# Criando um banco de dados

- CREATE DATABASE;
- CREATE TABLE;
- Tipos de dados;
- Campo de autonumeração (IDENTITY);
- Constraints.

Esta Leitura Complementar refere-se ao conteúdo das Aulas 6 e 7.

Editora IMPACTA

# 1.1. Introdução

Nesta leitura, veremos os recursos iniciais para criação de banco de dados: os comandos **CREATE DATABASE** e **CREATE TABLE**, os tipos de dados e as constraints.

Quando formos mostrar as opções de sintaxe de um comando SQL, usaremos a seguinte nomenclatura:

- **[]**: Termos entre colchetes são opcionais;
- <>: Termos entre os sinais menor e maior são nomes ou valores definidos por nós;
- **{a1|a2|a3...}**: Lista de alternativas mutuamente exclusivas.

# 1.2. CREATE DATABASE

DATABASE é um conjunto de arquivos que armazena todos os objetos do banco de dados.

Para que um banco de dados seja criado no SQL Server, é necessário utilizar a instrução **CREATE DATABASE**, cuja sintaxe básica é a seguinte:

```
CREATE DATABASE <nome do banco de dados>
```

A seguir, veja um exemplo de criação de banco de dados:

```
CREATE DATABASE SALA_DE_AULA;
```

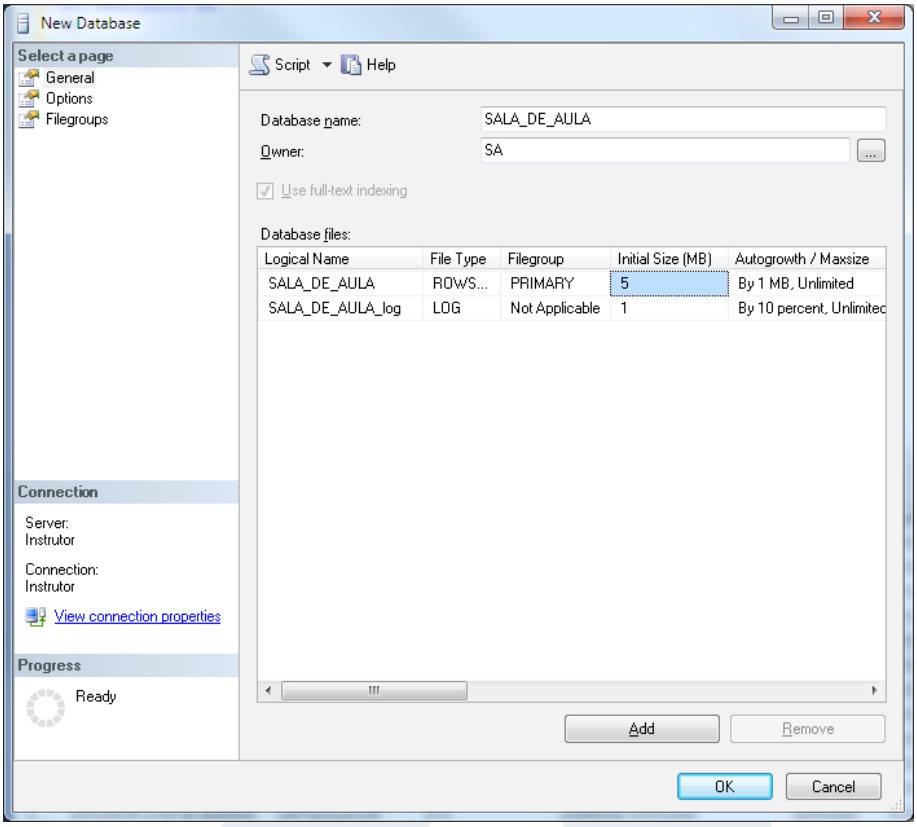
Essa instrução criará dois arquivos:

- **SALA_DE_AULA.MDF**: Armazena os dados;
- **SALA_DE_AULA_LOG.LDF**: Armazena os logs de transações.

Normalmente, esses arquivos estão localizados na pasta **DATA** dentro do diretório de instalação do SQL Server.

Assim que são criados, os bancos de dados possuem apenas os objetos de sistema, como tabelas, PROCEDURES, VIEWS, necessários para o gerenciamento das tabelas.

Também é possível criar um banco de dados graficamente. Através do SSMS, sobre a conexão, clique com o botão direito em **New Database**.




É importante que seja definida a localização dos arquivos que compõem o banco para o seu melhor gerenciamento.

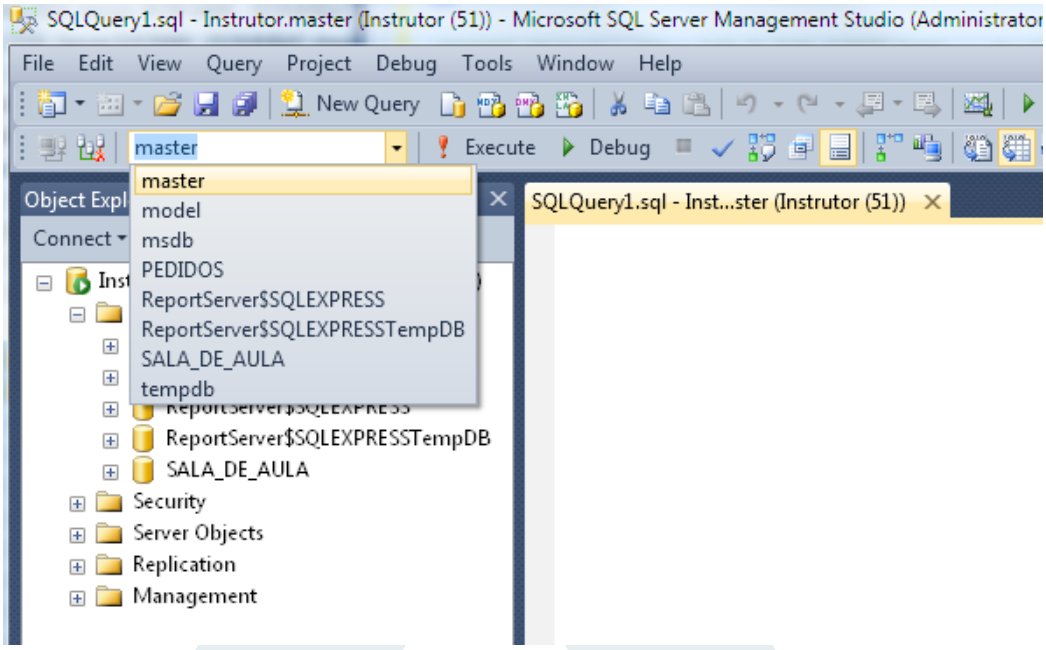
Para facilitar o acesso a um banco de dados, devemos colocá-lo em uso, mas isso não é obrigatório. Para colocar um banco de dados em uso, utilize o seguinte código:

```
USE <nome do banco de dados>
```

Veja um exemplo:

```
USE SALA_DE_AULA
```

Observe que na parte superior esquerda do SSMS existe um ComboBox que mostra o nome do banco de dados que está em uso.




# 1.3. CREATE TABLE

Os principais objetos de um banco de dados são suas tabelas, responsáveis pelo armazenamento dos dados.

A instrução **CREATE TABLE** deve ser utilizada para criar tabelas dentro de um banco de dados já existente. A sintaxe para uso dessa instrução é a seguinte:

```
CREATE TABLE <nome_tabela>
( <nome_campo1> <data_type> [IDENTITY [(<inicio>,<incremento>)]
                            [NOT NULL] [DEFAULT <exprDef>]
[, <nome_campo2> <data_type> [NOT NULL] [DEFAULT <exprDef>]
```

Em que:

- **<nome_tabela>**: Nome que vai identificar a tabela. A princípio, nomes devem começar por uma letra, seguida de letras, números e sublinhados. Porém, se o nome for escrito entre colchetes, poderá ter qualquer sequência de caracteres;

- **<nome_campo>**: Nome que vai identificar cada coluna ou campo da tabela. É criado utilizando a regra para os nomes das tabelas;

- **<data_type>**: Tipo de dado que será gravado na coluna (texto, número, data etc.);
- **[IDENTITY [(<inicio>,<incremento>)]:** Define um campo como autonumeração;
- **[NOT NULL]**: Define um campo que precisa ser preenchido, isto é, não pode ficar vazio (**NULL**);
- **[DEFAULT <exprDef>]**: Valor que será gravado no campo, caso ele fique vazio (**NULL**).

Com relação à sintaxe de **CREATE TABLE**, é importante considerar, ainda, as seguintes informações:

- A sintaxe descrita foi simplificada, há outras cláusulas na instrução **CREATE TABLE**;
- Uma tabela não pode conter mais de um campo **IDENTITY**;
- Uma tabela não pode conter mais de uma chave primária, mas pode ter uma chave primária composta por vários campos.

A seguir, veja um exemplo de como criar uma tabela em um banco de dados:

```
CREATE TABLE TB_ALUNO
(       COD_ALUNO               INT,
        NOME                    VARCHAR(30),
        DATA_NASCIMENTO         DATETIME,
        IDADE                   TINYINT,
        E_MAIL              VARCHAR(50),
        FONE_RES                CHAR(9),
        FONE_COM                CHAR(9),
        FAX                     CHAR(9),
        CELULAR                 CHAR(9),
        PROFISSAO               VARCHAR(40),
        EMPRESA                 VARCHAR(50) );
```

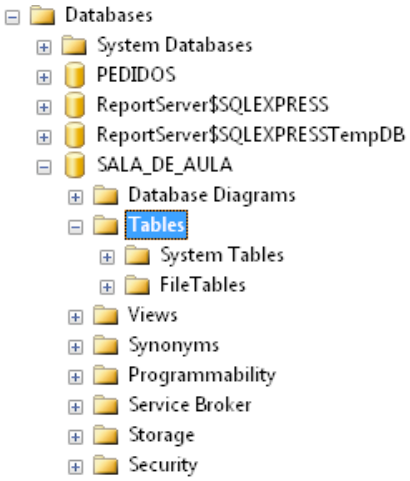
A estrutura dessa tabela não respeita as regras de normalização de dados.

