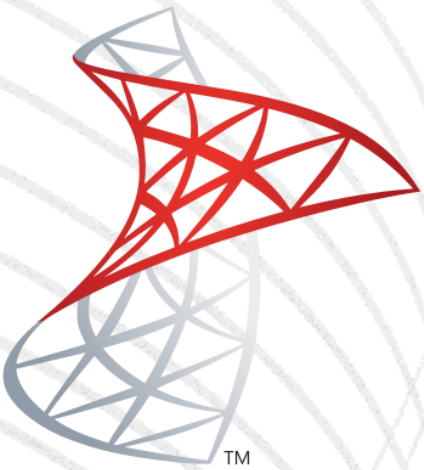


Microsoft®
SQL Server®

Esta apostila foi criada e editada com base em informações contidas nos sites **aprendendo.net**, **devmedia.com.br**, **dicasdeprogramacao.com.br**, **macoratti.net**, **gmx.net**, **targettrust.com.br**, **1keydata.com**, **technet.microsoft.com**, **msdn.microsoft.com** e **martinsfontespaulista.com.br**.

As imagens nela contidas foram capturadas com *PrintScreen* de computadores associados.

Sumário

1. Introdução 5

1.1. SGBD 5

1.2. A linguagem SQL 6

1.2.1. Linguagem de múltiplo uso 7

2. SQL Server 8

2.1. Management Studio 8

3. Criando banco de dados e tabelas 10

3.1. Criando um banco de dados com T-SQL 10

3.2. Criando um banco de dados com Editor Gráfico 11

3.3. Tabelas 13

3.2.1. Tipos de dados 13

3.2.2. Criando tabelas 14

3.4. Criando tabelas pelo Editor Gráfico 15

4. Inserindo registros em tabelas 17

5. Consultando Dados 19

5.1. Comando SELECT 19

5.1.1. Selecionando todas as Colunas 20

5.1.2. Selecionando Colunas Específicas e todas as Linhas 20

5.2. Cláusula WHERE com condições simples 21

6. Alterando tabelas 25

6.1. Alterando a estrutura 25

6.2. Alterando registros 27

6.3. Excluindo registros de uma tabela 28

7. Auto Incremento 30

8. Chave primária e chave estrangeira 33

8.1. Chave primária 33

8.2. Chave estrangeira 34

9. Relacionamentos 36

9.1. Criar o banco de dados. 36

9.2. Criar as tabelas 37

9.3. Criando relacionamentos entre as tabelas 40

10. Consultas Avançadas 43

10.1. ORDER BY 44

10.1.1. Ordenando por uma coluna 44

10.1.2. Ordenando por várias colunas 44
10.1.3. ORDER BY ASC e DESC 45
10.2. Cláusula TOP 47
10.2.1. Cláusula TOP com ORDER BY 47
10.2.3. Cláusula TOP WITH TIES com ORDER BY 48
10.3. Operador BETWEEN 49
11. Funções de agrupamento 51
11.1. COUNT 51
11.2. SUM 55
11.3. AVG 58
11.4. Max 62

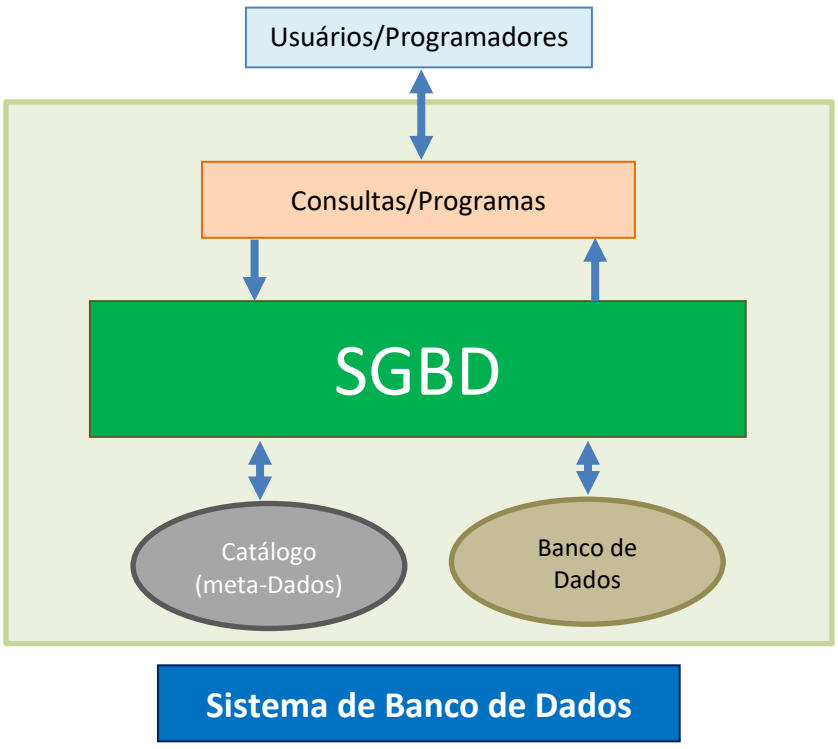
1. Introdução

Para dar início ao estudo desta apostila, é necessário ter como base dois conceitos relacionados à banco de dados, o SGBD e a linguagem SQL.

1.1. SGBD

A gerência de um banco de dados não é uma coisa a se deixar de lado, pois uma empresa pode depender dele, ou seja, pode ajudar a empresa a ter sucesso, mas também pode levá-la ao fracasso. Para garantir a consistência dos dados, controlar o acesso, manter os dados seguros, fornecer meios de acesso aos dados, ... foram criados os Sistemas de Gerenciamento de Bancos de Dados, ou SGBD (DBMS em inglês Database Management System).

Um SGBD é o conjunto de programas de computador (softwares) responsáveis pelo gerenciamento de uma base de dados. Seu principal objetivo é retirar da aplicação cliente a responsabilidade de gerenciar o acesso, a manipulação e a organização dos dados. O SGBD disponibiliza uma interface para que seus clientes possam incluir, alterar ou consultar dados previamente armazenados. Em bancos de dados relacionais a interface é constituída pelas APIs (Application Programming Interface) ou drivers do SGBD, que executam comandos na linguagem SQL (Structured Query Language).



Tudo que fazemos em um banco de dados passa pelo SGBD! O SGBD é responsável por tudo, salvar os dados no HD, manter em memória os dados mais acessados, ligar dados e metadados, disponibilizar uma interface para programas e usuários externos acessem o banco de dados (para banco de dados relacionais, é utilizada a linguagem SQL), encriptar dados, controlar o acesso a informações, manter cópias dos dados para recuperação de uma possível falha, garantir transações no banco de dados, enfim, sem o SGBD o banco de dados não funciona!

É comum as pessoas chamarem um SGBD de banco de dados, por exemplo: banco de dados Oracle, banco de dados MySQL, banco de dados SQL Server, etc. Na verdade esses são os SGBDs, banco de dados é o que eles oferecem, o correto é chamá-los de: SGBD Oracle, SGBD MySQL, SGBD SQL Server, etc. Cada um implementa um banco de dados (ou vários) de uma maneira diferente, mas para o usuário isso é quase transparente, pois a linguagem de acesso aos dados é a mesma, o SQL.

1.2. A linguagem SQL

SQL (Structured Query Language) é a linguagem padrão universal para manipular bancos de dados relacionais através dos SGBDs. Isso significa que todos os SGBDRs (Sistema de Gerenciamento de Banco de Dados Relacionais) oferecem uma interface para acessar o banco de dados utilizando a linguagem SQL, embora com algumas variações. Logo, saber o que é SQL e como utilizá-la é fundamental para qualquer desenvolvedor de softwares.

A linguagem SQL surgiu em meados da década de 70, sendo resultado de um estudo de E. F. Codd, membro do laboratório de pesquisa da IBM em San Jose, Califórnia. Este estudo tinha foco em desenvolver uma linguagem que se adapta ao modelo relacional. O primeiro sistema de BD baseado em SQL tornou-se comercial no final dos anos 70 juntamente com outros sistemas de BD's relacionais.

Em 1982, foi lançada a primeira versão padronizada da SQL, que vieram ganhando melhorias de acordo com sua evolução e tornando-se assim, a mais poderosa ferramenta para definição e manipulação de BD's e hoje utilizada em grande parte dos BD existente, tais como MySQL, SQLServer, Firebird dentre outros.

A "Linguagem Estruturada de Consultas" (SQL, traduzida para o português) é utilizada para interagir com o SGBD e executar várias tarefas como inserir e alterar registros, criar objetos no banco de dados, gerenciar usuário, consultar informações, controlar transações, etc. Todas as operações realizadas no banco de dados podem ser solicitadas ao SGBD utilizando esta linguagem.

1.2.1. Linguagem de múltiplo uso

A linguagem SQL é dividida em 4 agrupamentos de acordo com o tipo de operação a ser executada no banco de dados.

- **Linguagem de Manipulação de Dados ou DML (Data Manipulation Language):** A DML é um subconjunto da linguagem SQL, utilizada para **Selecionar** (SELECT), **Inserir**(INSERT), **Atualizar**(UPDATE) e **Apagar**(DELETE).

- **Linguagem de Definição de Dados ou DDL (Data Definition Language):** A DDL permite ao usuário a manipulação de tabelas e elementos associados, tipo chave primária e chaves estrangeira, índices, etc. Os principais comandos são CREATE, DROP, ALTER (em algumas situações).

- **Linguagem de Controle de Dados ou DCL (Data Control Language):** A DCL controla os aspectos destinados a autorização de dados e licenças de usuários para manipulação de dados dentro do BD.Alguns comandos comuns são GRANT(dá privilégios para usuários), REVOKE (revoga privilégios de usuários), COMMIT(em resumo grava dados no BD) e ROLLBACK(descarta dados existentes desde o último COMMIT).

- **Linguagem de Consultas de Dados ou DQL (Data Query Language):** embora na DQL exista somente um comando(SELECT) é o mais utilizado, principalmente para consultas parametrizadas. Lembre que o SELECT também é considerado um comanda DML.

2. SQL Server

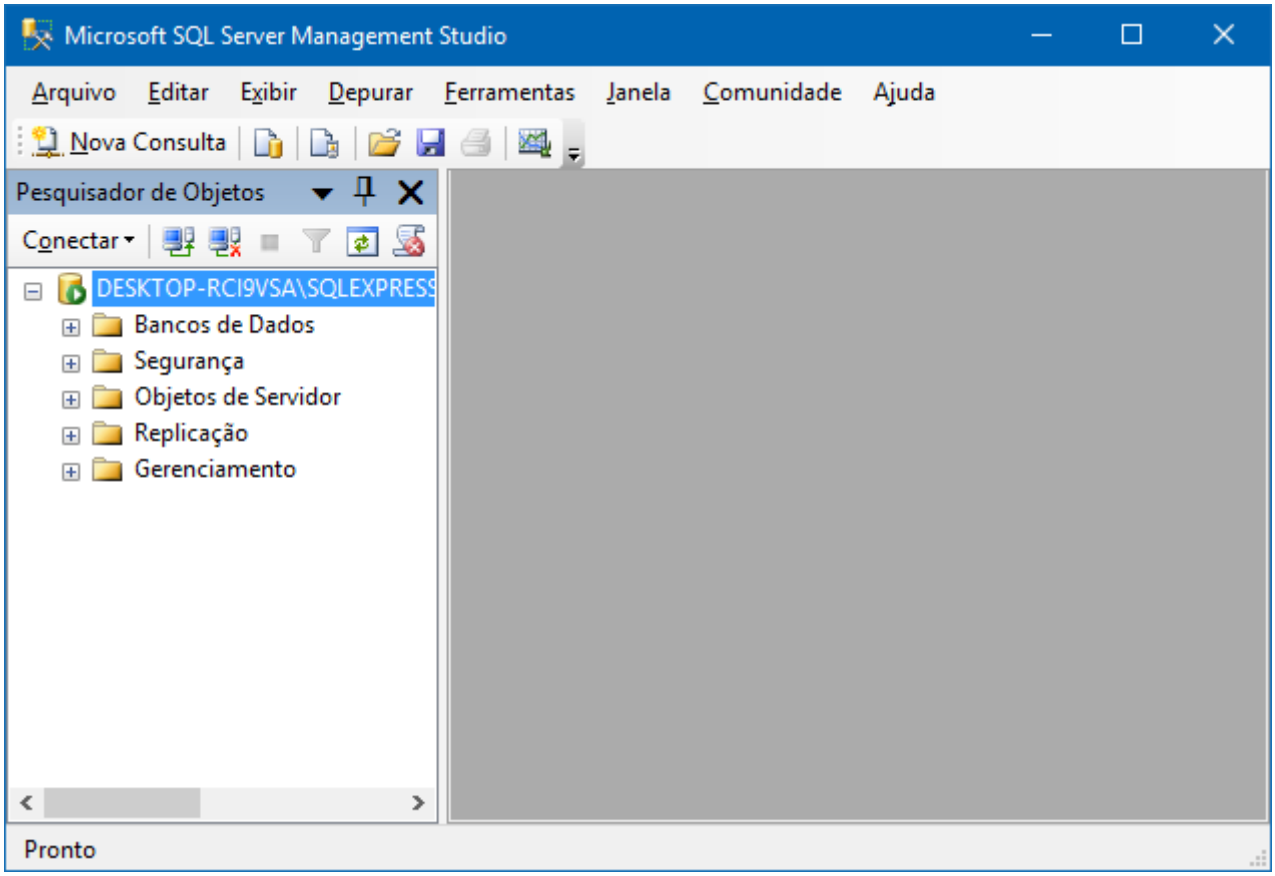
O Microsoft SQL Server é um sistema gerenciador de Banco de dados relacional (SGBD) desenvolvido pela Microsoft. Foi criado em parceria com a Sybase em 1988 inicialmente para a plataforma OS/2. Esta parceria durou até 1994, com o lançamento da versão para Windows NT e desde então a Microsoft mantém a manutenção do produto. Como um Banco de dados, é um produto de software cuja principal função é a de armazenar e recuperar dados solicitados por outras aplicações de software, seja aqueles no mesmo computador ou aqueles em execução em outro computador através de uma rede (incluindo a Internet). Há pelo menos uma dúzia de diferentes edições do Microsoft SQL Server destinadas a públicos diferentes e para diferentes cargas de trabalho.

2.1. Management Studio

O Management Studio é o substituto do Enterprise Manager com a principal interface de gerenciamento dos recursos do SQL Server 2008.

Ao carregar o programa, é exibido uma caixa de diálogo em que é possível selecionar o servidor ao qual ele estará ligado. Pode-se manter as opções padrão oferecidas ou selecionar o tipo de servidor que deve ser conectado, seu nome e método de autenticação.

