

DISCIPLINA
TÓPICOS DA ARQUITETURA
SQL SERVER



SUMÁRIO

APRESENTAÇÃO	3
UNIDADE 1.....	5
INTRODUÇÃO	6
1. MICROSOFT SQL SERVER MANAGEMENT STUDIO	7
2. PRIMEIROS SCRIPTS EM T-SQL.....	14
3. VARIÁVEIS NO T-SQL E TIPOS DE DADOS.....	17
CONSIDERAÇÕES FINAIS	25
REFERÊNCIAS.....	26
UNIDADE 2.....	27
INTRODUÇÃO	28
1. CONCATENAÇÃO DE STRINGS	29
2. OPERADORES ARITMÉTICOS	32
3. EXPRESSÕES LÓGICAS	34
4. CONTROLE DE FLUXO COM T-SQL.....	36
5. ADIÇÃO DE COMENTÁRIOS.....	41
CONSIDERAÇÕES FINAIS.....	44
REFERÊNCIAS.....	45
UNIDADE 3.....	46
INTRODUÇÃO	47
1. INTEGRIDADE REFERENCIAL E RELACIONAMENTOS	48
2. CHAVES ESTRANGEIRAS.....	51
3. CRIAÇÃO DE CAMPOS CALCULADOS	58
4. JOINS.....	59
5. CASE, GROUP BY E FUNÇÕES DE AGREGAÇÃO.....	65
CONSIDERAÇÕES FINAIS	70
REFERÊNCIAS.....	71
UNIDADE 4.....	72
INTRODUÇÃO	73
1. SYSTEM VIEWS	74
CONSIDERAÇÕES FINAIS	85
REFERÊNCIAS.....	86
CONCLUSÃO	87

APRESENTAÇÃO

Tópicos da Arquitetura SQL Server

Professor Me. Ricardo Bortolo Vieira

Olá prezados(as) alunos(as), seja bem vindo ao livro Tópicos da Arquitetura SQL Server. Sou o professor Ricardo Vieira, autor deste material. O assunto que abordarei no decorrer do nosso estudo poderá auxiliá-lo em sua carreira, ou abrir portas para o mundo dos negócios, mostrando-lhe como é bom trabalhar com banco de dados. Com o passar do tempo, as ferramentas para gestão de banco de dados foram tornam-se mais simples, rápidas e robustas.

Meu objetivo, por meio deste livro, é ensiná-lo como gerenciar um projeto de banco de dados no SQL Server, e apresentar algumas ferramentas que podem auxiliá-lo neste processo. Além disso, pretendo deixar claro que o uso correto dessa técnica poderá ajudá-lo a alcançar os objetivos estratégicos de sua empresa ou auxiliá-lo a colocar em prática uma nova ideia.

Então meu amigo(a), acompanhar e dominar as boas práticas de projetos de banco de dados não é mais um diferencial, mas sim uma questão de posicionamento de mercado.

Este livro está organizado em quatro unidades, além da introdução e conclusão, cada uma delas correspondendo a uma das partes importante na gestão de um banco de dados em SQL Server.

A primeira unidade trata sobre o ambiente de desenvolvimento do SQL SERVER e apresentará os primeiros passos na construção de scripts. Na segunda unidade é apresentado os tipos de operadores e fluxos de controle. Na terceira unidade é discutido Integridade referencial, Chaves e JOINS. Na quarta unidade abordaremos objetos específicos do SQL SERVER System Views e Sys.Objects.

ATENÇÃO

Todos devem conhecer ou ter ouvido falar sobre obra “Alice no país das maravilhas”, obra publicada em 1865 e adaptada pela primeira vez ao cinema pela Disney em 1951. Conta a história que Alice é uma menina que persegue um coelho branco de colete e relógio de bolso até uma toca. Ela cai e chega a um lugar muito esquisito, o País das Maravilhas. Lá ela encontra um gato que pode desaparecer, entre outros animais falantes. Ao encontrar este gato em uma encruzilhada ela lhe pergunta:

“__ Podes dizer-me, por favor, que caminho devo seguir para sair daqui?
__ Isso depende muito de para onde você quer ir. — respondeu o gato.
__ Preocupa-me pouco aonde ir — disse Alice.
__ Neste caso, pouco importa o caminho que sigas — replicou o gato.”