O SQL disponibiliza a criação da tabela de forma gráfica. Vejamos:

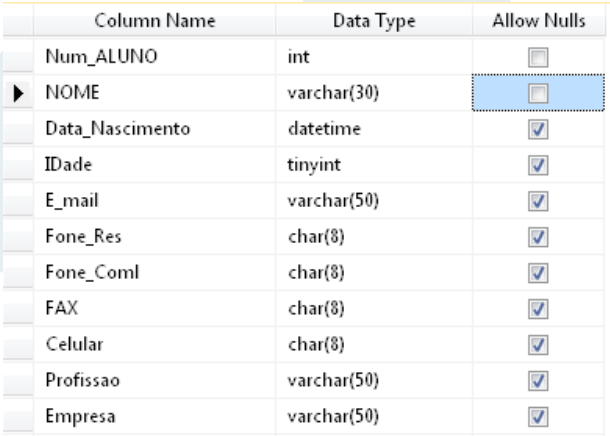
1. Expanda **Databases**;

2. Expanda o banco em que você deseja criar a tabela;

3. Clique com o botão direito em **Tables**;

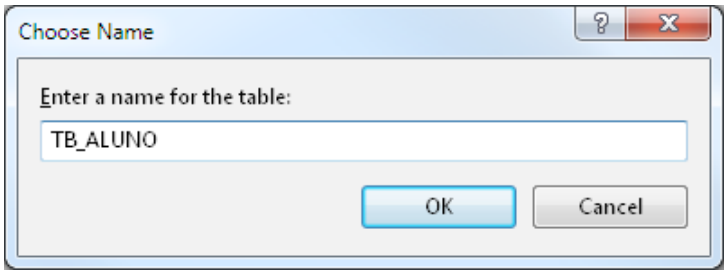



4. Informe os nomes dos campos;



| Column Name | Data Type | Allow Nulls |
|---|---|---|
| Num_ALUNO | int | ☐ |
| NOME | varchar(30) | ☐ |
| Data_Nascimento | datetime | ☑ |
| IDade | tinyint | ☑ |
| E_mail | varchar(50) | ☑ |
| Fone_Res | char(8) | ☑ |
| Fone_Coml | char(8) | ☑ |
| FAX | char(8) | ☑ |
| Celular | char(8) | ☑ |
| Profissao | varchar(50) | ☑ |
| Empresa | varchar(50) | ☑ |

5. No momento de salvar, informe o nome da tabela.



# 1.4. Tipos de dados

Cada elemento, como uma coluna, variável ou expressão, possui um tipo de dado. O tipo de dado especifica o tipo de valor que o objeto pode armazenar, como números inteiros, texto, data e hora etc. O SQL Server organiza os tipos de dados dividindo-os em categorias.

A seguir, serão descritas as principais categorias de tipos de dados utilizados na linguagem Transact-SQL.

## 1.4.1. Numéricos exatos

A tabela a seguir descreve alguns dos tipos de dados que fazem parte dessa categoria:

- **Inteiros**

| Nome | Descrição |
|---|---|
| **bigint**<br>**8 bytes** | Valor de número inteiro compreendido entre -2^63 (-9,223,372,036,854,775,808) e 2^63-1 (9,223,372,036,854,775,807). |
| **int**<br>**4 bytes** | Valor de número inteiro compreendido entre -2^31 (-2,147,483,648) e 2^31 - 1 (2,147,483,647). |
| **smallint**<br>**2 bytes** | Valor de número inteiro compreendido entre -2^15 (-32,768) e 2^15 - 1 (32,767). |
| **tinyint**<br>**1 byte** | Valor de número inteiro de 0 a 255. |

- **Bit**

| Nome | Descrição |
|---|---|
| **bit**<br>**1 byte** | Valor de número inteiro com o valor 1 ou o valor 0. |

- **Numéricos exatos**

| Nome | Descrição |
|---|---|
| **decimal(<T>,<D>)** | Valor numérico de precisão e escala fixas de -10^38 +1 até 10^38 –1. |
| **numeric(<T>,<D>)** | Valor numérico de precisão e escala fixas de -10^38 +1 até 10^38 –1. |

Nos numéricos exatos, é importante considerar as seguintes informações:

- <**T**>: Corresponde à quantidade máxima de algarismos que o número pode ter;
- <**D**>: Corresponde à quantidade máxima de casas decimais que o número pode ter;
- A quantidade de casas decimais <**D**> está contida na quantidade máxima de algarismos <**T**>;
- A quantidade de bytes ocupada varia dependendo de <**T**>.

- **Valores monetários**

| Nome | Descrição |
|---|---|
| **money**<br>**8 bytes** | Compreende valores monetários ou de moeda corrente entre -922.337.203.685.477,5808 e 922.337.203.685.477,5807. |
| **smallmoney**<br>**4 bytes** | Compreende valores monetários ou de moeda corrente entre -214,748.3648 e +214,748.3647. |

## 1.4.2. Numéricos aproximados

A tabela a seguir descreve alguns dos tipos de dados que fazem parte dessa categoria:

| Nome | Descrição |
|---|---|
| **float[(n)]** | Valor numérico de precisão flutuante entre -1.79E + 308 e -2.23E - 308, 0 e de 2.23E + 308 até 1.79E + 308. |
| **real**<br>o mesmo que<br>**float(24)** | Valor numérico de precisão flutuante entre -3.40E + 38 e -1.18E - 38, 0 e de 1.18E - 38 até 3.40E + 38. |

Em que:

- O valor de **n** determina a precisão do número. O padrão (default) é 53;

- Se **n** está entre 1 e 24, a precisão é de 7 algarismos e ocupa 4 bytes de memória. Com **n** entre 25 e 53, a precisão é de 15 algarismos e ocupa 8 bytes.

Esses tipos são chamados de "Numéricos aproximados" porque podem gerar imprecisão na parte decimal.

## 1.4.3. Data e hora

A tabela a seguir descreve alguns dos tipos de dados que fazem parte dessa categoria:

| Nome | Descrição |
|---|---|
| **datetime**<br>**8 bytes** | Data e hora compreendidas entre 1º de janeiro de 1753 e 31 de dezembro de 9999, com a exatidão de 3.33 milissegundos. |
| **smalldatetime**<br>**4 bytes** | Data e hora compreendidas entre 1º de janeiro de 1900 e 6 de junho de 2079, com a exatidão de 1 minuto. |
| **datetime2[(p)]**<br>**8 bytes** | Data e hora compreendidas entre 01/01/0001 e 31/12/9999 com precisão de até 100 nanossegundos, dependendo do valor de **p**, que representa a quantidade de algarismos na fração de segundo. Omitindo **p**, o valor default será 7. |
| **date**<br>**3 bytes** | Data compreendida entre 01/01/0001 e 31/12/9999, com precisão de 1 dia. |
| **time[(p)]**<br>**5 bytes** | Hora no intervalo de 00:00:00.0000000 a 23.59.59.9999999. O parâmetro **p** indica a quantidade de dígitos na fração de segundo. |
| **Datetimeoffset[(p)]** | Data e hora compreendidas entre 01/01/0001 e 31/12/9999 com precisão de até 100 nanossegundos e com indicação do fuso horário, cujo intervalo pode variar de -14:00 a +14:00. O parâmetro **p** indica a quantidade de dígitos na fração de segundo. |

## 1.4.4. Strings de caracteres ANSI

É chamada de **string** uma sequência de caracteres. No padrão ANSI, cada caractere é armazenado em 1 byte, o que permite a codificação de até 256 caracteres.

A tabela a seguir descreve alguns dos tipos de dados que fazem parte dessa categoria:

| Nome | Descrição |
|---|---|
| **char(<n>)** | Comprimento fixo de no máximo 8.000 caracteres no padrão ANSI. Cada caractere é armazenado em 1 byte. |
| **varchar(<n>)** | Comprimento variável de no máximo 8.000 caracteres no padrão ANSI. Cada caractere é armazenado em 1 byte. |
| **text** ou<br>**varchar(max)** | Comprimento variável de no máximo 2^31 - 1 (2,147,483,647) caracteres no padrão ANSI. Cada caractere é armazenado em 1 byte. |

Em que:

- **<n>**: Representa a quantidade máxima de caracteres que poderemos armazenar. Cada caractere ocupa 1 byte.

É recomendável a utilização do tipo **varchar(max)** em vez do tipo **text**. Esse último será removido em versões futuras do SQL Server. No caso de aplicações que já o utilizam, é indicado realizar a substituição pelo tipo recomendado. Ao utilizarmos **max** para **varchar**, estamos ampliando sua capacidade de armazenamento para 2 GB, aproximadamente.

## 1.4.5. Strings de caracteres Unicode

Em strings Unicode, cada caractere é armazenado em 2 bytes, o que amplia a quantidade de caracteres possíveis para mais de 65.000.

A tabela a seguir descreve alguns dos tipos de dados que fazem parte dessa categoria:

| Nome | Descrição |
|---|---|
| **nchar(<n>)** | Comprimento fixo de no máximo 4.000 caracteres Unicode. |
| **nvarchar(<n>)** | Comprimento variável de no máximo 4.000 caracteres Unicode. |
| **ntext** ou **nvarchar(max)** | Comprimento variável de no máximo 2^30 - 1 (1,073,741,823) caracteres Unicode. |

Em que:

- **<n>**: Representa a quantidade máxima de caracteres que poderemos armazenar. Cada caractere ocupa 2 bytes. Essa quantidade de 2 bytes é destinada a países cuja quantidade de caracteres utilizados é muito grande, como Japão e China.

Tanto no padrão ANSI quanto no UNICODE, existe uma tabela (ASCII) que codifica todos os caracteres. Essa tabela é usada para converter o caractere no seu código, quando gravamos, e para converter o código no caractere, quando lemos.

## 1.4.6. Strings binárias

No caso das strings binárias, não existe uma tabela para converter os caracteres, você interpreta os bits de cada byte de acordo com uma regra sua.