Ele exibe dois painéis em que os componentes e objetos são exibidos.



Pesquisador de Objetos: tem o funcionamento parecido com o do Explorador de Arquivos do Windows. Ele exibe uma árvore com os objetos mantidos pelo SQL Server.

Painel de documentos: exibe informações específicas de um objeto selecionado num dos dois outros painéis, assim como editores de consultas.

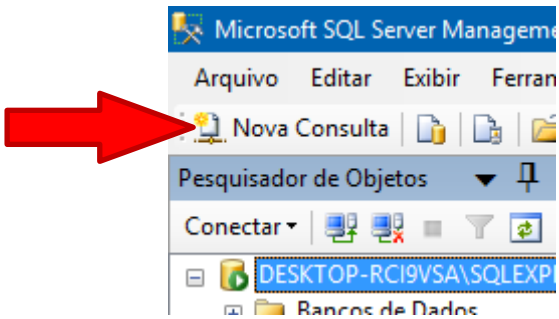
3. Criando banco de dados e tabelas

A criação do banco de dados e suas tabelas é, muitas vezes, um dos primeiros passos da parte prática do desenvolvimento de um novo projeto.

Para auxiliar nessa etapa, veremos neste capítulo como criar bancos de dados e tabelas no SQL Server, utilizando tanto instruções T-SQL quanto o editor gráfico.

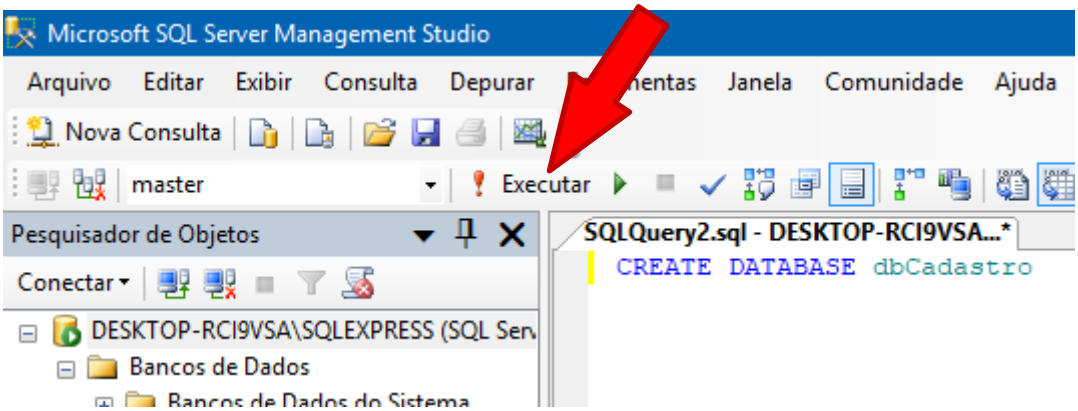
3.1. Criando um banco de dados com T-SQL

Vamos iniciar criando um banco de dados usando **T-SQL**, então, com o Management Studio aberto, clique no botão **Nova Consulta**, como mostra a figura abaixo:



Em seguida, na parte central da interface aparecerá uma tela em branco, na qual você poderá digitar os códigos para criar o banco de dados, tabelas, efetuar consultas, etc. Nesse momento, vamos executar o código a seguir e pressionar **F5** ou clicar em **Executar**, como mostra abaixo:

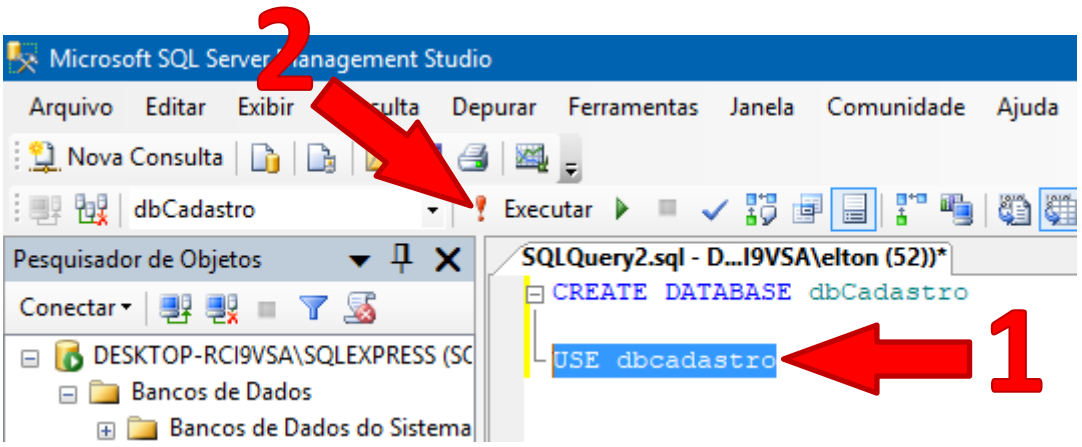
```
CREATE DATABASE dbCadastro
```



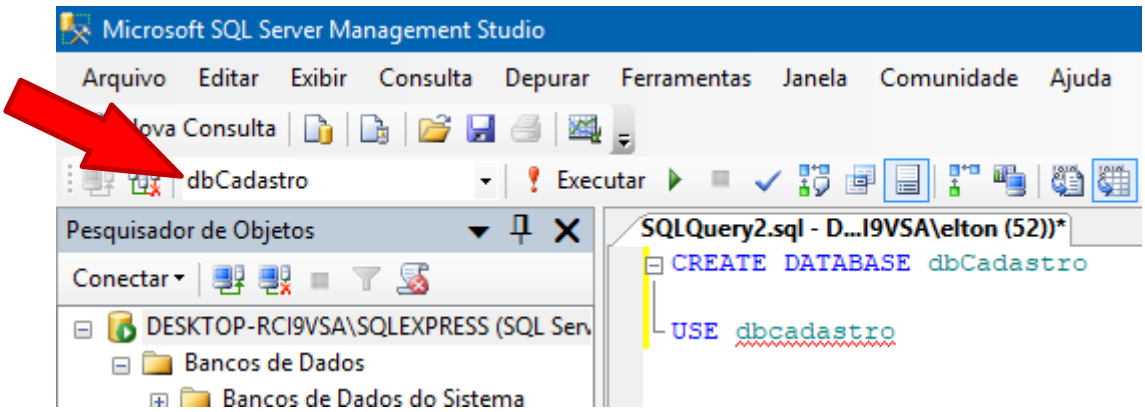
Com o banco criado, precisamos entrar no seu contexto para poder executar novos scripts dentro dele. No caso, precisamos acessá-lo para criar nossa tabela. Para isso, devemos executar o seguinte comando:

```
USE dbCadastro
```

Antes de clicar em Executar, é necessário selecionar a nova linha digitada, para que não execute a linha já executada anteriormente.



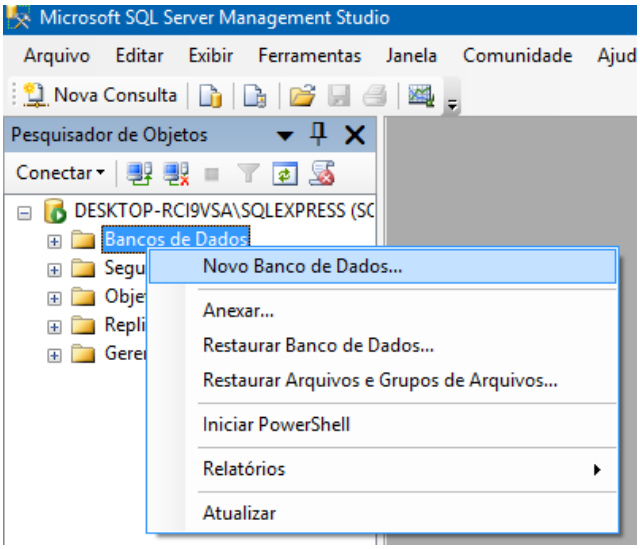
Para confirmar que a operação foi executada como esperado, veja na parte superior esquerda do Management Studio se o banco criado está em uso, como mostra abaixo:



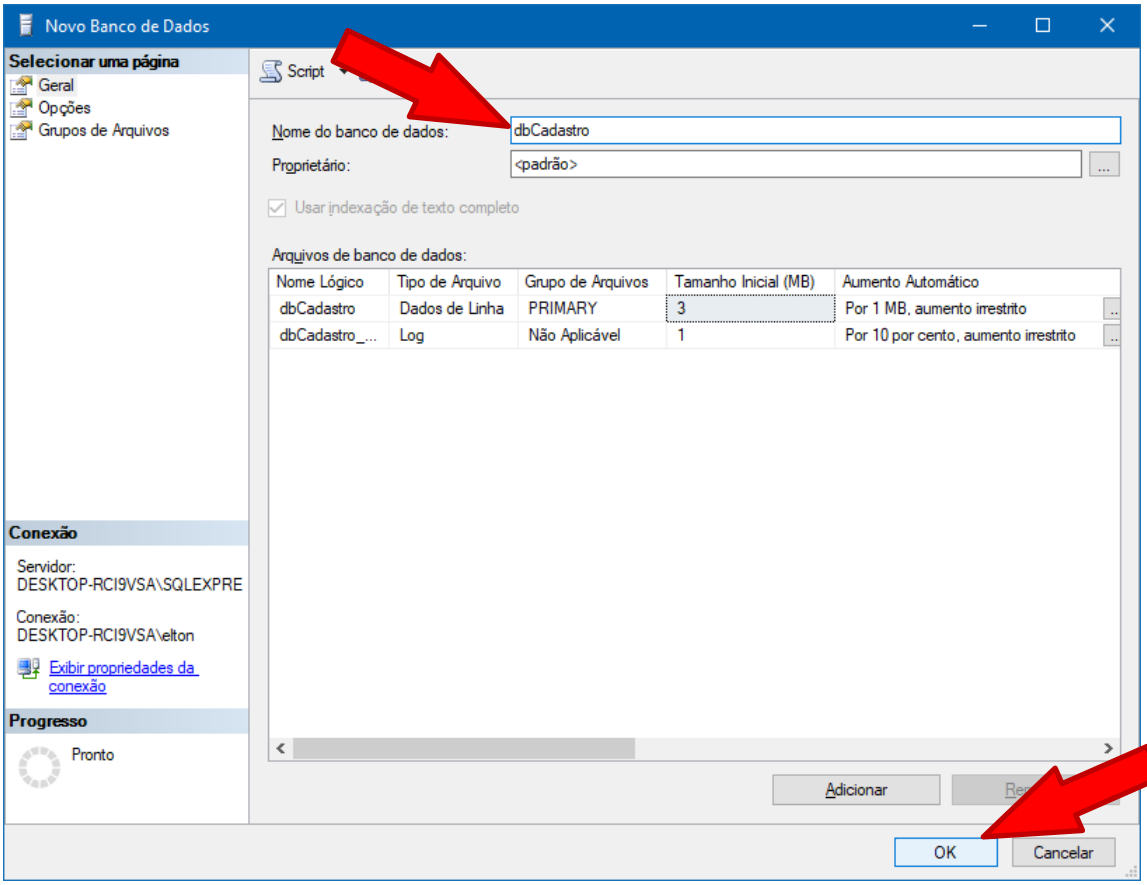
3.2. Criando um banco de dados com Editor Gráfico

Agora vamos refazer os procedimentos anteriores, ou seja, criar um banco de dados, mas sem utilizar scripts. Ao invés disso, utilizaremos o editor gráfico do Management Studio, que nos permite efetuar a mesma tarefa por meio de telas auxiliares.

Então, no Management Studio, localize o Pesquisador de Objetos, clique com o botão direito sobre a pasta **Banco de Dados** e em seguida em **Novo Banco de Dados**, como mostra a figura abaixo:



Feito isso, aparecerá uma janela para criar um banco de dados. Nela, basta preencher o nome do database e clicar em OK.



3.3. Tabelas

A tabela é um conjunto de dados dispostos em número infinito de colunas e número ilimitado de linhas. As colunas são tipicamente consideradas os campos da tabela, e caracterizam os tipos de dados que deverão constar nela.

3.2.1. Tipos de dados

Os tipos de dados entre os diferentes SGBDs são padronizados pelo ANSI, mas alguns bancos podem utilizar nomenclaturas diferentes. Segue a tabela de tipos comuns:

Tipo de dados	***Espaço de armazenamento***	***Descrição***
BINARY	1 byte por caractere	Qualquer tipo de dados pode ser armazenado em um campo deste tipo. Não é feita nenhuma conversão dos dados (em texto, por exemplo). O modo no qual os dados são inseridos em um campo binário especifica o modo como eles serão exibidos na saída.
BIT	1 byte	Valores Sim, Não e campos que só contêm um dos dois valores.
TINYINT	1 byte	Um valor inteiro entre 0 e 255.
MONEY	8 bytes	Um inteiro dimensionado entre – 922.337.203.685.477,5808 e 922.337.203.685.477,5807.
DATETIME (consulte DOUBLE)	8 bytes	Um valor de data ou hora entre os anos 100 e 9999.
UNIQUEIDENTIFIER	128 bits	Um número de identificação exclusivo usado com RPCs (Chamadas de Procedimento Remoto).
REAL	4 bytes	Um valor de ponto flutuante e precisão simples com um intervalo de – 3,402823E38 a – 1,401298E-45 para valores negativos, 1,401298E-45 a 3,402823E38 para valores positivos, e 0.
FLOAT	8 bytes	Um valor de ponto flutuante e precisão dupla com um intervalo de – 1,79769313486232E308 a – 4,94065645841247E-324 para valores negativos, 4,94065645841247E-324 a 1,79769313486232E308 para valores positivos, e 0.

SMALLINT	2 bytes	Um inteiro curto entre – 32.768 e 32.767.
INTEGER	4 bytes	Um inteiro longo entre – 2.147.483.648 and 2.147.483.647.
DECIMAL	17 bytes	Um tipo de dados numérico exato que armazena valores de 1028 - 1 a - 1028 - 1. É possível definir a precisão (1 - 28) e a escala (0 - precisão definida). A precisão e a escala padrão são 18 e 0, respectivamente.
TEXT	2 bytes por caractere	De zero a, no máximo, 2,14 gigabytes.
IMAGE	Como necessário	De zero a, no máximo, 2,14 gigabytes. Usado para objetos OLE.
CHARACTER	2 bytes por caractere	De zero a 255 caracteres.

3.2.2. Criando tabelas

Antes de criar uma tabela, você deve pensar cuidadosamente no nome da tabela. Nomes de tabelas podem ter até 128 caracteres. Os nomes de tabela devem iniciar com um caractere alfabético, mas também podem conter sublinhas (_), símbolos de @, sinais de libra, # e numerais.