Lewis Carroll

Car@ Alun@, este livro é um dos caminhos, porém tenham em evidência os objetivos que desejam alcançar, caso contrário nem o melhor caminho será suficiente para chegar.

Este livro foi criado especialmente para você, caro(a) aluno(a). Se aprofunde nos conteúdos, reflita sobre os assuntos relacionados, estude as propostas de leituras complementares e não pare de estudar, este assunto é muito vasto e não termina neste material, aliás, seu estudo apenas começa aqui. Quanto mais estudamos, mais sentimos que ainda existe muito a se conhecer, e é necessária muita sabedoria para aceitar isso. Desejo uma excelente caminhada em busca do conhecimento e muita humildade nesse processo.

Agora, mãos à obra! Tenha uma boa e agradável leitura!

UNIDADE 1

Professor Mestre Ricardo Vieira

Plano de Estudo:

1. MICROSOFT SQL SERVER MANAGEMENT STUDIO

2. PRIMEIROS SCRIPTS EM T-SQL

3. VARIÁVEIS NO T-SQL E TIPOS DE DADOS

3.1 *GETDATE ()* e *CURRENT_TIMESTAMP*

3.2 *CAST* e *CONVERT*

6. REFERÊNCIAS

Objetivos de Aprendizagem:

- Conceituar e contextualizar a utilização do Microsoft SQL Server Management Studio.
- Compreender os tipos de variáveis utilizadas no T-SQL Server.
- Estabelecer a importância dos Tipos de dados da linguagem SQL.

INTRODUÇÃO

Caro (a) aluno (a), esta unidade inicia os estudos com uma breve introdução ao conceito e a utilização do *Microsoft SQL Server Management Studio* dentro da linguagem SQL.

Na primeira parte você poderá encontrar os assuntos de grande importância como as funcionalidades do *Microsoft SQL Server Management Studio* e suas características e funções.

Na segunda parte, serão tratados os assuntos relacionados a uma introdução aos primeiros scripts da linguagem T-SQL.

Você poderá ver as Variáveis no T-SQL e Tipos de Dados e muito mais. Então vamos em frente.

Bons estudos e tenha um excelente aproveitamento!

#REFLITA

"O futuro não é definido em uma trilha. É algo que podemos decidir, e na medida em que não violamos quaisquer leis conhecidas do universo, provavelmente podemos fazê-lo funcionar da maneira que queremos".

(Alan Kay)

Caro(a) aluno(a), deixo a reflexão para todos nós, de como pensar a programação, de criar e superar as expectativas, de maneira que sempre possa buscar o seu melhor. Seja o melhor que você puder ser. O aprimoramento deve ser constante.

Referência: <https://citacoes.in/citacoes/1841701-alan-kay-the-future-is-not-laid-out-on-a-track-it-is-somet/>

1. MICROSOFT SQL SERVER MANAGEMENT STUDIO



Após você fazer a instalação do *SQL Server* em seu computador, você deverá clicar no menu *iniciar\programas* e verificar que uma série de programas foram instalados juntamente com o produto. Um desses programas é o *SQL Server Management Studio (SSMS)*, que permite acessar uma instância localmente ou na rede.

Com o *Management Studio*, o profissional que estiver desenvolvendo ou gerenciando as instâncias instaladas, poderá criar banco de dados e tabelas, atualizar, inserir e excluir registros de acordo com a necessidade de negócio, alterar configurações, criar rotinas de *backup* e validar as consistências deste *backups*.

O *SQL Server Management Studio (SSMS)*, é a principal interface que os profissionais envolvidos com o *SQL Server* utilizam para interagir com a

tecnologia, criando tabelas, escrevendo consultas, alterando e excluindo registros. A seguir alguns painéis e funcionalidades desse programa.

1.1. Menu File ou Arquivo

O *Menu File* ou *Arquivo* permite ao usuário abrir e salvar arquivos de *script*, que tenham a terminação "SQL".

Para abrir arquivos de *script*, clique em *File\Open\File*.