A tabela a seguir descreve alguns dos tipos de dados que fazem parte dessa categoria:

| Nome | Descrição |
|---|---|
| **binary(<n>)** | Dado binário com comprimento fixo de, no máximo, 8.000 bytes. |
| **varbinary(<n>)** | Dado binário com comprimento variável de, no máximo, 8.000 bytes. |
| **image** ou **varbinary(max)** | Dado binário com comprimento variável de, no máximo, 2^31 - 1 (2,147,483,647) bytes. |

Os tipos **image** e **varbinary(max)** são muito usados para importar arquivos binários para dentro do banco de dados. Imagens, sons ou qualquer outro tipo de documento podem ser gravados em um campo desses tipos. É recomendável a utilização do tipo **varbinary(max)** em vez do tipo **image**. Esse último será removido em versões futuras do SQL Server. No caso de aplicações que já o utilizam, é indicado realizar a substituição pelo tipo recomendado.

## 1.4.7. Outros tipos de dados

Essa categoria inclui tipos de dados especiais, cuja utilização é específica e restrita a certas situações. A tabela adiante descreve alguns desses tipos:

| Nome | Descrição |
|---|---|
| **table** | Serve para definir um dado tabular, composto de linhas e colunas, assim como uma tabela. |
| **cursor** | Serve para percorrer as linhas de um dado tabular. |
| **sql_variant** | Um tipo de dado que armazena valores de vários tipos suportados pelo SQL Server, exceto os seguintes: **text**, **ntext**, **timestamp** e **sql_variant**. |
| **Timestamp** ou **RowVersion** | Número hexadecimal sequencial gerado automaticamente. |
| **uniqueidentifier** | Globally Unique Identifier (GUID), também conhecido como Identificador Único Global ou Identificador Único Universal. É um número hexadecimal de 16 bytes semelhante a 64261228-50A9-467C-85C5-D73C51A914F1. |

| Nome | Descrição |
|---|---|
| **XML** | Armazena dados no formato XML. |
| **Hierarchyid** | Posição de uma hierarquia. |
| **Geography** | Representa dados de coordenadas terrestres. |
| **Geometry** | Representação de coordenadas euclidianas. |

## 1.5. Campo de autonumeração (IDENTITY)

A coluna de identidade, ou campo de autonumeração, é definida pela propriedade **IDENTITY**. Ao atribuirmos essa propriedade a uma coluna, o SQL Server cria números em sequência para linhas que forem posteriormente inseridas na tabela em que a coluna de identidade está localizada.

É importante saber que uma tabela pode ter apenas uma coluna do tipo identidade e que não é possível inserir ou alterar seu valor, que é gerado automaticamente pelo SQL-Server. Veja um exemplo:

```
DROP TABLE TB_ALUNO;        --Caso a tabela já exista
GO
CREATE TABLE TB_ALUNO
(       NUM_ALUNO               INT IDENTITY,
        NOME                    VARCHAR(30),
        DATA_NASCIMENTO         DATETIME,
        IDADE                   TINYINT,
        E_MAIL              VARCHAR(50),
        FONE_RES                CHAR(8),
        FONE_COM                CHAR(8),
        FAX                     CHAR(8),
        CELULAR                 CHAR(9),
      PROFISSAO                 VARCHAR(40),
      EMPRESA               VARCHAR(50) );
```

## 1.6. Constraints

As constraints são objetos utilizados para impor restrições e aplicar validações aos campos de uma tabela ou à forma como duas tabelas se relacionam. Podem ser definidas no momento da criação da tabela ou posteriormente, utilizando o comando **ALTER TABLE**.

## 1.6.1. Nulabilidade

Além dos valores padrão, também é possível atribuir valores nulos a uma coluna, o que significa que ela não terá valor algum. O **NULL** (nulo) não corresponde a nenhum dado, não é vazio ou zero, é nulo. Ao criarmos uma coluna em uma tabela, podemos acrescentar o atributo **NULL** (que já é padrão), para que aceite valores nulos, ou então **NOT NULL**, quando não queremos que determinada coluna aceite valores nulos. Exemplo:

```
CREATE TABLE TB_ALUNO
(       CODIGO      INT              NOT NULL,
        NOME              VARCHAR(30) NOT NULL,
        E_MAIL      VARCHAR(100)      NULL  );
```

## 1.6.2. Tipos de constraints

São diversos os tipos de constraints que podem ser criados: **PRIMARY KEY**, **UNIQUE**, **CHECK**, **DEFAULT** e **FOREIGN KEY**. Adiante, cada uma das constraints será descrita.

### 1.6.2.1. PRIMARY KEY (chave primária)

A chave primária identifica, de forma única, cada uma das linhas de uma tabela. Pode ser formada por apenas uma coluna ou pela combinação de duas ou mais colunas. A informação definida como chave primária de uma tabela não pode ser duplicada dentro dessa tabela.

- **Exemplos**

| | |
|---|---|
| Tabela CLIENTES | Código do Cliente |
| Tabela PRODUTOS | Código do Produto |

As colunas que formam a chave primária não podem aceitar valores nulos e devem ter o atributo **NOT NULL**.

- **Comando**

```
CREATE TABLE tabela
(
        CAMPO_PK                tipo  NOT NULL,
        ...,
        ...,
        CONSTRAINT NomeChavePrimária PRIMARY KEY (CAMPO_PK)   )
```

Onde usamos o termo **CAMPO_PK** na criação da **PRIMARY KEY**, também poderá ser uma combinação de campos separados por vírgula.

O comando será o seguinte, depois de criada a tabela:

```
ALTER TABLE tabela ADD
CONSTRAINT NomeChavePrimária PRIMARY KEY (CAMPO_PK)
```

A convenção para dar nome a uma constraint do tipo chave primária é **PK_NomeTabela**, ou seja, **PK** (abreviação de **PRIMARY KEY**) seguido do nome da tabela para a qual estamos criando a chave primária.

## 1.6.2.2. UNIQUE

Além do(s) campo(s) que forma(m) a **PRIMARY KEY**, pode ocorrer de termos outras colunas que não possam aceitar dados em duplicidade. Nesse caso, usaremos a constraint **UNIQUE**.

As colunas nas quais são definidas constraints **UNIQUE** permitem a inclusão de valores nulos, desde que seja apenas um valor nulo por coluna.

- **Exemplos**

| | |
|---|---|
| Na tabela CLIENTES | Campos CPF, RG e E-mail |
| Na tabela TIPOS_PRODUTO | Descrição do tipo de produto |
| Na tabela UNIDADES_MEDIDA | Descrição da unidade de medida |

- **Comando**

```
CREATE TABLE tabela
(
        ...
        CAMPO_UNICO             tipo  NOT NULL,
        ...,
        ...,
        CONSTRAINT NomeUnique UNIQUE (CAMPO_UNICO)      )
```

Onde usamos o termo **CAMPO_UNICO** na criação da **UNIQUE**, também poderá ser uma combinação de campos separados por vírgula.

O comando será o seguinte, depois de criada a tabela:

```
ALTER TABLE tabela ADD
CONSTRAINT NomeUnique UNIQUE (CAMPO_UNICO)
```

O nome da constraint deve ser sugestivo, como **UQ_NomeTabela_NomeCampoUnico**.

## 1.6.2.3. CHECK

Nesse tipo de constraint, criamos uma condição (semelhante às usadas com a cláusula **WHERE**) para definir a integridade de um ou mais campos de uma tabela.

- **Exemplo**

| | |
|---|---|
| Tabela CLIENTES | Data Nascimento < Data Atual<br>Data Inclusão <= Data Atual<br>Data Nascimento < Data Inclusão |
| Tabela PRODUTOS | Preço Venda >= 0<br>Preço Compra >= 0<br>Preço Venda >= Preço Compra<br>Data Inclusão <= Data Atual |

- **Comando**

```
CREATE TABLE tabela
(
        ...,
        ...,
        CONSTRAINT NomeCheck CHECK (Condição)     )
```

O comando será o seguinte, depois de criada a tabela:

```
ALTER TABLE tabela ADD
CONSTRAINT NomeCheck CHECK (Condição)
```

O nome da constraint deve ser sugestivo, como **CH_NomeTabela_DescrCondicao**.

## 1.6.2.4. DEFAULT

Normalmente, quando inserimos dados em uma tabela, as colunas para as quais não fornecemos valor terão, como conteúdo, **NULL**. Ao definirmos uma constraint do tipo **DEFAULT** para uma determinada coluna, este valor será atribuído a ela quando o **INSERT** não fornecer valor.

- **Exemplo**

| | |
|---|---|
| Tabela PESSOAS | Data Inclusão DEFAULT Data Atual<br>Sexo DEFAULT 'M' |

- **Comando**

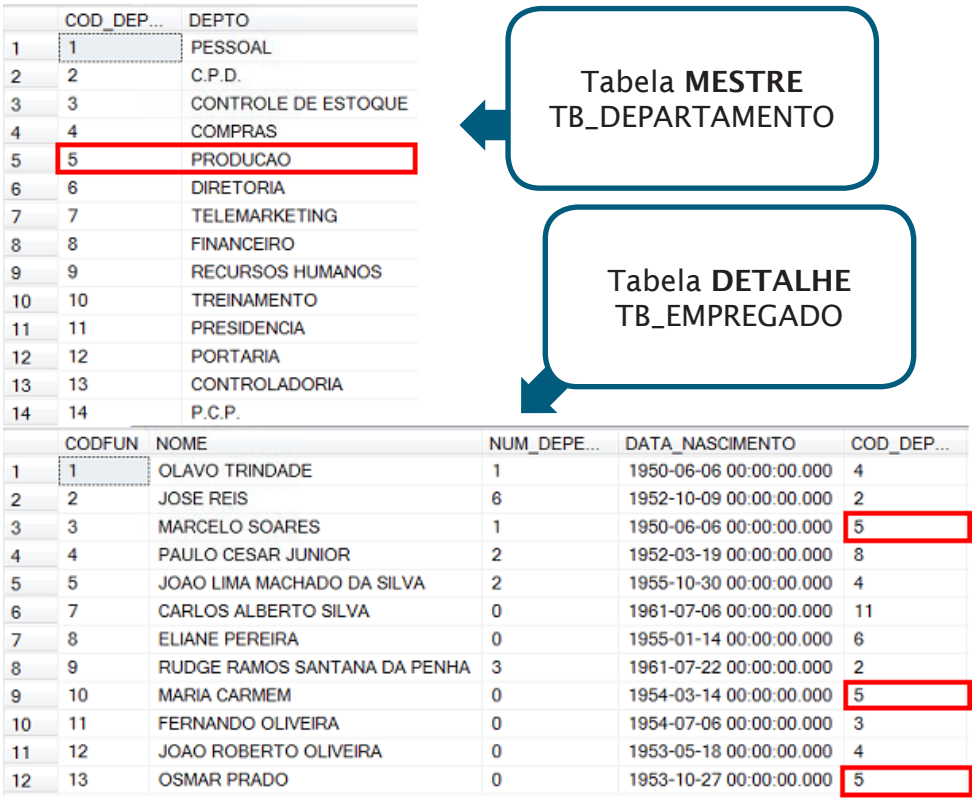
```
CREATE TABLE tabela
(
        ...,
        CAMPO_DEFAULT     tipo  [NOT NULL] DEFAULT valorDefault,
        ...,
)
```