Nomes de tabelas devem ser únicos para cada esquema dentro de um banco de dados. Diferentes esquemas, porém, podem conter tabelas com o mesmo nome e cada tabela pode ter até 1.024 colunas. Os nomes de coluna seguem as mesmas regras de atribuição de nomes das tabelas e devem ser únicas.

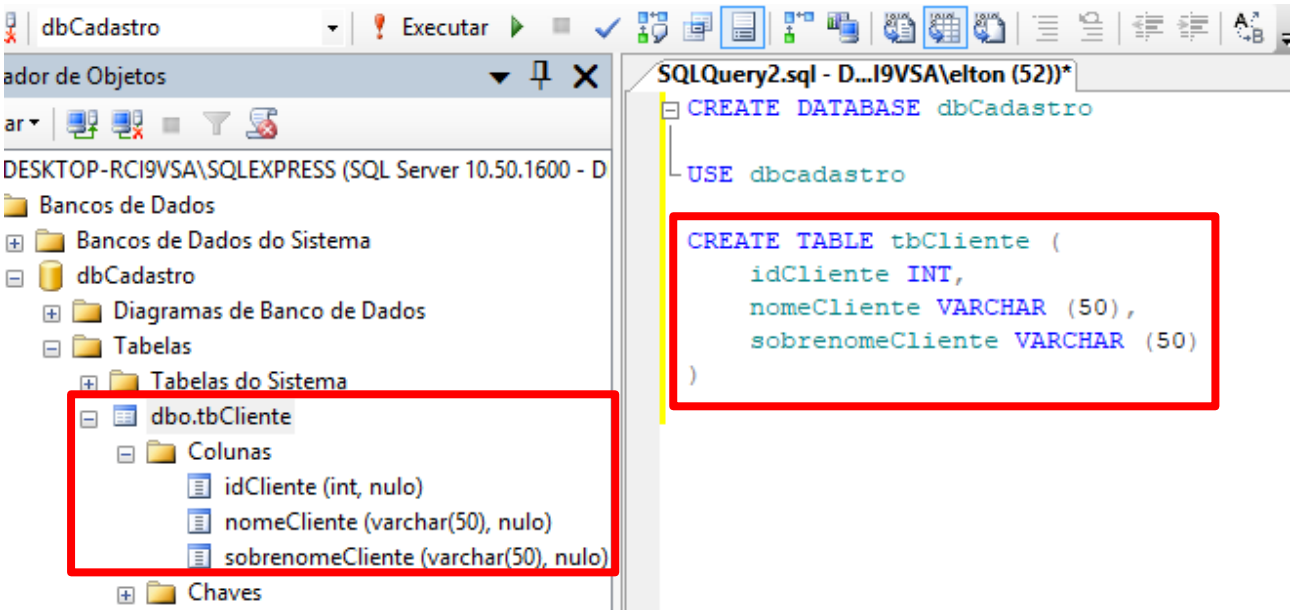
Muitas ferramentas de bases de dados permitem-lhe criar tabelas sem programar em linguagem SQL, mas como as tabelas são o receptáculo de todos os dados, é importante incluir a sintaxe CREATE TABLE.

A sintaxe SQL para CREATE TABLE é:

```
CREATE TABLE "nome_tabela"(
"coluna 1" "tipo_dados_para_coluna_1",
"coluna 2" "tipo_dados_para_coluna_2",
... )
```

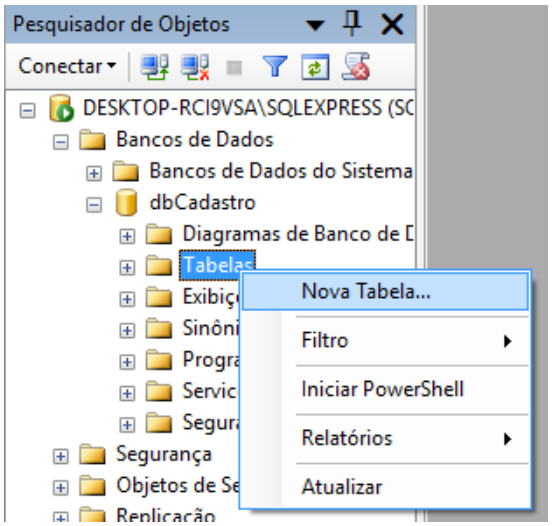
Assim, se pretendermos criar uma tabela de clientes conforme acima especificado, introduziríamos:

```
CREATE TABLE tbCliente (
    idCliente INT NOT NULL,
    nomeCliente VARCHAR (50) NOT NULL,
    sobrenomeCliente VARCHAR (50)
)
```

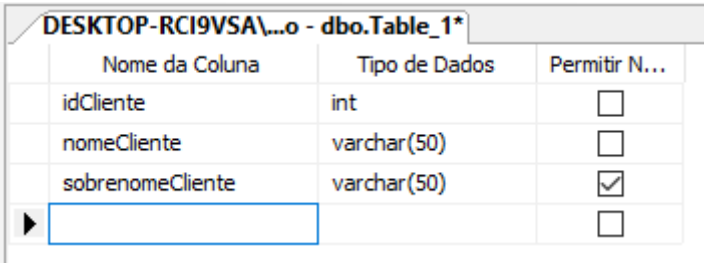


3.4. Criando tabelas pelo Editor Gráfico

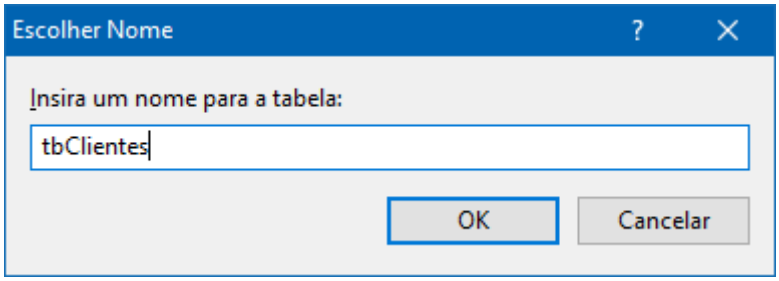
Agora vamos ver como criar uma tabela em um banco de dados sem o uso de scripts. Para isso, devemos expandi-lo no Pesquisador de Objetos, clicar com o botão direito em **Tabelas** e depois em **Nova Tabela**, como mostra abaixo:



Feito isso, será exibida uma tela na qual devemos preencher os campos da tabela, selecionando também o seu tipo, e se ela aceita valore nulos:



Para confirmar a criação da tabela, clique botão com o ícone de um disquete. Em seguida aparecerá uma janela para nomear a tabela. Depois disso ela é criada.



Conhecer o editor visual do Management Studio, bem como os scripts necessários para realizar tarefas comuns, é importante para quem trabalha com bancos de dados, pois esse tipo de procedimento costuma ser frequente durante o desenvolvimento dos projetos

4. Inserindo registros em tabelas

Na linguagem SQL, existe dois métodos de inserir dados em uma tabela. Um é inserir uma linha de cada vez, a outra é inserir várias linhas em simultâneo. O comando INSERT insere novas linhas na tabela. Podem ser inseridas uma ou mais linhas especificadas por expressões de valor, ou zero ou mais linhas resultantes de uma consulta.

Vamos abordar como podemos **INSERT** (Inserir) dados numa linha de cada vez:

A sintaxe para inserir dados numa linha da tabela de cada vez é a seguinte:

```
INSERT INTO "nome_tabela" ("coluna 1", "coluna 2", ...)
VALUES ("valor 1", "valor 2", ...);
```

Assumindo que há uma tabela com a seguinte estrutura

Tabela ***tbCliente***

Nome da coluna	***Tipo de dados***
idCliente	INT
nomeCliente	VARCHAR (50)
cpfCliente	VARCHAR (11)
dtnCliente	DATE
telCliente	VARCHAR (50)

e agora para inserir uma linha adicional na tabela:

```
INSERT INTO tbCliente
VALUES (01, 'Chimena de Souza', '11987654321', '31-08-1932',
'1734213012');
```

No exemplo anterior foi inserido uma linha de registro na tabela **tbCliente**. Para inserir várias linhas utilizando a sintaxe de VALUES multilinha:

```
INSERT INTO tbCliente (idCliente, nomeCliente, cpfCliente, dtnCliente,
telCliente) VALUES
(02,'Gumercindo de Oliveira','12345678910','02-01-1999','1734230102'),
(03,'Florisvaldo dos Santos','32165498700','28-03-1978','1734225169'),
(04,'Elton Rodrigo','78945612311','10-10-1910','1734213112');
```

5. Consultando Dados

Para extrair dados a partir de um banco de dados é necessário utilizar comandos em SQL, o SELECT, assim como é possível restringir as colunas que serão exibidas. Neste capítulo, será mostrado todos os comandos SQL necessários para executar estas ações.

5.1. Comando SELECT

O comando **SELECT** recupera os dados de uma ou mais tabelas, sendo um dos comandos mais simples e, ao mesmo tempo, mais extenso da SQL devido as suas funções, operandos, comandos, sub-comandos e cláusulas não obrigatórias.

A sintaxe básica da instrução de consulta em SQL é:

```
SELECT coluna1, coluna2,....colunaN
FROM tabela;
```

A consulta busca dados de determinadas tabelas.

Tabela ***tbCurso***

idCurso	***nomeCurso***	***precoCurso***	***durCurso***
1	PHP	200,00	80
2	Linux Avançado	100,00	60
3	Java Avançado	250,00	80
4	Photoshop	120,00	80
5	SQL Server Avançado	100,00	60
6	CorelDraw	120,00	60

Em sua forma mais simples, um comando SELECT deve incluir o seguinte:

- Uma cláusula SELECT que especifica as colunas a serem exibidas.
- Uma cláusula FROM que especifica as tabelas que possuem as colunas listadas na cláusula SELECT.

Na sintaxe:

- **SELECT:** é uma lista de uma ou mais colunas.
- **Column:** seleciona a coluna nomeada.
- **FROM table:** especifica a tabela que contém as colunas.

5.1.1. Selecionando todas as Colunas

Você pode exibir todas as colunas de dados de uma tabela colocando um asterisco (*) logo após a palavra chave SELECT. No exemplo abaixo, a tabela de tbCurso possui quatro colunas: idCurso, nomeCurso, precoCurso, durCurso.

```
SELECT * FROM tbCurso
```

Após executar o comando, na aba **Resultados**, será exibida a tabela tbCurso completa.

Resultados | Mensagens

	idCurso	nomeCurso	precoCurso	durCurso
1	1	PHP	200	80
2	2	Linux Avançado	100	60
3	3	Java Avançado	250	80
4	4	Photoshop	120	80
5	5	SQL Server Avançado	100	60
6	6	CorelDraw	120	60

Você também pode exibir todas as colunas da tabela listando-as depois da palavra chave SELECT. Por exemplo, o seguinte comando SQL, como no exemplo acima, também exibe todas as colunas e todas as linhas da tabela tbCurso:

```
SELECT idCurso,nomeCurso,precoCurso,durCurso FROM tbCurso
```

5.1.2. Selecionando Colunas Específicas e todas as Linhas

Você pode usar o comando SELECT para exibir colunas específicas da tabela especificando os nomes das colunas, separados por vírgulas. O exemplo abaixo exibe todos os ids e nomes da tabela tbCurso.

```
SELECT idCurso, nomeCurso FROM tbCurso
```

Resultados | Mensagens

	idCurso	nomeCurso
1	1	PHP
2	2	Linux Avançado
3	3	Java Avançado
4	4	Photoshop
5	5	SQL Server Avançado
6	6	CorelDraw

Na cláusula SELECT, especifique as colunas que você quer ver, na ordem na qual você quer que elas sejam mostradas. Por exemplo, para exibir nome antes do id do curso, você utiliza o seguinte comando:

```
SELECT nomeCurso, idCurso FROM tbCurso
```

Resultados | Mensagens

	nomeCurso	idCurso
1	PHP	1
2	Linux Avançado	2
3	Java Avançado	3
4	Photoshop	4
5	SQL Server Avançado	5
6	CorelDraw	6

5.2. Cláusula WHERE com condições simples

Podemos filtrar colunas para nos mostrar apenas os dados que nos interessa através da cláusula WHERE em conjunto com os operadores comparativos.

Sintaxe:

```
select  Coluna, Coluna, ..., Coluna from  Tabela
where Condição
```

Tabela: Nome da tabela.

Coluna: Nome de uma coluna – Para mostrar todas as colunas pode-se colocar apenas a máscara "*" no lugar do nome das colunas.

Condição: Cria uma condição para filtrar os dados utilizando os operadores comparativos.

Operador comparativo "igual" (=) e LIKE:

```
SELECT * FROM tbCurso WHERE durCurso = 80
```

Resultados | Mensagens

	idCurso	nomeCurso	precoCurso	durCurso
1	1	PHP	200	80
2	3	Java Avançado	250	80
3	4	Photoshop	120	80

No exemplo acima foram filtrados os cursos onde as durações eram iguais a 80.

Para busca de *string*, o conteúdo a ser filtrado deve ser especificado por apóstrofos ('), veja abaixo:

```
SELECT * FROM tbCurso WHERE nomeCurso = 'Photoshop'
```

Resultados | Mensagens

	idCurso	nomeCurso	precoCurso	durCurso
1	4	Photoshop	120	80

Para busca parcial de *string*, o SELECT fornece o operador **LIKE**. Veja o exemplo abaixo:

```
SELECT * FROM tbCurso WHERE nomeCurso LIKE 'Linux%'
```

Resultados | Mensagens

	idCurso	nomeCurso	precoCurso	durCurso
1	2	Linux Avançado	100	60

Neste comando, todos os cursos cujos nomes iniciam com Linux serão retornados. Se quisermos retornar os nomes que contenham 'Avançado' também no meio, podemos alterar para o exemplo a seguir:

```
SELECT * FROM tbCurso WHERE nomeCurso LIKE '%Avançado%'
```

Resultados | Mensagens

	idCurso	nomeCurso	precoCurso	durCurso
1	2	Linux Avançado	100	60
2	3	Java Avançado	250	80
3	5	SQL Server Avançado	100	60

Operador comparativo "diferente" (<>):

```
SELECT nomeCurso, durCurso FROM tbCurso WHERE durCurso = 80
```

Resultados | Mensagens

	nomeCurso	durCurso
1	PHP	80
2	Java Avançado	80
3	Photoshop	80

Agora utilizando o operador comparativo **<>**, foram retornados os nomes dos cursos onde as durações de cada eram "diferentes" de 80.

Operador comparativo "maior que" (>):

```
SELECT nomeCurso, precoCurso FROM tbCurso WHERE precoCurso > 100
```

Resultados | Mensagens

	nomeCurso	precoCurso
1	PHP	200
2	Java Avançado	250
3	Photoshop	120
4	CorelDraw	120

Nesse exemplo foram filtrados os cursos onde os valores da coluna **precoCurso** eram maiores que 100.

