_MESSAGE(), @ErrorSeverity = ERROR_SEVERITY(), @ErrorState = ERROR_STATE()

        RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState)
    END CATCH
END
GO
```

6.2.2.4 ReadmitirFuncionario

```
USE [Restaurant]
GO

/****** Object:  StoredProcedure [dbo].[spFuncionario_ReadmitirFuncionario]    Script Date: 17/06/2022 12:20:55 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

-- =============================================
-- Author:      <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Description,,>
-- =============================================
CREATE PROCEDURE [dbo].[spFuncionario_ReadmitirFuncionario]
    @funcionarioID NUMERIC (3)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    DECLARE @antigoEstado NUMERIC(1)

    SET @antigoEstado = (SELECT estado FROM Funcionário WHERE funcionarioID = @funcionarioID)

    IF (@antigoEstado = 1)
    BEGIN
        RAISERROR( 'O funcionário já está ativo!!' ,16,1)
        RETURN
    END

    BEGIN TRY
    UPDATE Funcionário
    SET estado = 1
    WHERE funcionarioID = @funcionarioID
    END TRY

    BEGIN CATCH
        DECLARE @ErrorMessage NVARCHAR(4000)
        DECLARE @ErrorSeverity INT
        DECLARE @ErrorState INT

        SELECT @ErrorMessage = ERROR_MESSAGE(), @ErrorSeverity = ERROR_SEVERITY(), @ErrorState = ERROR_STATE()

        RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState)
    END CATCH
END
GO
```

6.4 Views/Vistas

As vistas SQL foram utilizadas como auxilio para as consultas da Base de Dados. No total foram criadas 4 vistas:

PratosCarneServidosIntervaloDatas - permite obter os pratos de carne servidos num intervalo de datas.

PratosDaEmentaHoje- permite obter a ementa de hoje e os pratos que nela figuram.

ProdutosEmentaAmanha – permite obter quais os produtos necessarios para a ementa do dia seguinte.

DataPratoPeixeCarne – permite obter quais as datas onde foi pedido o prato de carne juntamente com o de peixe.

6.4.1 View PratosCarneServidosIntervaloDatas

```
SELECT p.produtoID, p.produtoNome
FROM PedidoEmentaProduto pep, Produto p, Ementa e
WHERE pep.produtoID = p.produtoID
      AND e.ementaID = pep.ementaID
      AND e.dataEmenta BETWEEN '2022-06-14' AND '2022-06-18'
      AND p.tipoProdutoID = 'C'
GROUP BY p.produtoID, p.produtoNome
```

6.4.2 View PratosDaEmentaHoje

```
SELECT p.produtoNome
FROM EmentaProduto ep, Produto p, Ementa e
WHERE ep.produtoID = p.produtoID
      AND ep.ementaID = e.ementaID
      AND e.dataEmenta = CAST (GETDATE() AS Date)
      AND (p.tipoProdutoID = 'P' OR p.tipoProdutoID = 'C' OR p.tipoProdutoID = 'SO')
```

6.4.3 View ProdutosEmentaAmanha

```
SELECT p.produtoNome
FROM Produto p, EmentaProduto ep, Ementa e
WHERE p.produtoID = ep.produtoID
      AND ep.ementaID = e.ementaID
      AND e.dataEmenta = CAST(GETDATE() + 1 AS Date)
```

6.4.4 View DataPratoPeixeCarne

```
SELECT t3.dataPedido
FROM (SELECT t2.pedidoID, t2.dataPedido
        FROM (SELECT DISTINCT t1.pedidoID, p.dataPedido, t1.tipoProdutoID
                FROM (SELECT  pep.pedidoID, p.tipoProdutoID
                        FROM PedidoEmentaProduto pep, Produto p
                        WHERE pep.produtoID = p.produtoID AND (p.tipoProdutoID = 'P' OR p.tipoProdutoID = 'C')) t1, Pedido p
                WHERE t1.pedidoID = p.pedidoID) t2
        WHERE MONTH(t2.dataPedido) =  MONTH(CAST (GETDATE() AS date))
        GROUP BY t2.pedidoID, t2.dataPedido
        HAVING COUNT(t2.dataPedido) >= 2) t3
GROUP BY t3.dataPedido
```

7. Conclusões e Trabalho Futuro

A criação e manutenção de uma base de dados é algo demorado e complexo. Requer concentração e emepenho da parte de toda a equipa de desenvolvimento de forma a serem alcançados os melhores resultados.

O resultado esperado foi obtido, a base de dados para o restaurante “Michelin Restaurant” foi efetuada com sucesso.

No futuro, com mais experiência e conhecimento, é esperado melhorar esta base de dados de forma a facilitar o resultado pretendido.

Bibliografia

Connolly, T., & Begg, C. (s.d.). Database Systems A Pratical Approach to Design, Implementation and Management. Sixth Edition.

Referências WWW

[01] **www.youtube.com**

Recorremos aos tutorias fornecidos pelo professor no moodle.

[02] www.app.diagrams.net

Plataforma onde realizamos os diagramas.

Lista de Siglas e Acrónimos

BD	Base de Dados
UC	Unidade curricular
ER	Entidade-Relação
SQL	Structured Query Language
PK	Primary Key
FK	Foreign Key
NIF	Número de Identificação Fiscal

Anexos

ANEXO I

ANEXO I

1. Restaurant.bak
2. baseDados.drawio
3. Imagens