O comando será o seguinte, depois de criada a tabela:

```
ALTER TABLE tabela ADD
CONSTRAINT NomeDefault DEFAULT (valorDefault) FOR CAMPO_DEFAULT
```

## 1.6.2.5. FOREIGN KEY (chave estrangeira)

É chamado de chave estrangeira ou **FOREIGN KEY** o campo da tabela **DETALHE** que se relaciona com a chave primária da tabela **MESTRE**.




| | COD_DEP... | DEPTO |
|---|---|---|
| 1 | 1 | PESSOAL |
| 2 | 2 | C.P.D. |
| 3 | 3 | CONTROLE DE ESTOQUE |
| 4 | 4 | COMPRAS |
| 5 | 5 | PRODUCAO |
| 6 | 6 | DIRETORIA |
| 7 | 7 | TELEMARKETING |
| 8 | 8 | FINANCEIRO |
| 9 | 9 | RECURSOS HUMANOS |
| 10 | 10 | TREINAMENTO |
| 11 | 11 | PRESIDENCIA |
| 12 | 12 | PORTARIA |
| 13 | 13 | CONTROLADORIA |
| 14 | 14 | P.C.P. |

| | CODFUN | NOME | NUM_DEPE... | DATA_NASCIMENTO | COD_DEP... |
|---|---|---|---|---|---|
| 1 | 1 | OLAVO TRINDADE | 1 | 1950-06-06 00:00:00.000 | 4 |
| 2 | 2 | JOSE REIS | 6 | 1952-10-09 00:00:00.000 | 2 |
| 3 | 3 | MARCELO SOARES | 1 | 1950-06-06 00:00:00.000 | 5 |
| 4 | 4 | PAULO CESAR JUNIOR | 2 | 1952-03-19 00:00:00.000 | 8 |
| 5 | 5 | JOAO LIMA MACHADO DA SILVA | 2 | 1955-10-30 00:00:00.000 | 4 |
| 6 | 7 | CARLOS ALBERTO SILVA | 0 | 1961-07-06 00:00:00.000 | 11 |
| 7 | 8 | ELIANE PEREIRA | 0 | 1955-01-14 00:00:00.000 | 6 |
| 8 | 9 | RUDGE RAMOS SANTANA DA PENHA | 3 | 1961-07-22 00:00:00.000 | 2 |
| 9 | 10 | MARIA CARMEM | 0 | 1954-03-14 00:00:00.000 | 5 |
| 10 | 11 | FERNANDO OLIVEIRA | 0 | 1954-07-06 00:00:00.000 | 3 |
| 11 | 12 | JOAO ROBERTO OLIVEIRA | 0 | 1953-05-18 00:00:00.000 | 4 |
| 12 | 13 | OSMAR PRADO | 0 | 1953-10-27 00:00:00.000 | 5 |

Observando a figura, notamos que o campo **COD_DEPTO** da tabela **TB_EMPREGADO** (detalhe) é chave estrangeira, pois se relaciona com **COD_DEPTO** (chave primária) da tabela **TB_DEPARTAMENTO** (mestre).

Podemos ter essa mesma estrutura sem termos criado uma chave estrangeira, embora a chave estrangeira garanta a integridade referencial dos dados, ou seja, com a chave estrangeira, será impossível existir um registro de **TB_EMPREGADO** com um **COD_DEPTO** inexistente em **TB_DEPARTAMENTO**. Quando procurarmos o **COD_DEPTO** do empregado em **TB_DEPARTAMENTO**, sempre encontraremos correspondência.

Se a tabela **MESTRE** não possuir uma chave primária, não será possível criar uma chave estrangeira apontando para ela.

Para criarmos uma chave estrangeira, temos que considerar as seguintes informações envolvidas:

- A tabela **DETALHE**;
- O campo da tabela **DETALHE** que se relaciona com a tabela **MESTRE**;
- A tabela **MESTRE**;
- O campo chave primária da tabela mestre.

O comando para a criação de uma chave estrangeira é o seguinte:

```
CREATE TABLE tabelaDetalhe
(
        ...,
        CAMPO_FK    tipo  [NOT NULL],
        ...,
        CONSTRAINT NomeChaveEstrangeira FOREIGN KEY(CAMPO_FK)
        REFERENCES tabelaMestre(CAMPO_PK_TABELA_MESTRE)
)
```

Ou utilize o seguinte comando depois de criada a tabela:

```
ALTER TABLE tabelaDetalhe ADD
        CONSTRAINT NomeChaveEstrangeira FOREIGN KEY(CAMPO_FK)
        REFERENCES tabelaMestre(CAMPO_PK_TABELA_MESTRE)
```

## 1.6.3. Criando constraints

A seguir, veremos como criar constraints com o uso de **CREATE TABLE** e **ALTER TABLE**, bem como criá-las graficamente a partir da interface do SQL Server Management Studio.

### 1.6.3.1. Criando constraints com CREATE TABLE

A seguir, temos um exemplo de criação de constraints com o uso de **CREATE TABLE**:

1. Primeiramente, crie o banco de dados **TESTE_CONSTRAINT**:

```
CREATE DATABASE TESTE_CONSTRAINT;
GO
USE TESTE_CONSTRAINT;
```

2. Agora, crie a tabela **TB_TIPO_PRODUTO** com os tipos (categorias) de produto:

```
-- Tabela de tipos (categorias) de produto
CREATE TABLE TB_TIPO_PRODUTO
(       COD_TIPO                INT IDENTITY NOT NULL,
        TIPO                    VARCHAR(30) NOT NULL,
    -- Convenção de nome: PK_NomeTabela
    CONSTRAINT PK_TB_TIPO_PRODUTO PRIMARY KEY (COD_TIPO),
    -- Convenção De nome: UQ_NomeTabela_NomeCampo
    CONSTRAINT UQ_TB_TIPO_PRODUTO_TIPO UNIQUE( TIPO ) );
```

3. Em seguida, teste a constraint **UNIQUE** criada:

```
-- Testando a constraint UNIQUE
INSERT TB_TIPO_PRODUTO VALUES ('MOUSE');
INSERT TB_TIPO_PRODUTO VALUES ('PEN-DRIVE');
INSERT TB_TIPO_PRODUTO VALUES ('HARD DISK');
-- Ao tentar inserir, o SQL gera um erro de violação de
-- constraint UNIQUE
INSERT TB_TIPO_PRODUTO VALUES ('HARD DISK');
```

4. O próximo passo é criar a tabela de produtos (**TB_PRODUTO**):

```
-- Tabela de TB_PRODUTO
CREATE TABLE TB_PRODUTO
(   ID_PRODUTO              INT IDENTITY NOT NULL,
    DESCRICAO               VARCHAR(50),
    COD_TIPO                INT,
    PRECO_CUSTO                   NUMERIC(10,2),
    PRECO_VENDA                   NUMERIC(10,2),
    QTD_REAL                NUMERIC(10,2),
    QTD_MINIMA              NUMERIC(10,2),
    DATA_CADASTRO           DATETIME DEFAULT GETDATE(),
    SN_ATIVO                CHAR(1) DEFAULT 'S',
    CONSTRAINT PK_TB_PRODUTO PRIMARY KEY( ID_PRODUTO ),
    CONSTRAINT UQ_TB_PRODUTO_DESCRICAO UNIQUE( DESCRICAO ),

    CONSTRAINT CK_TB_PRODUTO_PRECOS
               CHECK( PRECO_VENDA >= PRECO_CUSTO ),
    CONSTRAINT CK_TB_PRODUTO_DATA_CAD
               CHECK( DATA_CADASTRO <= GETDATE() ),
    CONSTRAINT CK_TB_PRODUTO_SN_ATIVO
               CHECK( SN_ATIVO IN ('N','S') ),
    -- Convenção de nome: FK_TabelaDetalhe_TabelaMestre
    CONSTRAINT FK_TB_PRODUTO_TIPO_PRODUTO
               FOREIGN KEY (COD_TIPO)
               REFERENCES TB_TIPO_PRODUTO (COD_TIPO)    );
```

5. Veja um modelo para a criação de chave estrangeira:

```
-- Criação de chave estrangeira

--   CONSTRAINT FK_TabelaDetalhe_TabelaMestre
--         FOREIGN KEY (campoTabelaDetalhe)
--         REFERENCES TabelaMestre( campoPK_TabelaMestre )
```

6. Feito isso, teste a constraint **DEFAULT** criada anteriormente. A sequência de código adiante gera os valores para os campos **DATA_CADASTRO** e **SN_ATIVO** que não foram mencionados no **INSERT**:

```
INSERT TB_PRODUTO
(DESCRICAO, COD_TIPO, PRECO_CUSTO, PRECO_VENDA,
 QTD_REAL, QTD_MINIMA)
VALUES ('TESTANDO INCLUSAO', 1, 10, 12, 10, 5 );
--
SELECT * FROM TB_PRODUTO;
```