O mesmo pode ser realizado para busca de *string*, onde o conteúdo a ser retornado será da coluna nomeCurso, veja abaixo:

```
SELECT nomeCurso FROM tbCurso WHERE nomeCurso > 'linux'
```

Resultados | Mensagens

	nomeCurso
1	PHP
2	Linux Avançado
3	Photoshop
4	SQL Server Avançado

Operador comparativo "menor que" (<):

```
SELECT * FROM tbCurso WHERE precoCurso < 130
```

idCurso	nomeCurso	precoCurso	durCurso
2	Linux Avançado	100	60
4	Photoshop	120	80
5	SQL Server Avançado	100	60
6	CorelDraw	120	60

6. Alterando tabelas

6.1. Alterando a estrutura

Assim que uma tabela for criada na base de dados, muitas vezes o utilizador poderá querer alterar a estrutura da tabela. Os casos típicos incluem o seguinte:

- Adicionar uma coluna
- Remover uma coluna
- Alterar o nome de uma coluna
- Alterar o tipo de dados de uma coluna

Note que o acima apresentado não constitui uma lista exaustiva. Existem outros momentos em que ALTER TABLE é utilizado para alterar a estrutura da tabela, tal como alterar a especificação da chave primária ou adicionar uma restrição única a uma coluna.

A sintaxe SQL para ALTER TABLE é:

```
ALTER TABLE "nome_tabela"
[alter specification];
```

[alter specification] depende do tipo de alteração que pretendemos efetuar. Para os fins acima citados, as instruções [alter specification] são:

- Adicionar uma coluna: ADD "coluna 1" "tipo de dados para a coluna 1"
- Remover uma coluna: DROP "coluna 1"
- Alterar o nome de uma coluna: sp_RENAME "Tabela.antigo nome da coluna" "novo nome da coluna" "tipo de dados para novo nome da coluna"
- Alterar o tipo de dados de uma coluna: ALTER COLUMN "coluna 1" "novo tipo de dados"

Vamos analisar cada um desses exemplos utilizando a tabela abaixo:

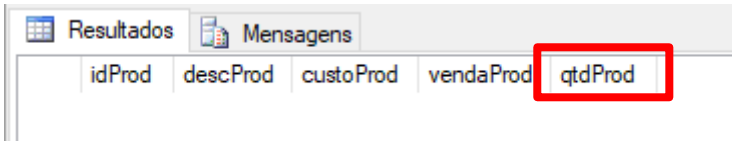
Tabela ***tbProdutos***

Nome da Coluna	***Tipo de dados***
idprod	Int
descProd	Varchar (50)
custoProd	Float
vendaProd	Float

Primeiro, queremos adicionar uma coluna denominada "Quantidade" a esta tabela. Veja abaixo:

```
ALTER TABLE tbProdutos ADD qtdProd FLOAT;
```

Estrutura da tabela obtida:

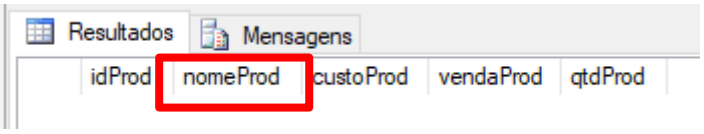


A seguir, queremos renomear "descProd" para "nomeProd". Para renomear uma coluna de uma tabela no Banco de Dados SQL Server é necessário utilizar a seguinte sintaxe:

```
sp_RENAME '[Tabela.NomeColuna]', '[NovoNomeColuna]'
```

Para tal, introduzimos

```
sp_RENAME 'tbProdutos.descProd', 'nomeProd'
```



Em seguida, queremos alterar o tipo de dados de "custoProd" para decimal (10,2). Veja abaixo:

```
ALTER TABLE tbProdutos ALTER COLUMN custoProd DECIMAL(10,2)
```

Nome da Coluna	Tipo de Dados	Permitir N...
idProd	int	☑
nomeProd	char(30)	☑
custoProd	decimal(10, 2)	☑
vendaProd	float	☑
qtdProd	float	☑

Para remover um campo de uma tabela é ainda mais simples, não havendo a necessidade de inserir o tipo do campo.

```
ALTER TABLE tbProdutos DROP COLUMN qtdProd
```

Resultados | Mensagens

idProd	nomeProd	custoProd	vendaProd

6.2. Alterando registros

Para alterar linhas em uma tabela usa-se o comando UPDATE. Este comando faz parte das instruções DML (Data Manipulation Language) ou Linguagem de manipulação de dados. A instrução UPDATE pode alterar um ou mais registros, veja a sintaxe:

```
UPDATE "nomeTabela" SET "nomeColuna" = 'valor'
```

Consideramos a tabela tbProdutos com os registros abaixo:

idProd	nomeProd	custoProd	vendaProd	qtdProd
1	Arroz	4.50	8.99	30
2	Feijão	3.12	5.89	20
3	Sabonete	0.75	1.30	50
4	Maionese	2.45	4.90	10
5	Creme Dental	1.20	2.40	18
6	Macarrão	0.80	2.00	63
7	Detergente	1.60	3.19	30
8	Saco para lixo	2.33	4.50	50

Para atualizar todos os registros de uma tabela, especifique o nome da tabela e use a cláusula SET para especificar os campos a serem alterados.

```
UPDATE tbProdutos SET qtdProd = 90
```

Resultados | Mensagens

idProd	nomeProd	custoProd	vendaProd	qtdProd
1	Arroz	4.50	8.99	90
2	Feijão	3.12	5.89	90
3	Sabonete	0.75	1.30	90
4	Maionese	2.45	4.90	90
5	Creme Dental	1.20	2.40	90
6	Macarrão	0.80	2.00	90
7	Detergente	1.60	3.19	90
8	Saco para lixo	2.33	4.50	90

Na maioria dos casos, convém qualificar a instrução UPDATE com uma cláusula WHERE para limitar o número de registros alterados.

```
UPDATE tbProdutos SET vendaProd = 2.39 WHERE nomeProd = 'Creme Dental'
```

Resultados | Mensagens

	idProd	nomeProd	custoProd	vendaProd	qtdProd
1	1	Arroz	4.50	8.99	90
2	2	Feijão	3.12	5.89	90
3	3	Sabonete	0.75	1.30	90
4	4	Maionese	2.45	4.90	90
5	5	Creme Dental	1.20	2.39	90
6	6	Macarrão	0.80	2.00	90
7	7	Detergente	1.60	3.19	90
8	8	Saco para lixo	2.33	4.50	90

6.3. Excluindo registros de uma tabela

Para excluir os dados atuais de uma tabela, use a instrução DELETE, que é conhecida normalmente como uma consulta exclusão. Esse processo é conhecido também como truncamento da tabela. A instrução DELETE pode remover um ou mais registros de uma, veja a sintaxe abaixo:

```
DELETE FROM "nomeTabela"
```

A instrução DELETE não remove a estrutura da tabela, apenas os dados atuais mantidos pela estrutura. Para remover todos os registros de uma tabela, use a instrução DELETE e especifique as tabelas das quais deseja excluir todos os registros.

```
DELETE FROM tbProdutos
```

Na maioria dos casos, convém qualificar a instrução DELETE com uma cláusula WHERE para limitar o número de registros a remover.

```
DELETE FROM tbProdutos WHERE idProd = 3
```

Se desejar remover dados somente de determinados campos de uma tabela, use a instrução UPDATE e defina esses campos igualmente como NULL, mas somente se forem campos anuláveis.

```
UPDATE tbProdutos SET nomeProd = NULL
```

	idProd	nomeProd	custoProd	vendaProd	qtdProd
1	1	NULL	4.50	8.99	90
2	2	NULL	3.12	5.89	90
3	4	NULL	2.45	4.90	90
4	5	NULL	1.20	2.39	90
5	6	NULL	0.80	2.00	90
6	7	NULL	1.60	3.19	90
7	8	NULL	2.33	4.50	90

7. Auto Incremento

Trata-se de um recurso nativo do SQL Server presente nas propriedades da coluna, no momento da criação da tabela.

A propriedade IDENTITY é utilizada para atributos (campos/colunas) das tabelas nas funções CREATE TABLE e ALTER TABLE, e tem como finalidade incrementar um valor a cada nova inserção.

A sintaxe para usar está propriedade é:

```
IDENTITY [ (início , incremento ) ]
```

Onde:

- Início: Valor a ser utilizado para o primeiro valor inserido na coluna.
- Incremento: Valor a ser incrementado a cada nova inserção.

Os argumentos não são obrigatórios, mas quando informados, é necessário que sejam passados os 2. Se for informado apenas 1 argumento será gerado um erro

Exemplo para criação de uma tabela utilizando um campo auto incremento:

```
CREATE TABLE tbClientes
 (
     Codigo int identity(1,1),
    Nome varchar(50),
    UF varchar(2)
)
```

Nesse exemplo a tabela tbClientes possui 2 atributos, sendo Codigo tendo seu primeiro valor válido igual a 1 e se auto incrementando de 1 em 1, ou seja, primeiro registro igual a 1, o segundo igual a 2 e assim por diante.

Obs: É permitido somente um campo IDENTITY por tabela.

A inserção em tabelas que possuem campos auto incrementos deve-se suprimir da sintaxe INSERT os mesmos, conforme exemplo a seguir.

```
INSERT INTO tbClientes VALUES
('Fulano de Tal', 'SP')
```

Resultados | Mensagens

	Codigo	Nome	UF
1	1	Fulano de Tal	SP

Como foi demonstrado no exemplo, deve-se suprimir o atributo Codigo que é auto incremento para o valor do mesmo ser gerado automaticamente conforme suas configurações.

Para a sequência será preciso rodar o bloco de comandos SQL, conforme é mostrado a seguir.

```
INSERT INTO tbClientes VALUES
('João dos Santos', 'BA'),
('Maria José', 'AM'),
('Claudio Oliveira', 'SP'),
('Antonio da Silva', 'MG')
```

O resultado de um SELECT na tabela tbClientes deve ser o seguinte:

Resultados | Mensagens

	Codigo	Nome	UF
1	1	Fulano de Tal	SP
2	2	João dos Santos	BA
3	3	Maria José	AM
4	4	Claudio Oliveira	SP
5	5	Antonio da Silva	MG

Para saber o valor atual do IDENTITY, basta usar o seguinte comando:

```
SELECT IDENT_CURRENT('tbClientes')
```

Dando sequência, pressuponha que seja rodada a seguinte instrução SQL:

```
DELETE FROM tbClientes
```

Nesse caso todos os registros da tabela foram apagados, porem a sequência para auto incremento continua vigente, ou seja, o próximo registro a ser inserido terá o valor 5 para o atributo **Codigo**.

Caso for necessário, é possível "reiniciar" a sequência de auto incremento utilizando o comando abaixo:

```
DBCC Checkident( tbCliente, reseed, 0)
```

Observa-se que a sequência foi alterada para 0, sendo assim a próxima inserção terá valor igual a "1". Não é obrigado iniciar a sequência do 0, podendo alterar esse valor conforme as necessidades vigentes.

8. Chave primária e chave estrangeira

A linguagem SQL traz muitos conceitos importantes. Entre eles, os conceitos de chave primária e chave estrangeira. Tais opções são essenciais para que possamos definir, principalmente, os relacionamentos entre as entidades de uma base de dados. Diante disso, esse capítulo visa apresentar esses conceitos, colocando cada um dentro de seu escopo e mostrando como e quando utilizá-los.

8.1. Chave primária

A chave primária, ou *primary key*, é o conceito mais básico relacionado à organização em um banco de dados. Toda tabela irá possuir uma, e somente uma, chave primária. Essa chave é utilizada como o identificador único da tabela, sendo, então, representada, por aquele campo (ou campos) que não receberá valores repetidos.

Por causa disso, existe uma lista de características que deve ser levada em consideração ao definir uma chave primária:

1. Chaves primárias não podem ser nulas;
2. Cada registro na tabela deve possuir uma, e somente uma, chave primária;
3. Normalmente, chaves primárias são incrementadas automaticamente pelo banco de dados, ou seja, não há necessidade de passarmos esse valor em um INSERT. Entretanto, essa é uma opção configurada na criação da base de dados que não é obrigatória. Nos casos em que ela (incremento automático) não é definida, é preciso garantir que não haverá valores repetidos nessa coluna;
4. São as chaves para o relacionamento entre entidades, ou tabelas, da base de dados. Assim, haverá, na tabela relacionada, uma referência a essa chave primária (que será, na tabela relacionada, a chave estrangeira).

Para criarmos uma chave primária, precisamos de um código como o mostrado abaixo:

```
CREATE TABLE Pessoa
(
      ID_Pessoa INT PRIMARY KEY IDENTITY,
      Nome VARCHAR(255),
      Endereco VARCHAR(255),
      Cidade VARCHAR(255)
)
```

O campo ID_Pessoa da tabela gera um campo de valores inteiros (integer), com uma *constraint* PRIMARY KEY que indica que esse campo se trata da chave primária da tabela. Note, ainda, que o definimos como sendo de auto incremento (IDENTITY).

De forma simples, Constraints, dentro do SQL, são regras/restrições definidas para uma coluna de uma tabela do banco de dados. Podemos dizer, também, que representam propriedades que os dados de certa coluna precisam obedecer. Chaves primárias e chaves estrangeiras são exemplos de constraints.

Vale ressaltar que a chave primária é essencial para o funcionamento da base de dados, representando um registro único que facilita buscas e garante que cada valor dentro da tabela será diferente do outro.

8.2. Chave estrangeira

A chave estrangeira, ou foreign key, é um conceito ligeiramente diferente. Ela não diz respeito, especificamente, a uma tabela, mas sim a um relacionamento entre tabelas. De forma sucinta, a chave estrangeira é uma referência, em uma tabela, a uma chave primária de outra tabela. Para facilitar a compreensão, tomemos como exemplo duas tabelas: Pessoa e Carro. Para montarmos um relacionamento entre elas, poderíamos ter, na tabela Carro, o campo ID_Pessoa fazendo referência à chave primária da tabela Pessoa.

Diferentemente da chave primária, a chave estrangeira:

1. Pode ser nula (NOT NULL);
2. É um campo em uma tabela que faz referência a um campo que é chave primária em outra tabela;
3. É possível ter mais de uma (ou nenhuma) em uma tabela.

Um alerta: sobre as chaves estrangeiras aceitarem o valor null, é preciso ter cuidado. Embora não haja, efetivamente, nenhum problema com isso, tal característica pode gerar o que é chamado de

"registro órfão", isto é, um registro sem dados para um determinado relacionamento. Por exemplo, um registro de Pessoa que não possui Carro. Embora comum na realidade, é preciso levar em consideração essa regra de negócio na aplicação para evitar problemas.

A criação de chaves estrangeiras em uma tabela se dá de duas formas: a Listagem 2 mostra a adição da chave estrangeira diretamente quando criamos a tabela; já na Listagem 3, vemos a utilização do comando ALTER TABLE para inserir essa *constraint* em uma tabela já existente.

```
CREATE TABLE Carro
(
     ID_Carro INT PRIMARY KEY IDENTITY,
     Nome VARCHAR(50),
     Marca VARCHAR(50),
     CONSTRAINT fk_PesCarro FOREIGN KEY (ID_Pessoa) REFERENCES
Pessoa (ID_Pessoa)
)
```

Observe que adicionamos uma *constraint* chamada "fk_PesCarro" (nome padrão: misto dos nomes das tabelas relacionadas com o prefixo "fk") como uma FOREIGN KEY (chave estrangeira) e associamos essa foreign key ao campo ID_Pessoa da tabela Carro. Ainda na mesma linha, definimos a referência propriamente dita (palavra-chave REFERENCES) à tabela Pessoa, especificamente ao campo ID_Pessoa da tabela Pessoa.

9. Relacionamentos

Relacionamentos são informações relacionadas entre sí, mas geralmente armazenadas em diferentes tabelas ou de alguma forma em uma tabela relacionada com ela mesma.

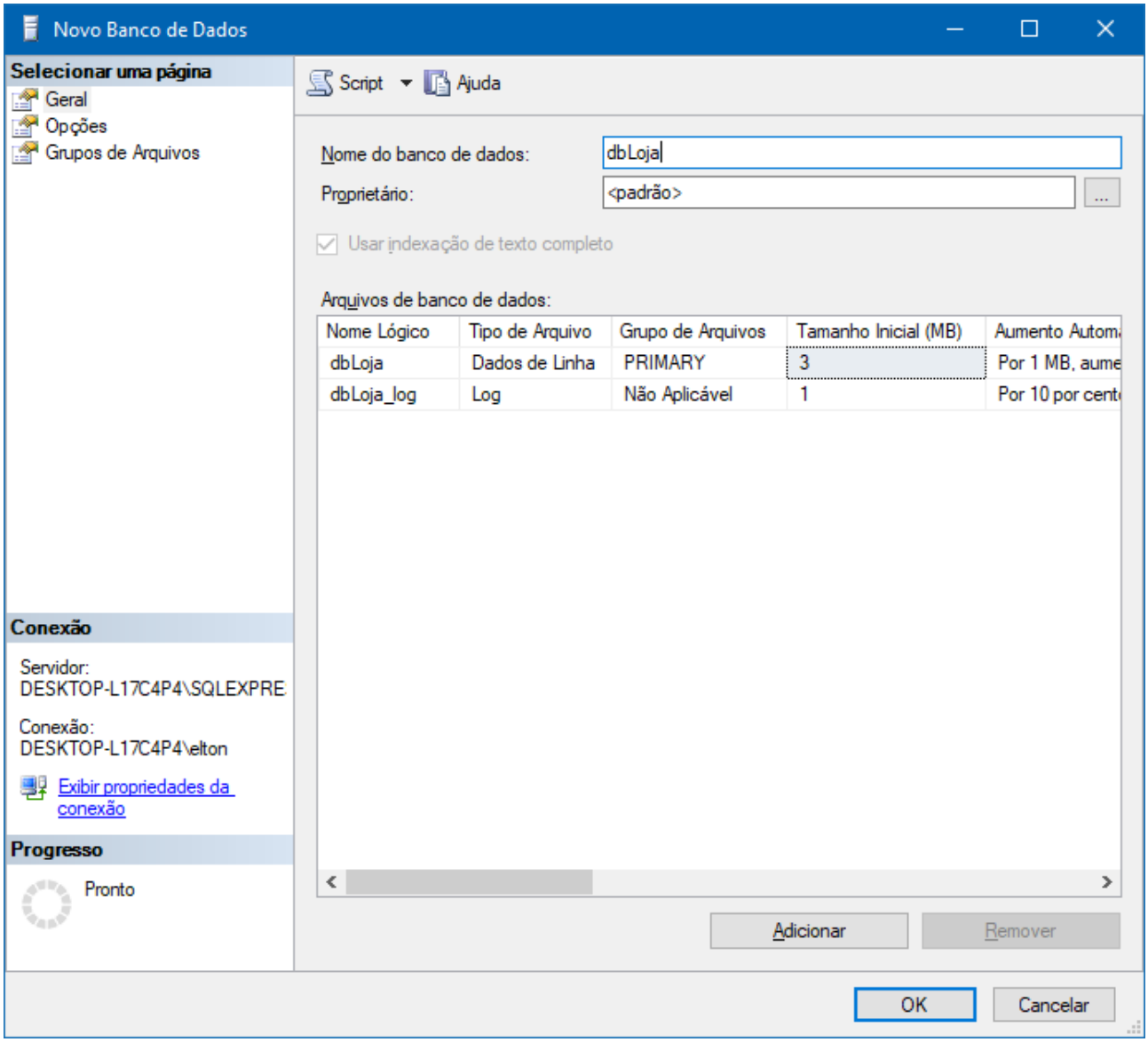
Um banco de dados é composto por diversas tabelas, como por exemplo: Clientes, Produtos, Pedidos, Detalhes do Pedido, etc. Embora as informações estejam separadas em cada uma das Tabelas, na prática devem existir ***relacionamentos*** entre as tabelas. Por exemplo: Um Pedido é feito por um Cliente e neste Pedido podem existir diversos itens, itens que são gravados na tabela Detalhes do Pedido. Além disso cada Pedido possui um número único (Código do pedido), mas um mesmo Cliente pode fazer diversos pedidos e assim por diante.

Veremos como criar de forma básica e pratica os relacionamentos edentre as tabelas utilizando o **Database Diagrams** no SQL Server Management Studio.

Primeiramente deve-se criar um banco, depois as tabelas e consequentemente os relacionamentos.

9.1. Criar o banco de dados.

Para criação do banco de dados, vamos fazer uso do assistente para a criação do banco, clicando com o botão direito do mouse sobre a pasta **Banco de dados** e escolhendo a opção **Novo Banco de Dados**..., em seguida abrirá um assistente de criação do banco, como mostra a figura abaixo:



Na opção **Nome do banco de dados** escolha o nome do banco que irá ser criado, em seguida clique em **OK**.

Antes de iniciar a criação das tabelas, nunca esqueça de entrar no contexto do banco a ser utilizado.

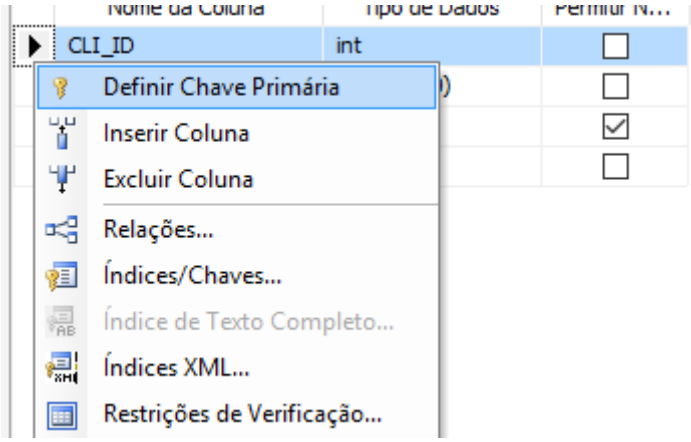
9.2. Criar as tabelas

Vamos criar nossa primeira tabela para o banco **dbLoja**, o nome da mesma será **tbCLIENTES**, expanda pasta do banco, clique com o botão direito do mouse sobre a pasta **Tabelas** e escolha a opção **Nova Tabela**, em seguida abrirá um assistente de criação da tabela, como mostra a figura:

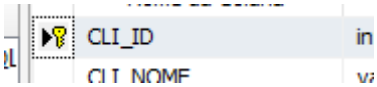
	Nome da Coluna	Tipo de Dados	Permitir N...
	CLI_ID	int	☐
	CLI_NOME	varchar(50)	☐
▶	CLI_DATANASC	datetime	☑

Uma chave primária é utilizada para identificar de forma única cada linha numa tabela. Pode fazer parte do próprio registo atual ou pode ser um campo artificial (um que não tenha nada a ver com o registo atual). Uma chave primária pode ser composta por um ou mais campos numa tabela. Quando são utilizados vários campos como chave primária, são denominados por chave composta.

Para definir uma coluna da tabela como chave primária no SQL Server, simplesmente clique com o botão direito no nome da coluna e escolha **Definir Chave Primária.**



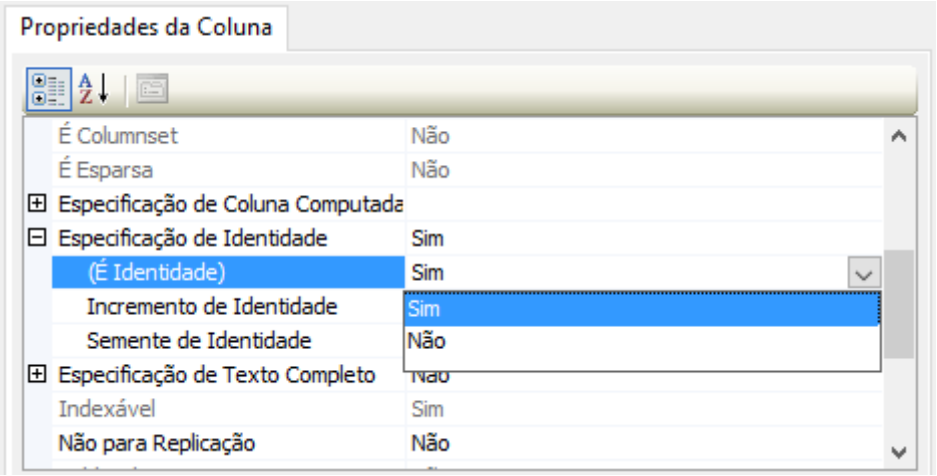
Após isso, um ícone de uma chave será indicado ao lado do nome da coluna.



Vamos criar a segunda tabela com o nome de **tbPRODUTOS** com as seguintes especificações, como mostra a abaixo:

	Nome da Coluna	Tipo de Dados	Permitir N...
🔑	PRO_ID	int	☐
	PRO_NOME	varchar(50)	☑
▶	PRO_VALOR	decimal(10, 2)	☑

Antes de concluir a criação da tabela **tbPRODUTOS**, clique sobre a coluna PRO_ID, vá até a **Propriedades da Coluna** que fica logo abaixo do assistente de criação da tabela e mude a opção **Especificação de Identidade** para **Sim**, com isso ele será auto incremento:



Crie a terceira tabela com o nome de **tbESTOQUE**, com as seguintes especificações:

Nome da Coluna	Tipo de Dados	Permitir N...
ESTOQUE_ID	int	☐
PROD_ID	int	☐
ESTOQUE_QTDE	float	☑

Antes de concluir a criação da tabela **tbESTOQUE**, mude a opção para aceitar o auto incremento como foi mostrado na tabela tbPRODUTOS.

Crie a quarta tabela com o nome de **tbPEDIDOS**, com as seguintes especificações:

Nome da Coluna	Tipo de Dados	Permitir N...
PED_ID	int	☐
CLI_ID	int	☐
PED_DATA	datetime	☑
PED_VALOR	float	☑
PED_QTDE	float	☑

Antes de concluir a criação da tabela **tbPEDIDOS**, mude a opção para aceitar o auto incremento como foi mostrado nas tabelas tbPRODUTOS e tbESTOQUE.

Crie a quinta tabela com o nome de **tbITENS_PEDIDO**, com as seguintes especificações, como mostra abaixo:

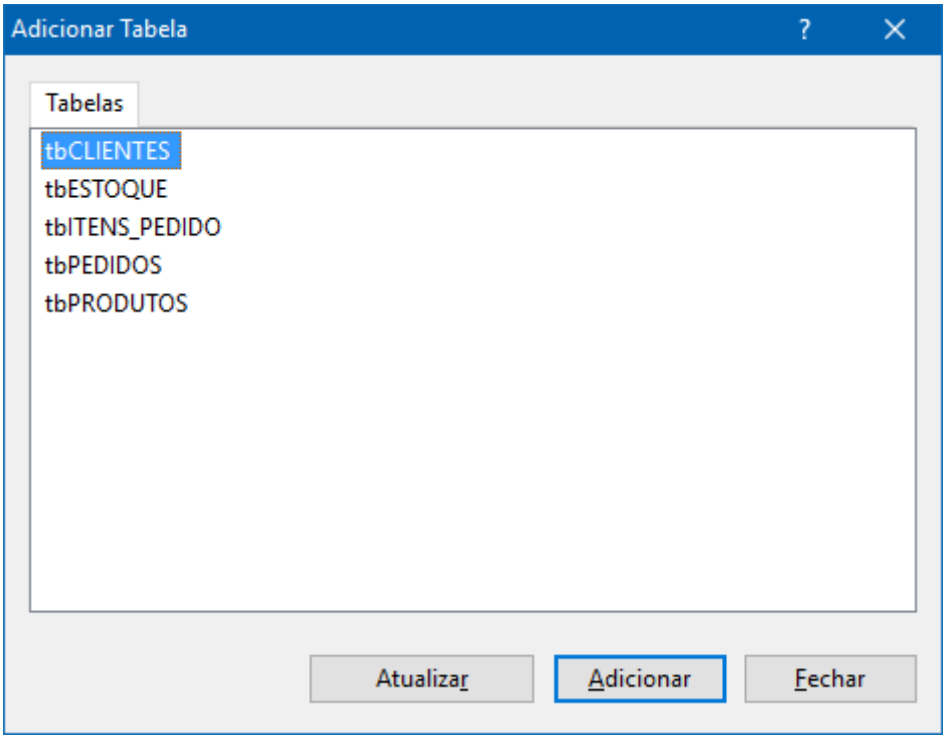
	Nome da Coluna	Tipo de Dados	Permitir N...
🔑	PED_ID	int	☐
🔑	PRO_ID	int	☐
	ITENSPED_VLRUNIT	float	☑
	ITENSPED_QTDE	float	☑
▶	ITENSPED_VLRTOTAL	float	☑

9.3. Criando relacionamentos entre as tabelas

Agora vamos criar os relacionamentos entra as tabelas.

Para dar início ao processo, no próprio Management Studio, expanda o banco onde foram criados as tabelas, em seguida com o botão direito do mouse clique sobre a pasta **Diagramas de Banco de Dados**, escolha a opção **Novo Diagramas de Banco de Dados**, antes de aparecer o assistente aparecerá uma janela, clique em SIM para dar início ao processo de criação dos relacionamentos.