7. No código seguinte, teste a constraint **UNIQUE**:

```
-- Gera erro, pois viola a constraint UNIQUE
INSERT TB_PRODUTO
(DESCRICAO, COD_TIPO, PRECO_CUSTO, PRECO_VENDA,
 QTD_REAL, QTD_MINIMA)
VALUES ('TESTANDO INCLUSAO', 10, 10, 12, 10, 5 );
```

8. No próximo código, teste a constraint **FOREIGN KEY**:

```
-- Gera um erro, pois viola a constraint FOREIGN KEY
INSERT TB_PRODUTO
(DESCRICAO, COD_TIPO, PRECO_CUSTO, PRECO_VENDA,
 QTD_REAL, QTD_MINIMA)
VALUES ('TESTANDO INCLUSAO 2', 10, 10, 12, 10, 5 );
```

9. Por fim, o código adiante testa a constraint **CHECK**:

```
-- Gera um erro, pois viola a constraint CHECK -
-- (CK_PRODUTO_PRECOS)
INSERT TB_PRODUTO
(DESCRICAO, COD_TIPO, PRECO_CUSTO, PRECO_VENDA,
 QTD_REAL, QTD_MINIMA)
VALUES ('TESTANDO INCLUSAO 2', 1, 14, 12, 10, 5 );
```

## 1.6.3.2. Criando constraints com ALTER TABLE

O exemplo a seguir demonstra a criação de constraints utilizando **ALTER TABLE**:

```
USE TESTE_CONSTRAINT;
```

```
DROP TABLE TB_PRODUTO
GO
DROP TABLE TB_TIPO_PRODUTO
GO
-- Criação da tabela TIPO_PRODUTO
CREATE TABLE TB_TIPO_PRODUTO
(       COD_TIPO                INT IDENTITY NOT NULL,
        TIPO                    VARCHAR(30) NOT NULL );
-- Criando as constraints com ALTER TABLE
ALTER TABLE TB_TIPO_PRODUTO ADD
    CONSTRAINT PK_TIPO_PRODUTO PRIMARY KEY (COD_TIPO);

ALTER TABLE TB_TIPO_PRODUTO ADD
    CONSTRAINT UQ_TIPO_PRODUTO_TIPO UNIQUE( TIPO );
```

```
-- Criando a tabela PRODUTOS
CREATE TABLE TB_PRODUTO
(   ID_PRODUTO              INT IDENTITY NOT NULL,
    DESCRICAO               VARCHAR(50),
    COD_TIPO                INT,
    PRECO_CUSTO                   NUMERIC(10,2),
    PRECO_VENDA                   NUMERIC(10,2),
    QTD_REAL                NUMERIC(10,2),
    QTD_MINIMA              NUMERIC(10,2),
    DATA_CADASTRO           DATETIME,
    SN_ATIVO                CHAR(1) );
```

```
-- Criando as constraints com ALTER TABLE
ALTER TABLE TB_PRODUTO ADD
    CONSTRAINT PK_TB_PRODUTO PRIMARY KEY( ID_PRODUTO );
```

```
ALTER TABLE TB_PRODUTO ADD
    CONSTRAINT UQ_TB_PRODUTO_DESCRICAO UNIQUE( DESCRICAO );
```

```
-- Criando várias constraints em um único ALTER TABLE
ALTER TABLE TB_PRODUTO ADD
    CONSTRAINT CK_TB_PRODUTO_PRECOS
               CHECK( PRECO_VENDA >= PRECO_CUSTO ),
    CONSTRAINT CK_TB_PRODUTO_DATA_CAD
               CHECK( DATA_CADASTRO <= GETDATE() ),
    CONSTRAINT CK_TB_PRODUTO_SN_ATIVO
               CHECK( SN_ATIVO IN ('N','S') ),
    CONSTRAINT FK_TB_PRODUTO_TIPO_PRODUTO
               FOREIGN KEY (COD_TIPO)
               REFERENCES TB_TIPO_PRODUTO (COD_TIPO),
   CONSTRAINT DF_TB_PRODUTO_SN_ATIVO DEFAULT ('S') FOR SN_ATIVO,
   CONSTRAINT DF_TB_PRODUTO_DATA_CADASTRO DEFAULT (GETDATE())
                              FOR DATA_CADASTRO;
```

## 1.6.3.3. Criando constraints graficamente

O exemplo a seguir demonstra a criação de constraints graficamente. Primeiramente, crie as tabelas **TIPO_PRODUTO** e **PRODUTOS** no banco de dados **TESTE_CONSTRAINT**:

```
USE TESTE_CONSTRAINT;
```

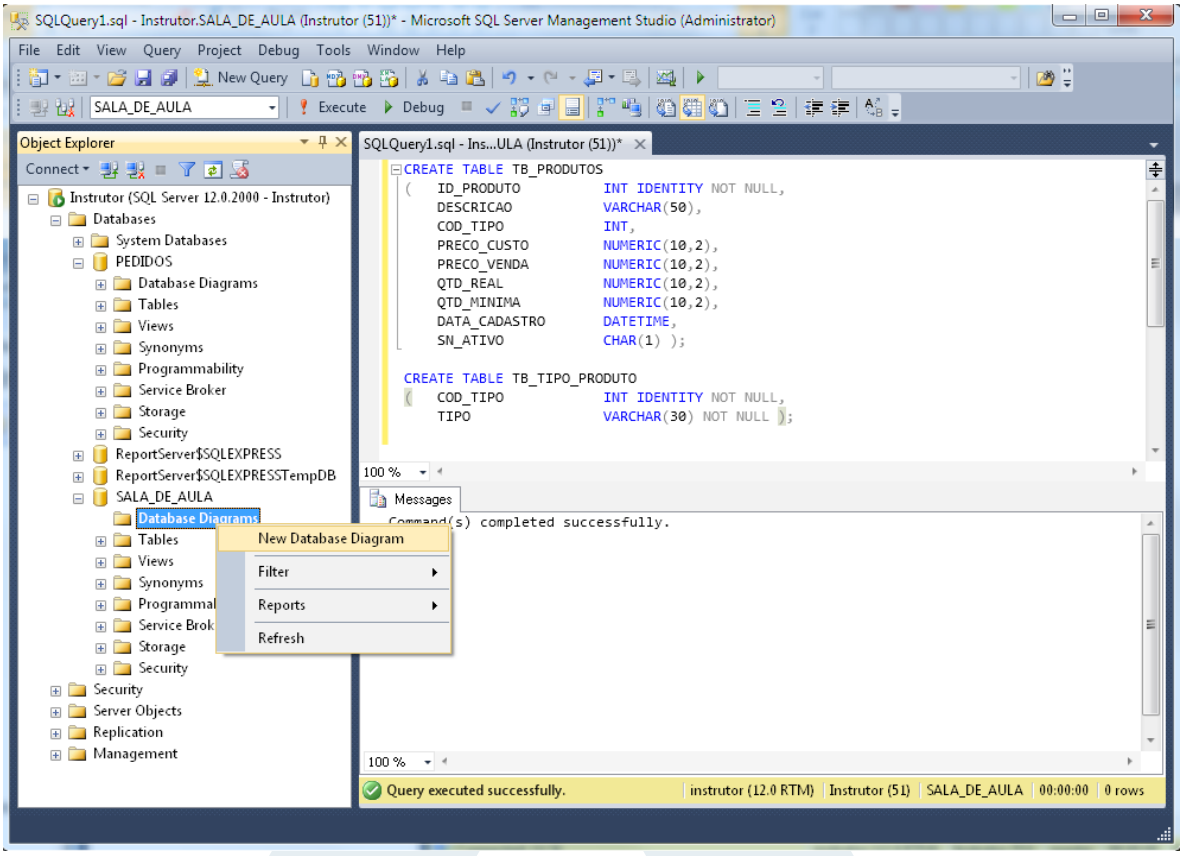
```
DROP TABLE TB_PRODUTO
GO
DROP TABLE TB_TIPO_PRODUTO
GO
-- Criação da tabela TIPO_PRODUTO
CREATE TABLE TB_TIPO_PRODUTO
(       COD_TIPO                INT IDENTITY NOT NULL,
        TIPO                    VARCHAR(30) NOT NULL );
```

```
-- Criando a tabela PRODUTO
CREATE TABLE TB_PRODUTOS
(   ID_PRODUTO              INT IDENTITY NOT NULL,
    DESCRICAO               VARCHAR(50),
    COD_TIPO                INT,
    PRECO_CUSTO                   NUMERIC(10,2),
    PRECO_VENDA                   NUMERIC(10,2),
    QTD_REAL                NUMERIC(10,2),
    QTD_MINIMA              NUMERIC(10,2),
    DATA_CADASTRO           DATETIME,
    SN_ATIVO                CHAR(1) );
```

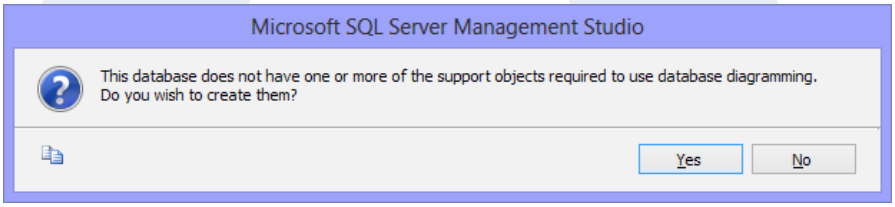
Depois, realize os seguintes passos:

1. No Object Explorer, selecione o banco de dados **TESTE_CONSTRAINT** e abra os subitens do banco;

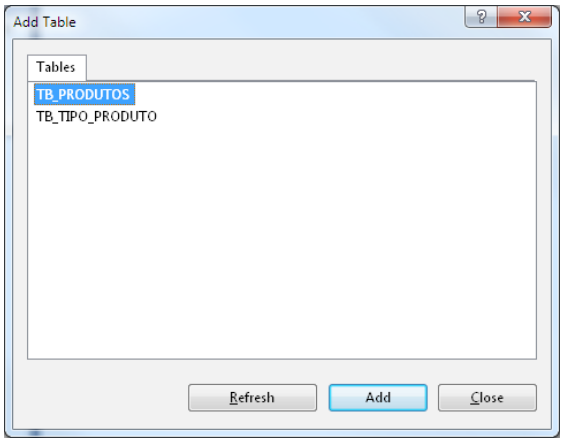
2. Clique com o botão direito do mouse no item **Database Diagrams** e selecione a opção **New Database Diagram**;

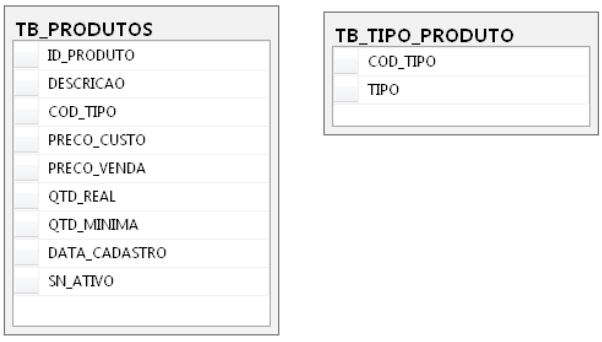



3. Na caixa de diálogo exibida, clique em **Yes** (**Sim**);

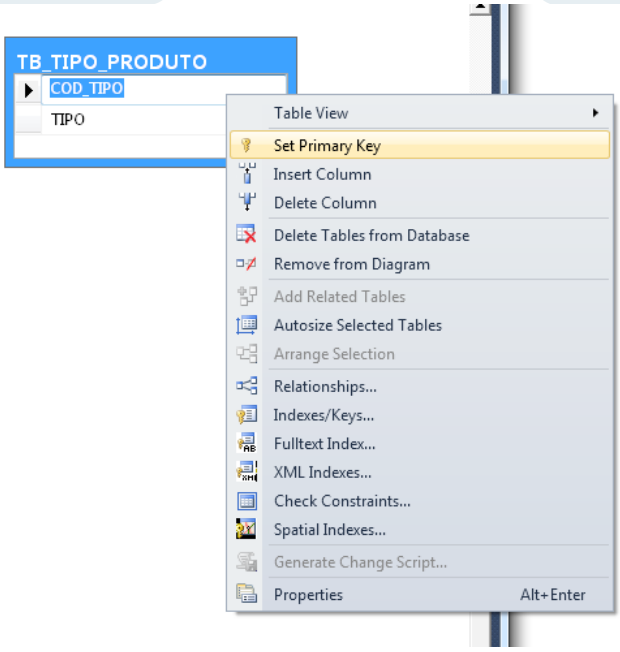


4. Uma janela com os nomes das tabelas existentes no banco de dados será exibida. Selecione as duas tabelas, clique em **Add** (**Adicionar**) e, depois, em **Close** (**Fechar**). Note que as tabelas aparecerão na área de desenho do diagrama;

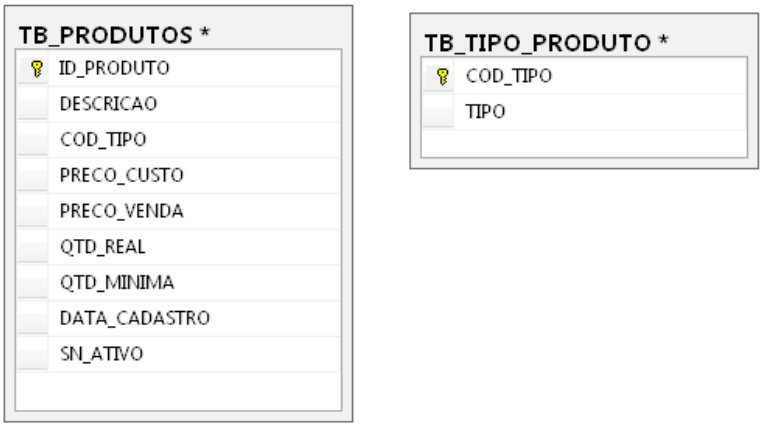


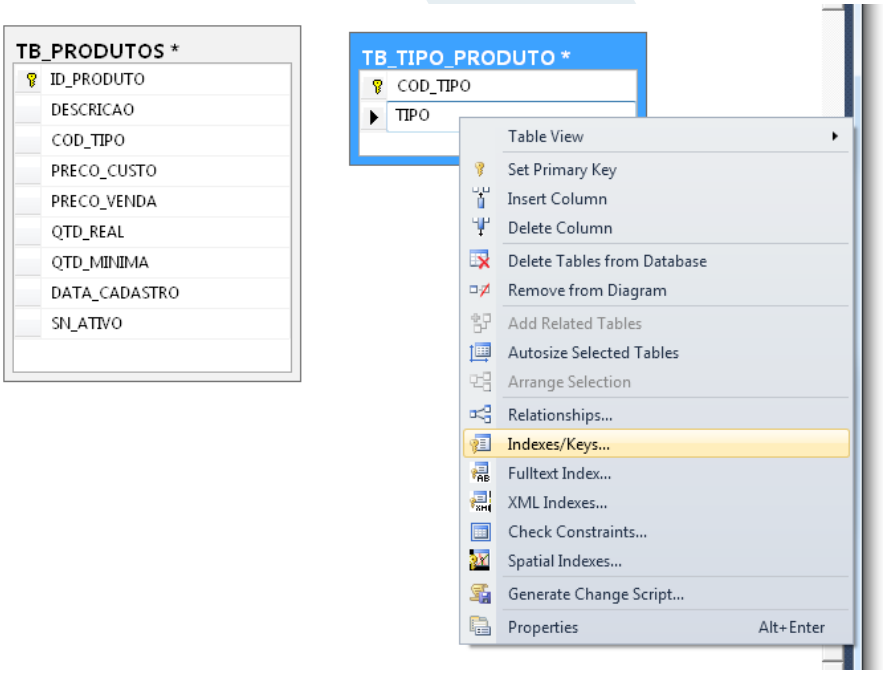

5. Clique com o botão direito do mouse no campo **COD_TIPO** da tabela **TB_TIPO_PRODUTO** e selecione a opção **Set Primary Key** (**Definir Chave Primária**);



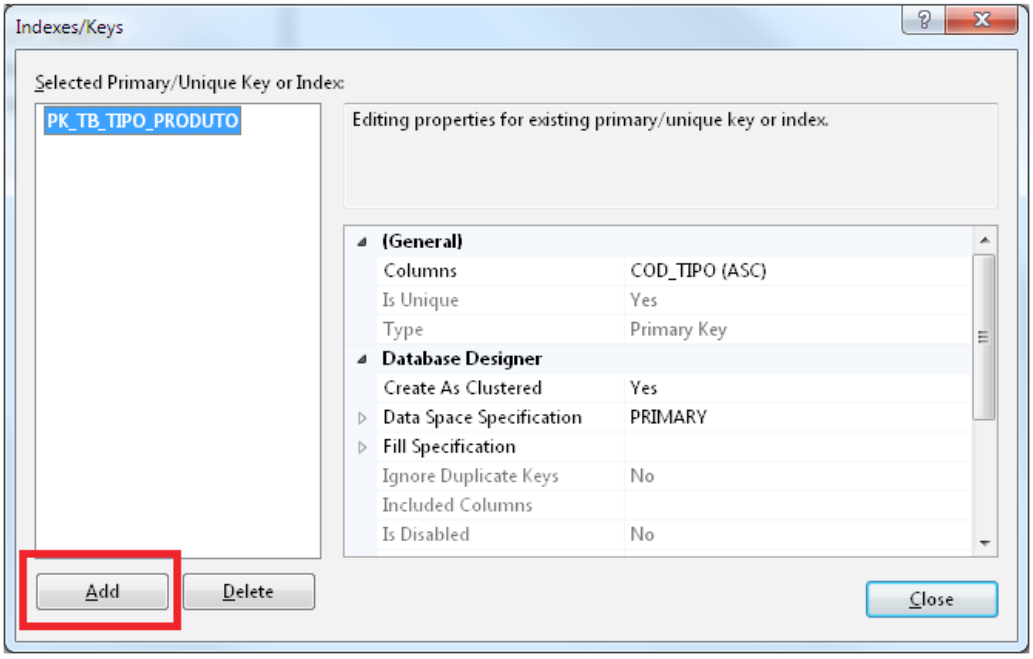
6. Clique com o botão direito do mouse no campo **ID_PRODUTO** da tabela **TB_PRODUTO** e selecione a opção **Set Primary Key** (**Definir Chave Primária**). Com isso, serão criadas as chaves primárias das duas tabelas;



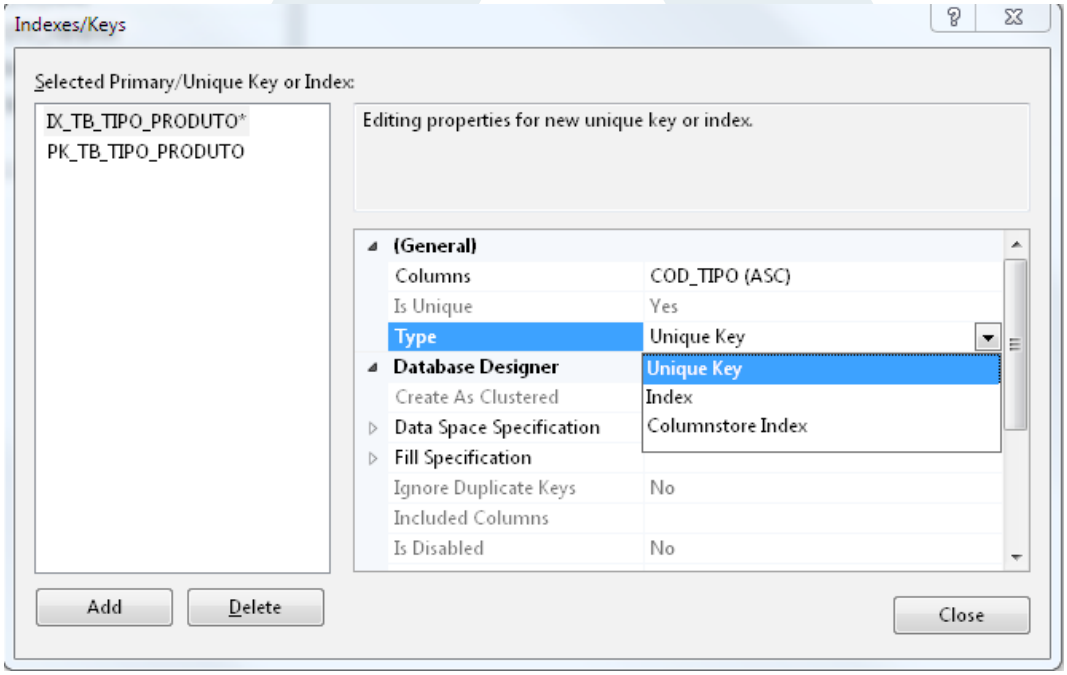

7. Para criar a chave única (**UNIQUE CONSTRAINT**), selecione a tabela **TB_TIPO_PRODUTO**, clique com o botão direito do mouse sobre ela e clique na opção **Indexes/Keys...** (Índices/Chaves...) para abrir a caixa de diálogo **Indexes/Keys**;