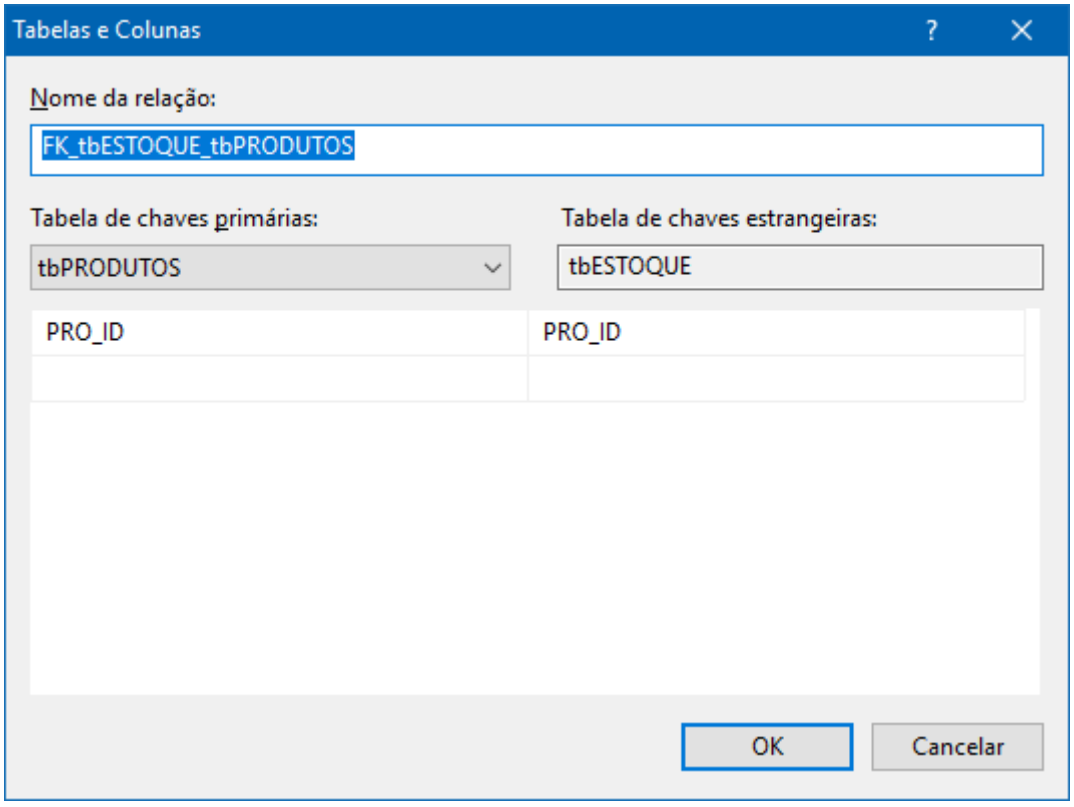
Aparecerá uma janela para você adicionar a tabelas, como mostra a figura:



Para ter uma visão ampla do seu banco de dados, adicione todas as tabelas, isso vai depender de como você queira trabalhar, veja na figura abaixo o exemplo do nosso caso:

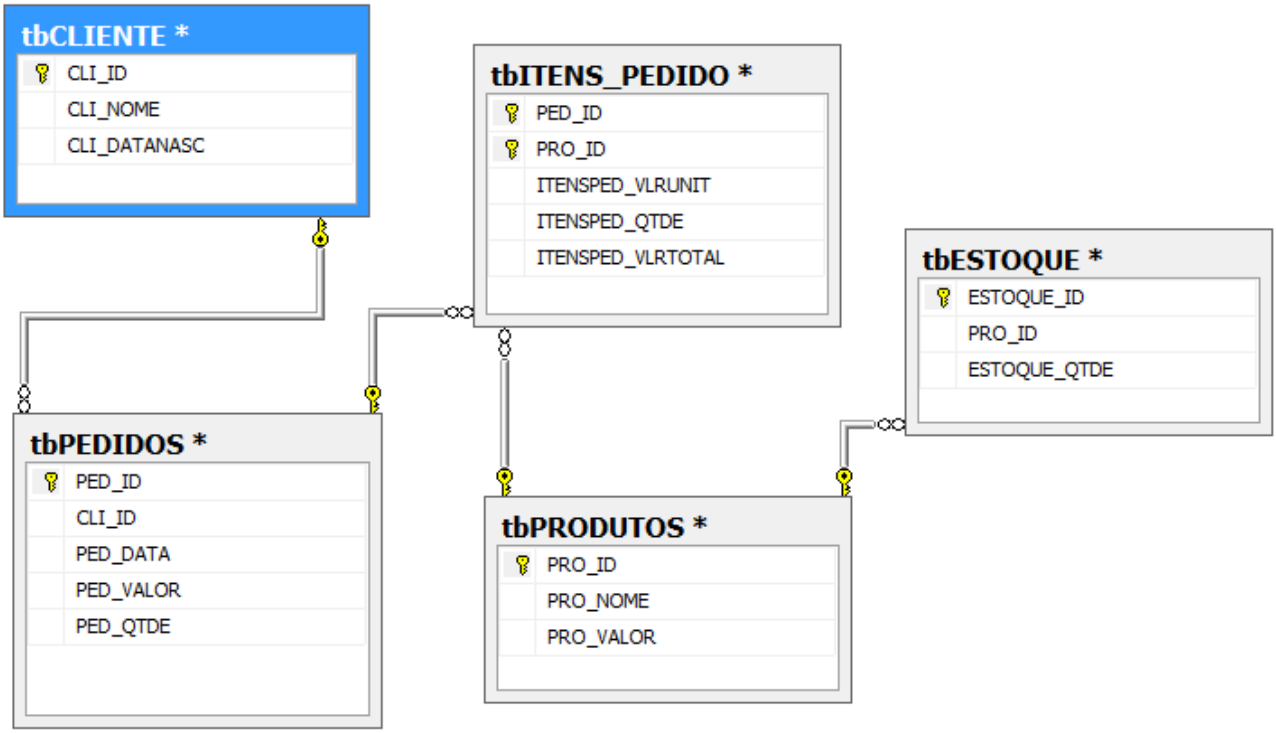
Vamos visualizar a tabela tbPRODUTOS, como mostra acima, o campo PRO_ID também existe na tabela tbESTOQUE, ou seja, o campo PRO_ID na tabela tbESTOQUE será a chave estrangeira desse relacionamento, neste caso, clique nesse campo e arraste com o mouse até o campo PRO_ID na tabela tbPRODUTOS.

Depois que você clicar e arrastar aparecerá um assistente, criando um relacionamento e informando os campos e tabelas inseridos e quem estará com a chave primaria e estrangeira, como mostra a figura:



Para concluir clique em OK, agora existe um relacionamento entre as duas tabelas.

Finalize os relacionamentos como mostra a figura abaixo:



10. Consultas Avançadas

Uma consulta é uma solicitação para obter dados que estão armazenados no SQL Server. Embora as consultas tenham vários modos de interação com um usuário, todas elas realizam a mesma tarefa: elas apresentam o conjunto de resultados de uma instrução SELECT para o usuário.

Ainda existem algumas cláusulas para esse comando que são utilizadas para recuperar as informações armazenadas em um banco de dados.

Tendo como base a tabela abaixo, vamos realizar as seguintes consultas:

Tabela Funcionários

Nome da Coluna	***Tipo de dados***
Id	Int Identity
Nome	Varchar (50) not null
Sobrenome	Varchar (50) not null
Salario	Int not null
Sexo	Char not null
Cidade	Varchar (50) not null

Insira os seguintes registros na tabela:

Id	Nome	Sobrenome	Salario	Sexo	Cidade
1	Charles	Xavier	6890	M	Nova York
2	Tony	Stark	11876	M	Nova York
3	Wanda	Maximoff	5300	F	Miami
4	Scott	Summers	6800	M	Los Angeles
5	Bruce	Banner	7800	M	Woshington
6	Ororo	Monroe	3700	F	Nova York
7	Anna	Marie	2500	F	Los Angeles
8	Steven	Rogers	7840	M	Nova York
9	Barry	Allen	3400	M	Gotham City
10	Natasha	Romanoff	4700	F	Los Angeles
11	Bruce	Wayne	3300	M	Gotham City

10.1. ORDER BY

Quando precisamos que o resultado de nossa consulta a uma tabela seja fornecido em uma ordem específica, de acordo com um determinado critério, devemos usar a cláusula ORDER BY que, como o próprio nome diz, considera uma certa ordem para retornar os dados de uma consulta. A sintaxe padrão é a seguinte:

```
SELECT coluna1, coluna2 FROM nomedatabela ORDER BY coluna1
```

10.1.1. Ordenando por uma coluna

Usaremos os exemplos a seguir usando a tabela **Funcionarios**:

```
SELECT Nome, Sobrenome, Salario FROM Funcionarios ORDER BY Nome
```

O comando acima determina que as colunas **Nome**, **Sobrenome** e **Salario** sejam selecionadas. Além disso, a exibição dos valores da coluna **Nome** é retornada em ordem alfabética, como podemos ver:

	Nome	Sobrenome	Salario
1	Anna	Marie	2500
2	Barry	Allen	3400
3	Bruce	Banner	7800
4	Bruce	Wayne	3300
5	Charles	Xavier	6890
6	Natasha	Romanoff	4700
7	Ororo	Monroe	3700
8	Scott	Summers	6800
9	Steven	Rogers	7840
10	Tony	Stark	11876
11	Wanda	Maximoff	5300

10.1.2. Ordenando por várias colunas

É possível usar a cláusula ORDER BY para ordenar os dados por várias colunas, com base nos nomes das colunas e nas posições das mesmas no SELECT. Com o comando a seguir conseguimos isso:

```
SELECT Nome, Sexo, Cidade FROM Funcionarios ORDER BY Sexo, Cidade
```

O resultado será esse:

	Nome	Sexo	Cidade
1	Anna	F	Los Angeles
2	Natasha	F	Los Angeles
3	Wanda	F	Miami
4	Ororo	F	Nova York
5	Barry	M	Gotham City
6	Bruce	M	Gotham City
7	Scott	M	Los Angeles
8	Charles	M	Nova York
9	Tony	M	Nova York
10	Steven	M	Nova York
11	Bruce	M	Woshington

10.1.3. ORDER BY ASC e DESC

A cláusula ORDER BY pode ser utilizada com as opções ASC e DESC, descritas abaixo:

- **ASC** – Quando utilizada, esta opção faz com que as linhas sejam retornadas em ordem ascendente.
- **DESC** – Quando utilizada, esta opção faz com que as linhas sejam retornadas em ordem descendente.

Caso não especifiquemos ASC ou DESC, os dados da tabela serão retornados em ordem ascendente, que é o valor padrão.

Veja abaixo o uso das opções:

ASC – Executemos a seguinte instrução:

```
SELECT Nome, Sobrenome, Salario FROM Funcionarios ORDER BY
Salario ASC
```

E veremos o seguinte resultado:

	Nome	Sobrenome	Salario
1	Anna	Marie	2500
2	Bruce	Wayne	3300
3	Barry	Allen	3400
4	Ororo	Monroe	3700
5	Natasha	Romanoff	4700
6	Wanda	Maximoff	5300
7	Scott	Summers	6800
8	Charles	Xavier	6890
9	Bruce	Banner	7800
10	Steven	Rogers	7840
11	Tony	Stark	11876

A opção ASC também pode ser usada para ordenar os dados de uma tabela conforme duas colunas diferentes. Para isso, devemos usar a seguinte instrução:

```
SELECT Nome, Sexo, Salario FROM Funcionarios ORDER BY Sexo ASC,
Salario ASC
```

Iremos obter o seguinte resultado:

	Nome	Sexo	Salario
1	Anna	F	2500
2	Ororo	F	3700
3	Natasha	F	4700
4	Wanda	F	5300
5	Bruce	M	3300
6	Barry	M	3400
7	Scott	M	6800
8	Charles	M	6890
9	Bruce	M	7800
10	Steven	M	7840
11	Tony	M	11876

DESC – A instrução a seguir demonstra o uso da opção DESC junto ao ORDER BY:

```
SELECT Id, Nome, Sobrenome, Cidade FROM Funcionarios ORDER BY Id DESC
```

O resultado será esse:

	Id	Nome	Sobrenome	Cidade
1	11	Bruce	Wayne	Gotham City
2	10	Natasha	Romanoff	Los Angeles
3	9	Barry	Allen	Gotham City
4	8	Steven	Rogers	Nova York
5	7	Anna	Marie	Los Angeles
6	6	Ororo	Monroe	Nova York
7	5	Bruce	Banner	Woshington
8	4	Scott	Summers	Los Angeles
9	3	Wanda	Maximoff	Miami
10	2	Tony	Stark	Nova York
11	1	Charles	Xavier	Nova York

Assim como o ASC, a opção DESC também pode ser adotada para ordenar os dados de uma tabela conforme duas ou mais colunas diferentes. Para isso, podemos usar o seguinte comando:

```
SELECT Nome, Salario, Sexo FROM Funcionarios ORDER BY Sexo DESC,
Salario DESC
```

O resultado será esse:

	Nome	Salario	Sexo
1	Tony	11876	M
2	Steven	7840	M
3	Bruce	7800	M
4	Charles	6890	M
5	Scott	6800	M
6	Barry	3400	M
7	Bruce	3300	M
8	Wanda	5300	F
9	Natasha	4700	F
10	Ororo	3700	F
11	Anna	2500	F

10.2. Cláusula TOP

Como resultado de uma consulta a uma tabela de banco dados, temos a opção de retornar a quantidade de linhas desejada, a partir da primeira linha selecionada. Para isso, usamos a cláusula TOP. Usando a tabela Funcionarios como exemplo, usamos a seguinte instrução para retornar as cinco primeiras linhas desta tabela:

```
SELECT TOP 5 * FROM Funcionarios
```

O resultado será esse:

Id	Nome	Sobrenome	Salario	Sexo	Cidade
1	Charles	Xavier	6890	M	Nova York
2	Tony	Stark	11876	M	Nova York
3	Wanda	Maximoff	5300	F	Miami
4	Scott	Summers	6800	M	Los Angeles
5	Bruce	Banner	7800	M	Woshington

10.2.1. Cláusula TOP com ORDER BY

As cláusulas TOP e ORDER BY podem e devem ser utilizadas de forma conjunta. Usando novamente a tabela Funcionarios como exemplo, vamos exibir os três primeiros Nomes de menor Salario, dentre a relação de Nomes da lista. Usamos a seguinte instrução:

```
SELECT TOP 3 * FROM Funcionarios ORDER BY Salario ASC
```

Que nos resultará nisso:

Id	Nome	Sobrenome	Salario	Sexo	Cidade
7	Anna	Marie	2500	F	Los Angeles
11	Bruce	Wayne	3300	M	Gotham City
9	Barry	Allen	3400	M	Gotham City

Da mesma forma, se desejar retornar os quatro Salarios mais altos da tabela, por exemplo, usa-se a seguinte instrução:

```
SELECT TOP 4 * FROM Funcionarios ORDER BY Salario DESC
```

Que nos resultará nesses valores:

Id	Nome	Sobrenome	Salario	Sexo	Cidade
2	Tony	Stark	11876	M	Nova York
8	Steven	Rogers	7840	M	Nova York
5	Bruce	Banner	7800	M	Woshington
1	Charles	Xavier	6890	M	Nova York

10.2.3. Cláusula TOP WITH TIES com ORDER BY

Permitida apenas em instruções SELECT e quando uma cláusula ORDER BY é especificada, a cláusula TOP WITH TIES determina que linhas adicionais sejam retornadas a partir do conjunto de resultados base com o mesmo valor apresentado nas colunas ORDER BY, sendo exibidas como as últimas das linhas **TOP n** (PERCENT).