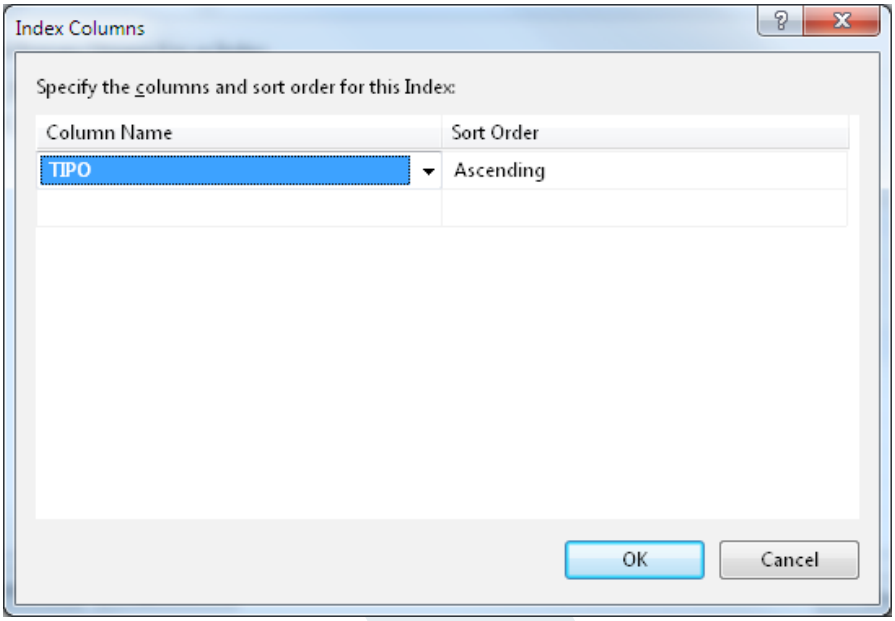


8. Clique no botão **Add** (**Adicionar**);

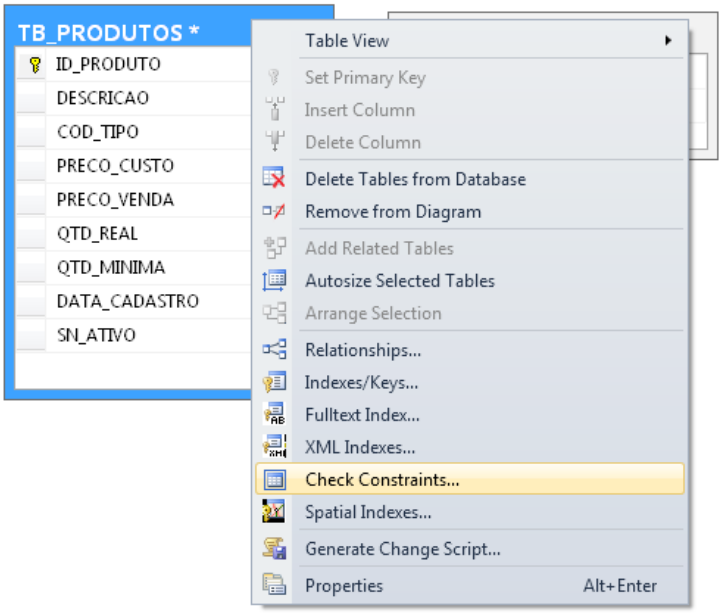


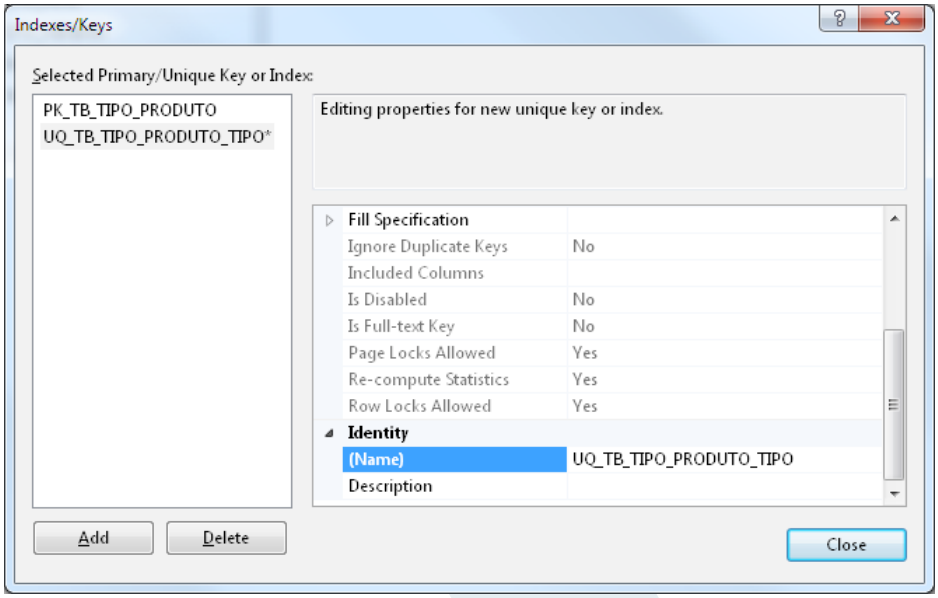

9. Altere as propriedades **Type** (**Tipo**) para **Unique Key** (**Chave Exclusiva**) e **Columns (Colunas)** para **TIPO**;



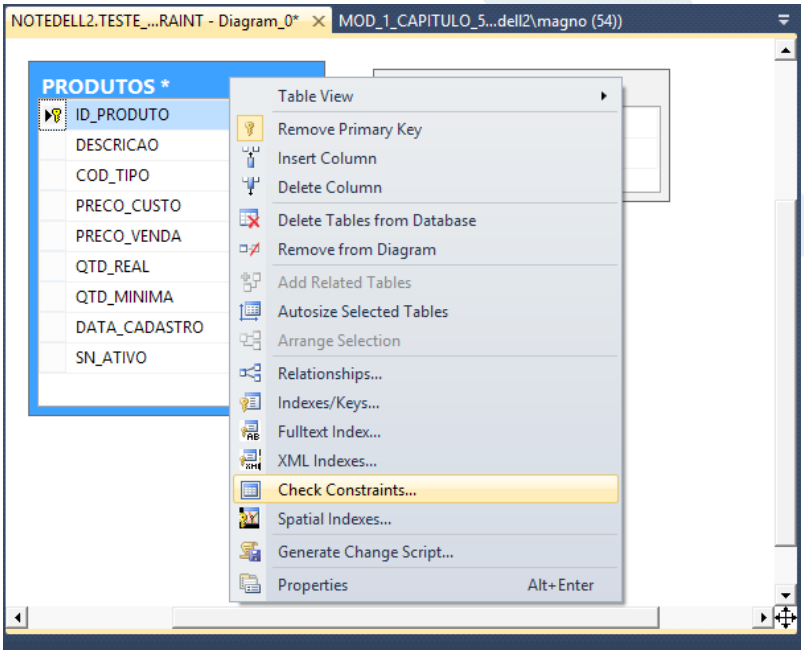



10. Altere as propriedades **Name** (**Nome**) para **UQ_TB_TIPO_PRODUTO_TIPO**. Depois, clique em **Close** (**Fechar**);

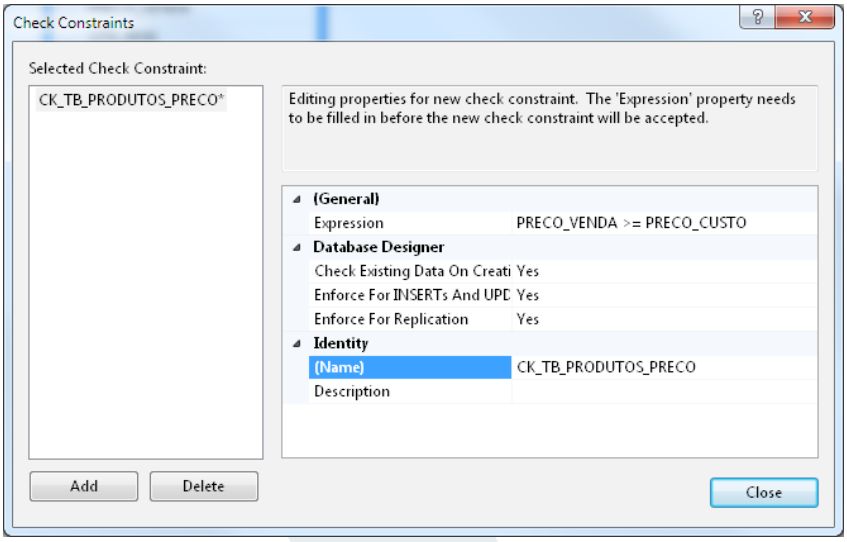




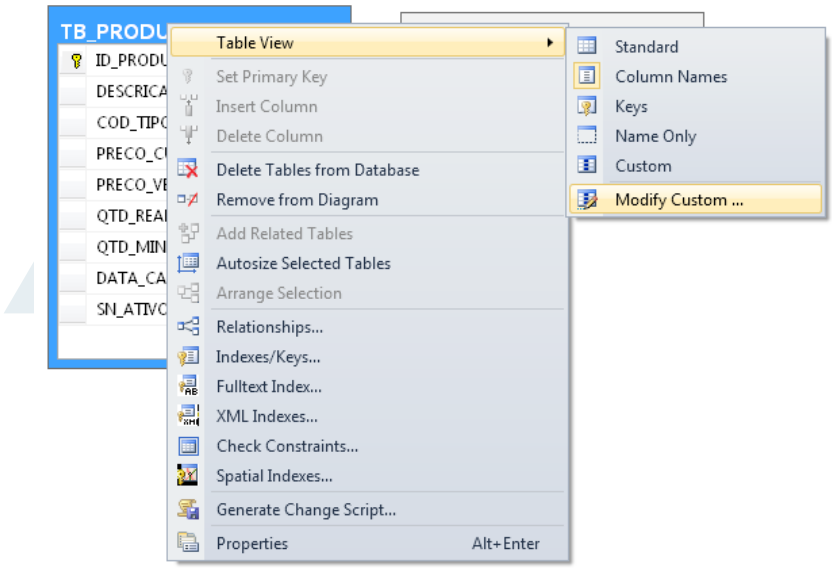

11. Para criar as constraints **CHECK**, clique com o botão direito do mouse sobre a tabela **TB_PRODUTO** e selecione a opção **Check Constraints...** (**Restrições de Verificação...**);



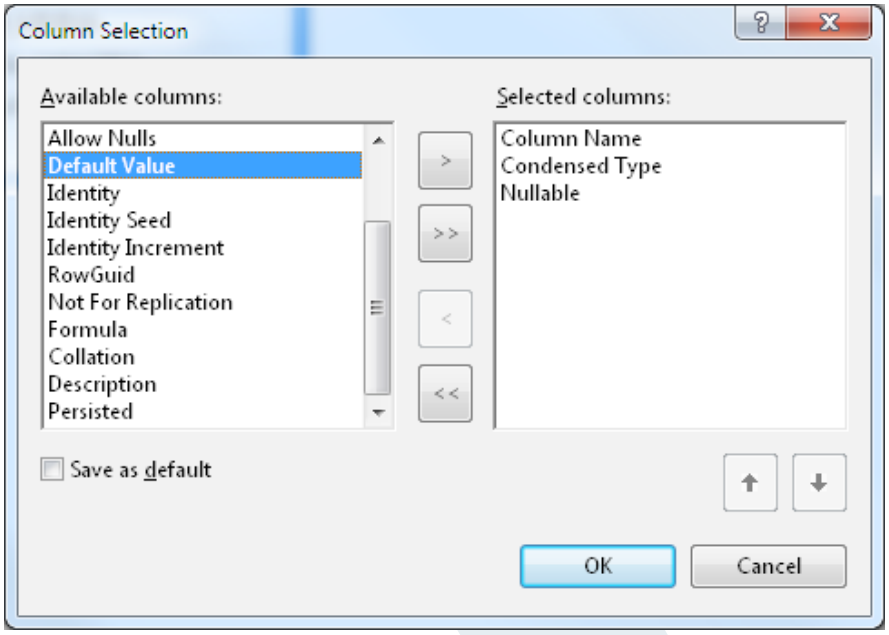
12. Na caixa de diálogo que é aberta, clique no botão **Add** e depois altere as seguintes propriedades:



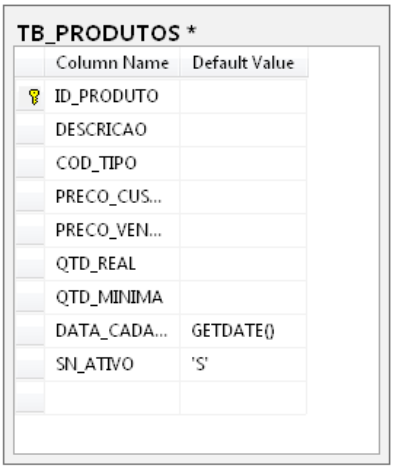

13. Para definir os valores default (padrão) de cada campo, clique com o botão direito do mouse na tabela **TB_PRODUTO**, selecione **Table View** (**Exibição da Tabela...**) e depois **Modify Custom View** (**Modificar Personalização**);



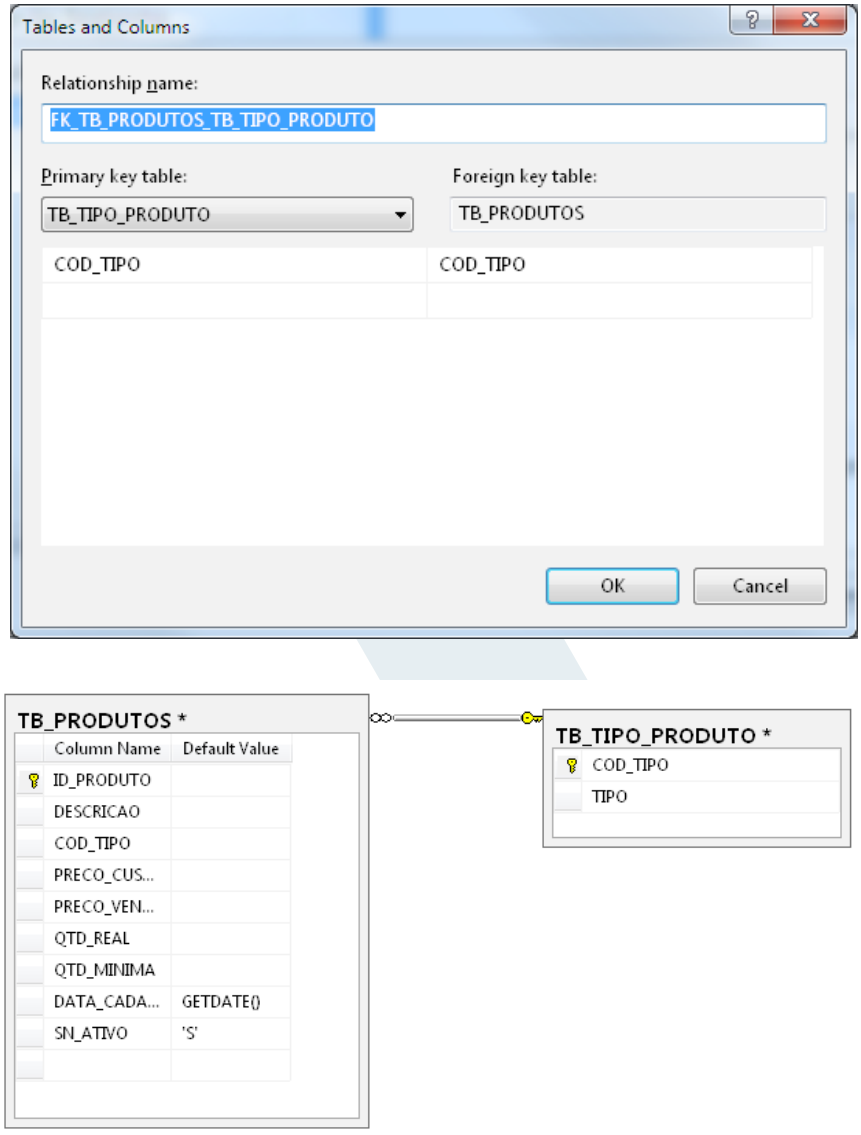
14. Na janela aberta, selecione as colunas **Condensed Type** e **Nullable** e remova-as clicando no botão <. Em seguida, adicione a coluna **Default Value** clicando no botão >. Feche a janela clicando em **OK**;




15. Para exibir os valores padrão, clique com o botão direito do mouse sobre a tabela **TB_PRODUTO**, selecione **Table View** e clique na opção **Custom**. Por fim, informe o valor padrão ao lado do nome do campo **DATA_CADASTRO** e **SN_ATIVO**;




16. Para definir a chave estrangeira, selecione o campo **COD_TIPO** da tabela **TB_PRODUTO** e arraste-o até o campo **COD_TIPO** da tabela **TIPO_PRODUTO**.



# Pontos principais

**Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes da leitura.**

- Os objetos que fazem parte de um sistema são criados dentro de um objeto denominado **database**, ou seja, uma estrutura lógica formada por dois tipos de arquivo: um responsável pelo armazenamento de dados e outro que armazena as transações feitas. Para que um banco de dados seja criado no SQL Server, é necessário utilizar a instrução **CREATE DATABASE**;

- Os dados de um sistema são armazenados em objetos denominados tabelas (**tables**). Cada uma das colunas de uma tabela refere-se a um atributo associado a uma determinada entidade. A instrução **CREATE TABLE** deve ser utilizada para criar tabelas dentro de bancos de dados já existentes;

- Cada elemento, como uma coluna, uma variável ou uma expressão, possui um tipo de dado. O tipo de dado especifica o tipo de valor que o objeto pode armazenar, como números inteiros, texto, data e hora etc.;

- Normalmente, as tabelas possuem uma coluna contendo valores capazes de identificar uma linha de forma exclusiva. Essa coluna recebe o nome de chave primária, cuja finalidade é assegurar a integridade dos dados da tabela;

- As constraints são objetos utilizados com a finalidade de definir regras referentes à integridade e à consistência nas colunas das tabelas que fazem parte de um sistema de banco de dados;

- Para assegurar a integridade dos dados de uma tabela, o SQL Server oferece cinco tipos diferentes de constraints: **PRIMARY KEY**, **FOREIGN KEY**, **UNIQUE**, **CHECK** e **DEFAULT**;

- Cada uma das constraints possui regras de utilização. Uma coluna que é definida como chave primária, por exemplo, não pode aceitar valores nulos. Em cada tabela, pode haver somente uma constraint de chave primária;

- Podemos criar constraints com o uso de **CREATE TABLE**, **ALTER TABLE** ou graficamente (a partir da interface do SQL Server Management Studio).