Considerando nossa tabela de Funcionarios, vamos exibir seus dados em ordem crescente conforme a coluna Salario. Isso é feito pela seguinte instrução:

```
SELECT * FROM Funcionarios ORDER BY Salario
```

O resultado é esse:

Id	Nome	Sobrenome	Salario	Sexo	Cidade
7	Anna	Marie	2500	F	Los Angeles
11	Bruce	Wayne	3300	M	Gotham City
9	Barry	Allen	3400	M	Gotham City
6	Ororo	Monroe	3700	F	Nova York
10	Natasha	Romanoff	4700	F	Los Angeles
3	Wanda	Maximoff	5300	F	Miami
4	Scott	Summers	6800	M	Los Angeles
1	Charles	Xavier	6890	M	Nova York
5	Bruce	Banner	7800	M	Woshington
8	Steven	Rogers	7840	M	Nova York
2	Tony	Stark	11876	M	Nova York

Vamos supor agora que precisamos obter como resultado o Salario com o menor valor. Porém, é preciso considerar a existência de Salario com o mesmo valor. Neste caso, executamos a seguinte instrução para retornar os Salários com o menor valor:

```
SELECT TOP 1 WITH TIES * FROM Funcionarios ORDER BY Salario ASC
```

O resultado será esse:

Id	Nome	Sobrenome	Salario	Sexo	Cidade
6	Ororo	Monroe	3300	F	Nova York
9	Barry	Allen	3300	M	Gotham City
11	Bruce	Wayne	3300	M	Gotham City

Podemos observar na imagem acima que existem três pessoas com o mesmo Salário, ou seja, todos eles representam o menor valor da tabela.

Neste exemplo, além das cláusulas ORDER BY e TOP, já descritas anteriormente, a cláusula WITH TIES retorna a primeira linha da tabela, além de todas as linhas que apresentam o valor idêntico ao valor do Salário que a cláusula TOP 1 selecionou.

Da mesma forma, por exemplo, se quisermos retornar um resultado com os dois Salários que possuem maior valor, seguido dos salários que apresentam o mesmo valor da segunda pessoa, usamos a seguinte instrução:

```
SELECT TOP 2 WITH TIES * FROM Funcionarios ORDER BY Salario DESC
```

Que nos resultará nisso:

Id	Nome	Sobrenome	Salario	Sexo	Cidade
2	Tony	Stark	11876	M	Nova York
5	Bruce	Banner	7800	M	Woshington
8	Steven	Rogers	7800	M	Nova York

10.3. Operador BETWEEN

Esse operador é usado quando precisamos recuperar as linhas de uma tabela cujo valor de um campo encontra-se em um intervalo especificado. Esse tipo de consulta é muito comum quando queremos filtrar os dados por intervalos de datas, por exemplo, para retornar os registros de um determinado período. Sua sintaxe é a seguinte:

```
SELECT campos FROM tabela WHERE campo BETWEEN inicio_intervalo AND
fim_intervalo
```

O seguinte script retorna os registros da tabela Funcionarios cujo Salario esteja entre 3000 e 6000:

```
SELECT * FROM Funcionarios WHERE Salario BETWEEN 3000 AND 6000
```

Id	Nome	Sobrenome	Salario	Sexo	Cidade
3	Wanda	Maximoff	5300	F	Miami
6	Ororo	Monroe	3700	F	Nova York
9	Barry	Allen	3400	M	Gotham City
10	Natasha	Romanoff	4700	F	Los Angeles
11	Bruce	Wayne	3300	M	Gotham City

O operador NOT BETWEEN seleciona os valores fora do intervalo definido.

Para o exemplo vamos agora selecionar os funcionários com salários fora do intervalo 2000 e 4000.

```
SELECT * FROM Funcionarios WHERE Salario NOT BETWEEN 2000 AND 4000
```

Que nos resultará nisso:

Id	Nome	Sobrenome	Salario	Sexo	Cidade
1	Charles	Xavier	6890	M	Nova York
2	Tony	Stark	11876	M	Nova York
3	Wanda	Maximoff	5300	F	Miami
4	Scott	Summers	6800	M	Los Angeles
5	Bruce	Banner	7800	M	Woshington
8	Steven	Rogers	7800	M	Nova York
10	Natasha	Romanoff	4700	F	Los Angeles

11. Funções de agrupamento

Funções de grupo operam conjuntos de linhas visando a fornecer um resultado para o grupo. Até o momento trabalhamos apenas com funções que tratavam apenas uma linha de cada vez. A diferença básica é que serão utilizados grupos de linhas. Esses grupos podem ser constituídos desde toda a tabela até subgrupos da tabela.

Existem diversas funções de grupo que são implementadas pelo padrão SQL. Essas funções auxiliam a computar uma variedade de medidas baseadas em valores das colunas do banco de dados. As principais funções de grupo são:

Função	***Ação***
COUNT	Retorna o número de linhas afetadas pelo comando.
SUM	Retorna o somatório do valor das colunas especificadas.
AVG	Retorna a média aritmética dos valores das colunas.
MIN	Retorna o menor valor da coluna de um grupo de linhas.
MAX	Retorna o maior valor da coluna de um grupo de linhas.

11.1. COUNT

Diferentemente das outras funções de grupo, o COUNT retorna o número de linhas que atende a uma determinada condição. Podemos utilizá-lo com um asterisco entre parênteses, para indicar que queremos saber a quantidade total de linhas, independentemente de haver linhas com colunas nulas ou não.

Casos queiramos saber quantas linhas existem e quais destas não têm valor nulo em determinada coluna, especificamos essa coluna entre parênteses.

Vamos usar como base a tabela FILMES:

codigo	titulo	personagem	interprete	ano
1	De Volta para o Futuro	Emmet Brown	Christopher Lloyd	1985
2	Star Wars	Luke Skywalker	Mark Hamill	1977
3	Donnie Darko	Donnie Darko	NULL	NULL
4	Rodky	Rocky Balboa	NULL	1976
5	A Identidade Bourne	Jason Bourne	Matt Damon	2002
6	Jogos Mortais	Jigsaw	Tobin Bell	2004
7	Matrix	Neo	Keanu Reeves	1999
8	O Senhor dos Anéis	Gollum	Andy Serkis	NULL
9	Piratas do Caribe	Jack Sparrow	NULL	2003
10	Os Caçadores da Arca Perdida	Indiana jones	Harrison Ford	1981
11	Star Wars	Darth Vader	David Prowse	1977
12	Clube da Luta	Tyler Durden	Brad Pitt	NULL
13	O Silêncio dos Inocentes	Clarice Starling	Jodie Foster	1991
14	Matrix	Agente Smith	Hugo Weaving	1999
15	Psicose	Norman Bates	NULL	NULL
16	O Senhor dos Anéis	Aragorn	Viggo Mortensen	2001

A função COUNT(*) retorna o número de registros (linhas) de uma tabela.

SINTAXE

```
SELECT COUNT(*) FROM nome_da_tabela
```

Para exibir o total de registros na tabela FILMES, vamos executar a sentença abaixo:

```
SELECT COUNT(*) FROM FILMES
```

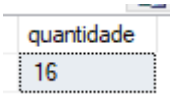
Após a execução do comando, teremos o resultado, conforme exibido na imagem a seguir:

	(Nenhum nome de coluna)
1	16

Repare que a coluna não recebeu nenhum nome. Caso seja preciso alterar o nome da coluna, por exemplo, para "quantidade", crie uma alias "apelido". Para isso coloque um AS após a função `COUNT(*)`.

```
SELECT COUNT(*) AS quantidade FROM FILMES
```

Após a execução do comando, o nome da coluna será exibido como "quantidade", veja a imagem abaixo:



quantidade
16

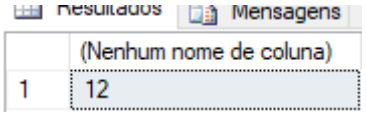
A função `COUNT(nome_da_coluna)` retorna o número valores de uma coluna. Valores nulos não entram na contagem.

SINTAXE

```
SELECT COUNT(nome_da_coluna) FROM nome_da_tabela
```

Aqui o total de registros que possuem na coluna **ano**:

```
SELECT COUNT(ano) FROM FILMES
```



Resultados | Mensagens

	(Nenhum nome de coluna)
1	12

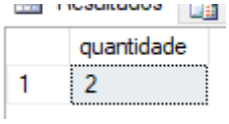
Efeito semelhante poderíamos ter com o comando:

```
SELECT COUNT(*) FROM FILMES WHERE ano IS NOT NULL
```

Suponhamos que queira exibir a quantidade de registros onde o título seja "O Senhor dos Anéis". Para fazer a contagem, executamos a sentença abaixo:

```
SELECT COUNT(titulo) AS quantidade FROM FILMES WHERE titulo = 'O Senhor
dos Anéis'
```

Após a execução do comando, teremos o resultado, conforme exibido na imagem a seguir:



	quantidade
1	2

Para o próximo exemplo, vamos utilizar a tabela SERVICOS como mostra abaixo:

codOrdem	codServico	dataInicial	dataFinal
34329	A	2015-11-10	2015-11-12
34330	A	2015-12-06	2015-12-08
34331	A	2015-01-05	2016-01-06
34332	A	2016-01-30	NULL
34333	B	2015-11-15	2015-11-17
34334	B	2015-12-18	2015-12-19
34335	B	2015-01-05	2016-01-06
34336	B	2016-01-28	NULL
34337	C	2015-11-08	2015-11-11
34338	C	2015-12-05	2015-12-08
34339	C	2015-01-15	2016-01-17
34340	C	2016-01-27	NULL

O primeiro passo é exibir a quantidade de ordens de serviços que já foram finalizadas.

As ordens de serviços finalizadas são aquelas que possuem uma data de encerramento. Logo vamos utilizar a função `COUNT` na coluna dataFinal

Para fazer a contagem, executamos a sentença abaixo:

```
SELECT COUNT(dataFinal) AS quantidade FROM SERVICOS
```

Após a execução do comando, teremos o resultado, conforme exibido na imagem a seguir:

	quantidade
1	9

Observação: perceba que os valores nulos não são incluídos na contagem.

Próximo exemplo é exibir a quantidade de ordens de serviços que não foram finalizadas, que são aquelas que ainda não possuem data de encerramento, logo vamos fazer a contagem dos registros onde o valor a coluna "dataFinal" seja nulo.

Como precisamos contar valores nulos, vamos utilizar "`COUNT(*)`", pois COUNT(nome_da_coluna) ignora valores nulos.

```
SELECT COUNT(*) AS quantidade FROM SERVICOS WHERE dataFinal IS NULL
```

Após a execução do comando, teremos o resultado, conforme exibido na imagem a seguir:

quantidade
3

Para exibir a distribuição por tipo de serviço vamos agrupar os dados da coluna "codServico" através do comando "`group by`".

```
SELECT codServico,COUNT(codServico) AS quantidade FROM SERVICOS
GROUP BY codServico
```

Após a execução do comando, teremos o resultado, conforme exibido na imagem a seguir:

codServico	quantidade
A	4
B	4
C	4

11.2. SUM

Essa função, também como o nome em inglês já sugere, efetua a soma de valores. Sua sintaxe também é idêntica às das outras funções:

```
SELECT SUM(nome_da_coluna)FROM nome_da_tabela
```

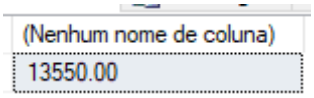
Para o entendimento dessa função, vamos utilizar a tabela PRODUTOS como mostra a imagem abaixo:

id	nome	fabricante	quantidade	vlUnitario	tipo
1	Playstation 4	Sony	100	1999.00	Console
2	Core i5 8GB Ram 1TB HD	Dell	200	1899.00	Notebook
3	XBOX One	Microsoft	350	1699.00	Console
4	GT-I6220 Quad Band	Samsung	300	499.00	Smartphone
5	iPhone 4 32GB	Apple	50	1499.00	Smartphone
6	Playstation 3	Sony	100	399.00	Console
7	Moto G3	Motorola	200	899.00	Smartphone
8	Core i7 16GB Ram 2TB HD	HP	50	2359.00	Notebook
9	XBOX 360 120GB	Microsoft	200	1299.00	Console
10	Wii 120GB	Nintendo	250	999.00	Console

Vamos a um exemplo que soma os preços de todos os produtos cadastrados nessa tabela.

```
SELECT SUM(vlUnitario) FROM PRODUTOS
```

O resultado será esse abaixo:

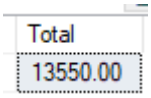


(Nenhum nome de coluna)
13550.00

Repare que, assim como a função COUNT, a coluna não recebeu nenhum nome. Caso seja preciso alterar o nome da coluna, por exemplo, para "total", pode-se usar o parâmetro AS após a função SUM.

```
SELECT SUM(vlUnitario) AS Total FROM PRODUTOS
```

Após a execução do comando, teremos o resultado, conforme exibido na imagem a seguir:



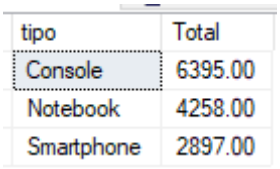
Total
13550.00

Suponha que queira calcular o total dos produtos por "tipo". Antes de efetuarmos a soma, devemos agrupar os tipos que estão armazenados na coluna "tipo". Para agruparmos utilizaremos o "GROUP BY".

Para calcularmos a soma por segmento, executamos a sentença abaixo:

```
SELECT tipo, SUM(vlUnitario) AS Total FROM PRODUTOS GROUP BY
tipo
```

Após a execução, teremos o total dos produtos agrupado por tipo. Conforme podemos visualizar na imagem abaixo:



tipo	Total
Console	6395.00
Notebook	4258.00
Smartphone	2897.00

Para ordenar o total em ordem crescente, ou seja, do valor menor para o maior, deve-se usar "ORDER BY" seguido do nome da coluna, neste caso a coluna ordenada será 'total'.

Para ordenar os valores, utilizamos a sentença abaixo:

```
SELECT tipo, SUM(vlUnitario) AS Total FROM PRODUTOS GROUP BY
tipo ORDER BY Total
```

Após a execução, teremos os valores em ordem crescente. Conforme podemos visualizar na imagem abaixo:

tipo	Total
Smartphone	2897.00
Notebook	4258.00
Console	6395.00

Para ordenar o total em ordem decrescente, ou seja, do maior valor para o menor devemos acrescentar o "DESC" depois do "ORDER BY".

Para ordenar os totais em ordem decrescente, utilizamos a sentença abaixo:

```
SELECT tipo, SUM(vlUnitario) AS Total FROM PRODUTOS GROUP BY
tipo ORDER BY Total DESC
```

Após a execução, teremos os valores totais em ordem decrescente. Conforme podemos visualizar na imagem abaixo:

tipo	Total
Console	6395.00
Notebook	4258.00
Smartphone	2897.00

Vamos calcular os valores, mas agora considerando a quantidade de produtos.

Dentro da função "SUM" devemos fazer a multiplicação da quantidade pelo valor do produto.

Exemplo: foi gasto R$ 497.750,00 com o “Notebook”.

Para calcularmos a multiplicação por segmento, executamos a sentença abaixo:

```
SELECT tipo, SUM(quantidade * vlUnitario) AS Total FROM PRODUTOS
GROUP BY tipo ORDER BY Total
```

Após a execução, teremos os totais por tipo. Conforme podemos visualizar na imagem abaixo:

tipo	Total
Smartphone	404450.00
Notebook	497750.00
Console	1344000.00

Vamos supor que queiramos exibir a multiplicação da quantidade e o valor unitário de cada produto.

Veja a sentença abaixo:

```
SELECT
nome, quantidade, vlUnitario,
SUM(quantidade * vlUnitario) AS Total
FROM PRODUTOS
GROUP BY nome, quantidade, vlUnitario
```

Após a execução, serão exibidos todos os produtos da com o valor total de cada um. Veja abaixo:

nome	quantidade	vlUnitario	Total
Core i5 8GB Ram 1TB HD	200	1899.00	379800.00
Core i7 16GB Ram 2TB HD	50	2359.00	117950.00
GT-I6220 Quad Band	300	499.00	149700.00
iPhone 4 32GB	50	1499.00	74950.00
Moto G3	200	899.00	179800.00
Playstation 3	100	399.00	39900.00
Playstation 4	100	1999.00	199900.00
Wii 120GB	250	999.00	249750.00
XBOX 360 120GB	200	1299.00	259800.00
XBOX One	350	1699.00	594650.00

11.3. AVG

A função AVG retorna a média de valores de uma coluna.

```
SELECT AVG(nome_da_coluna) FROM nome_da_tabela
```

Para os próximos exemplos, utilizaremos a tabela "FORNECEDOR". Veja a imagem abaixo:

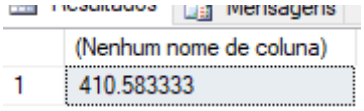
id	produto	segmento	data	valor	quantidade
1	Pen Drive 4GB	Papelaria e informática	2013-09-12	22.00	10
3	HD 500 GB	Papelaria e informática	2013-10-20	300.00	2
4	Tonner	Papelaria e informática	2013-11-01	250.00	2
5	Cadeira	Marcenaria	2013-09-19	50.00	3
6	Mesa	Marcenaria	2013-10-21	600.00	1
7	Armário	Marcenaria	2013-09-12	900.00	1
8	Corrimão	Serralheria	2013-09-12	400.00	1
9	Portão	Serralheria	2013-10-22	1500.00	1
10	Grade de proteção para janelas	Serralheria	2013-11-03	800.00	2
11	Detergente	Limpeza e higiêne	2013-09-20	5.00	3
12	Desinfetante	Limpeza e higiêne	2013-11-23	40.00	5
13	Papel toalha	Limpeza e higiêne	2013-11-04	60.00	2

Suponha que queira calcular a média das despesas de uma empresa. O valor das despesas estão armazenadas na coluna valor, logo devemos calcular a média dos valores desta coluna.

Para calcularmos a média, executamos a sentença abaixo:

```
SELECT AVG(valor) FROM FORNECEDOR
```

Após a execução, teremos a média com as despesas. Conforme podemos visualizar na imagem abaixo:

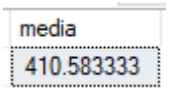


O nome da coluna aparece como "Nenhum nome de coluna", mas vamos supor que precisamos que seja exibido "media".

Podemos modificar o nome desta coluna, ou seja, criar um alias (apelido). Colocamos o alias depois do "AS"

```
SELECT AVG(valor) AS media FROM FORNECEDOR
```

Após a execução, o nome da coluna será exibido como "media".



Podemos, também, calcular a média das despesas por seguimento, ou seja:

- A média dos gastos com o segmento Papelaria e informática
- A média dos gastos com o segmento Marcenaria
- A média dos gastos com o segmento Serralheria
- A média dos gastos com o segmento Limpeza e higiene

Antes de efetuarmos a média, devemos agrupar os segmentos que estão armazenados na coluna "segmento". Para agruparmos utilizaremos o " GROUP BY".

Para calcularmos a média por segmento, executamos a sentença abaixo:

```
SELECT segmento, AVG(valor) AS media FROM FORNECEDOR GROUP BY
segmento
```

Após a execução, teremos a média dos gastos agrupados por segmento. Conforme podemos visualizar na imagem abaixo:

segmento	media
Limpeza e higiêne	35.000000
Marcenaria	516.666666
Papelaria e informática	190.666666
Serralheria	900.000000

Para ordenar as médias em ordem crescente, ou seja, do valor menor para o maior, deve-se usar "ORDER BY" seguido do nome da coluna, neste caso, a coluna 'media' será ordenada.

Para ordenar as médias, utilizamos a sentença abaixo:

```
SELECT segmento, AVG(valor) AS media FROM FORNECEDOR GROUP BY
segmento ORDER BY media
```

Após a execução, teremos as médias em ordem crescente. Conforme podemos visualizar na imagem abaixo:

segmento	media
Limpeza e higiêne	35.000000
Papelaria e informática	190.666666
Marcenaria	516.666666
Serralheria	900.000000

Para ordenar as médias em ordem decrescente, ou seja, do maior valor para o menor devemos acrescentar o "DESC " depois do "ORDER BY".

Para ordenar as médias em ordem decrescente, utilizamos a sentença abaixo:

```
SELECT segmento, AVG(valor) AS media FROM FORNECEDOR GROUP BY
segmento ORDER BY media DESC
```

Após a execução, teremos as médias em ordem decrescente. Conforme podemos visualizar na imagem abaixo:

segmento	media
Serralheria	900.000000
Marcenaria	516.666666
Papelaria e informática	190.666666
Limpeza e higiêne	35.000000

vamos calcular as médias das despesas por segmento, mas agora considerando a quantidade de produtos.

dentro da função "AVG" devemos fazer a multiplicação da quantidade pelo valor do produto. Exemplo: a média com o segmento de "Limpeza e higiene", foi de 111.6666666666666667, pois compramos

- R$ 15,00 de detergente (3 * 5,00);
- R$ 200,00 de desinfetante (5 * 40,00);
- R$ 120,00 de papel toalha (2 * 60,00);

Média para o segmento "Limpeza e higiene":

(15,00 + 200,00 + 120,00) / 3 = 111.6666666666666667

Para calcularmos a média por segmento, executamos a sentença abaixo:

```
SELECT segmento, AVG(quantidade * valor) AS media FROM
FORNECEDOR GROUP BY segmento ORDER BY media
```

Após a execução, teremos as médias por segmento. Conforme podemos visualizar na imagem abaixo:

segmento	media
Limpeza e higiêne	111.666666
Papelaria e informática	440.000000
Marcenaria	550.000000
Serralheria	1166.666666

Vamos calcular a média das despesas por mês. Neste caso, para "setembro / 2013", "outubro / 2013" e "novembro / 2013".

O primeiro passo é saber como extrair o mês e o ano de uma data. Para extrair o mês e o ano de uma data podemos utilizar a função "date_part".

Exemplo da utilização de date_part:

```
SELECT data, datepart(month, data) AS mes, datepart(year, data)
AS ano FROM FORNECEDOR
```

Significado:

- `datepart(month, data)` : extraia o mês (month, em inglês é mês) da coluna data;
- `AS mes`: a coluna com o mês se chamará mês (alias);
- `datepart(year, data)` : extraia o ano (year, em inglês é ano) da coluna data;
- `AS ano`: a coluna com o ano se chamará ano (alias);

Após a execução, teremos uma coluna com o ano, e uma coluna com o mês. Conforme podemos visualizar na imagem abaixo:

data	mes	ano
2013-09-12	9	2013
2013-10-20	10	2013
2013-11-01	11	2013
2013-09-19	9	2013
2013-10-21	10	2013
2013-09-12	9	2013
2013-09-12	9	2013
2013-10-22	10	2013
2013-11-03	11	2013
2013-09-20	9	2013
2013-11-23	11	2013
2013-11-04	11	2013

Vamos utilizar essa função para fazer o cálculo das médias das despesas para os meses de setembro, outubro e novembro de 2013.

11.4. MAX

A função MAX retorna o valor máximo (maior valor) de uma coluna.

SINTAXE

```
SELECT MAX(nome_da_coluna) FROM nome_da_tabela;
```

Utilizaremos a tabela "IMOVEIS". Veja a imagem abaixo:

idCorretor	idImovel	valor
120	5421	180000.00
124	5421	90000.00
122	5421	130000.00
124	5421	512000.00
120	5421	70000.00
122	5421	622000.00
120	5421	127000.00

Qual o imóvel que possui o valor mais alto?

O valor dos imóveis estão armazenados na coluna valor, logo devemos calcular o valor mais alto a partir desta coluna.

Para calcularmos o valor máximo, executamos a sentença abaixo:

```
SELECT MAX(valor) FROM IMOVEIS
```

Após a execução, teremos o imóvel com o valor mais alto. Conforme podemos visualizar na imagem abaixo:

(Nenhum nome de coluna)
622000.00

Assim como as funções anteriores, podemos modificar o nome desta coluna, utilizando o parâmetro "AS".

```
SELECT MAX(valor) AS imovel_valor_maximo FROM IMOVEIS
```

imovel_valor_maximo
622000.00

Para o próximo exemplo utilizaremos a tabela TREINO como mostra abaixo:

idAtleta	data	tempo
42159	2014-11-20	01:20:13
53792	2014-11-20	01:31:13
84151	2014-11-20	01:17:42
42159	2014-11-21	01:22:24
53792	2014-11-21	01:36:46
84151	2014-11-21	01:19:08
42159	2014-11-22	01:18:08
53792	2014-11-22	01:32:29
84151	2014-11-22	01:23:54

Uma equipe de três atletas treina uma série de corridas. Queremos saber qual foi o pior treino para cada atleta, ou seja, qual corrida levou maior tempo.

já que procuramos o maior tempo por atleta, vamos agrupar os atletas que estão armazenados na coluna "idAtleta". Para agrupar, utilizaremos o " GROUP BY".

Para calcularmos o maior tempo por atleta, executamos a sentença abaixo:

```
SELECT idAtleta, MAX(tempo) AS pior_tempo
FROM TREINO
GROUP BY idAtleta
```

Após a execução, teremos o pior tempo para cada atleta. Conforme podemos visualizar na imagem abaixo:

idAtleta	pior_tempo
42159	01:22:24
53792	01:36:46
84151	01:23:54

Para ordenar a identificação dos atleta em ordem crescente, ou seja, do valor menor para o maior, deve-se usar "ORDER BY" seguido do nome da coluna, neste caso, a coluna 'idAtleta' será ordenada.

Para ordenar as identificações, utilizamos a sentença abaixo:

```
SELECT idAtleta,
MAX(tempo) AS pior_tempo
FROM TREINO
GROUP BY idAtleta
ORDER BY idAtleta
```

Após a execução, teremos as identificações dos atletas em ordem crescente. Conforme podemos visualizar na imagem abaixo:

idAtleta	pior_tempo
42159	01:22:24
53792	01:36:46
84151	01:23:54

11.5. MIN

A função MIN retorna o valor mínimo (menor valor) de uma coluna. Para exemplificar, vamos utilizar a tabela IMOVEIS utilizada anteriormente.

idCorretor	idImovel	valor
120	5421	180000.00
124	5421	90000.00
122	5421	130000.00
124	5421	512000.00
120	5421	70000.00
122	5421	622000.00
120	5421	127000.00

Os valores dos imóveis estão armazenados na coluna valor, logo devemos calcular o valor mais baixo a partir desta coluna.

Para calcularmos o valor mínimo, executamos a sentença abaixo:

```
SELECT MIN(valor)AS imovel_valor_minimo FROM IMOVEIS
```

Após a execução, teremos o imóvel com o valor mais baixo. Conforme podemos visualizar na imagem a seguir:

imovel_valor_minimo
70000.00

Assim como as funções anteriores, foi inserido um parâmetro AS para nomear a coluna.

Para o próximo exemplo utilizaremos a tabela TREINO. Queremos saber qual foi o melhor treino para cada atleta, ou seja qual corrida levou menos tempo.

Já que procuramos o menor tempo por atleta, vamos agrupar os atletas que estão armazenados na coluna "idAtleta". Para agrupar, utilizaremos o "GROUP BY".

Para calcularmos o menor tempo por atleta, executamos a sentença abaixo:

```
SELECT idAtleta, MIN(tempo) AS melhor_tempo
FROM TREINO
GROUP BY idAtleta
```

Após a execução, teremos o melhor tempo para cada atleta. Conforme podemos visualizar na imagem abaixo:

idAtleta	melhor_tempo
42159	01:18:08
53792	01:31:13
84151	01:17:42

Para ordenar a identificação dos atletas em ordem crescente, ou seja, do valor menor para o maior, deve-se usar "ORDER BY" seguido do nome da coluna, neste caso, a coluna 'idAtleta' será ordenada.

Para ordenar as identificações, utilizamos a sentença abaixo:

```
SELECT idAtleta,
MIN(tempo) AS pior_tempo
FROM TREINO
GROUP BY idAtleta
ORDER BY idAtleta
```

Após a execução, teremos as identificações dos atletas em ordem crescente. Conforme podemos visualizar na imagem abaixo:

idAtleta	melhor_tempo
42159	01:18:08
53792	01:31:13
84151	01:17:42