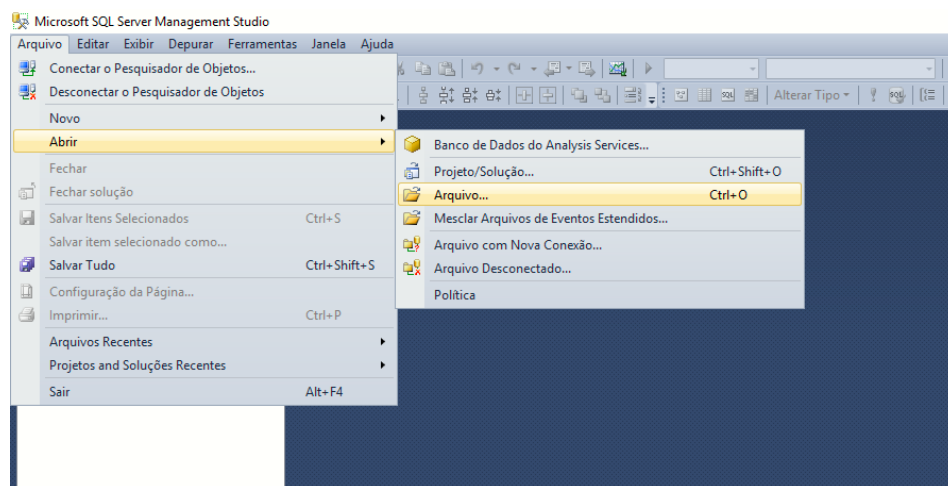


Figura 01. SSMS Menu Arquivo. Elaborado pelo autor.

1.2. Object ou Pesquisador Explorer de Objetos

O *Object Explorer* ou *Pesquisador de Objetos*, painel que por padrão fica localizado à esquerda da tela, mostra por meio de uma estrutura hierárquica as instâncias conectadas, seus Bancos de Dados e suas estruturas de objetos como tabelas, *procedures* e *functions*.

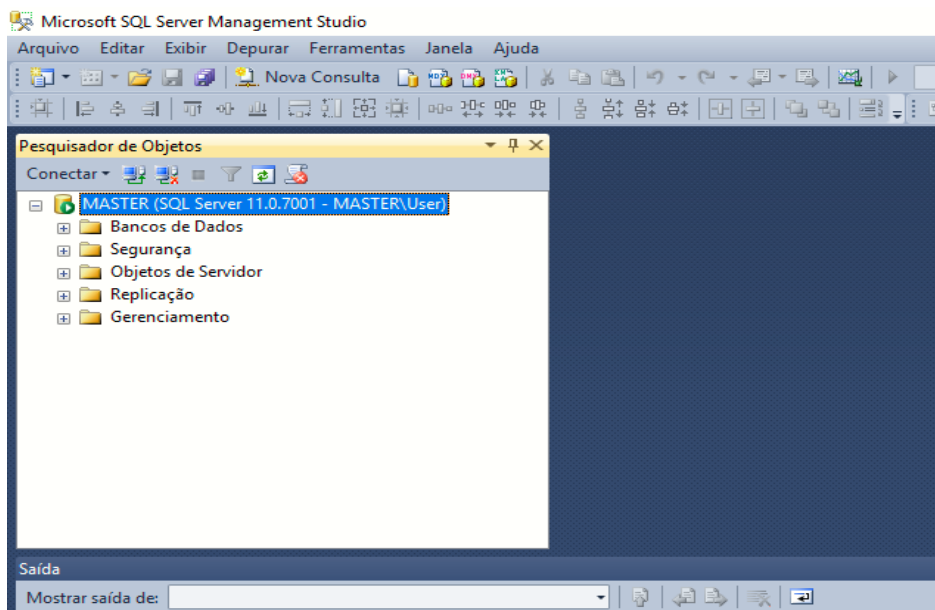


Figura 02. SSMS *Object Explorer*. Elaborado pelo autor.

Ou seja, é possível se conectar a diversas instâncias ao mesmo tempo e escrever *scripts*, acessar objetos e aplicar alterações de segurança em cada uma dessas instâncias, desde que você tenha as permissões necessárias para isso.

1.3. Editor de *Querys*

O editor de *querys* permite que o usuário escreva consultas e trechos de códigos *T-SQL* no *Management Studio*.

Na barra *default*, localizada na parte superior do *Management Studio*, com o botão *New Query* é possível abrir novas janelas de consultas e também usar a combinação de atalhos **CTRL+N**.

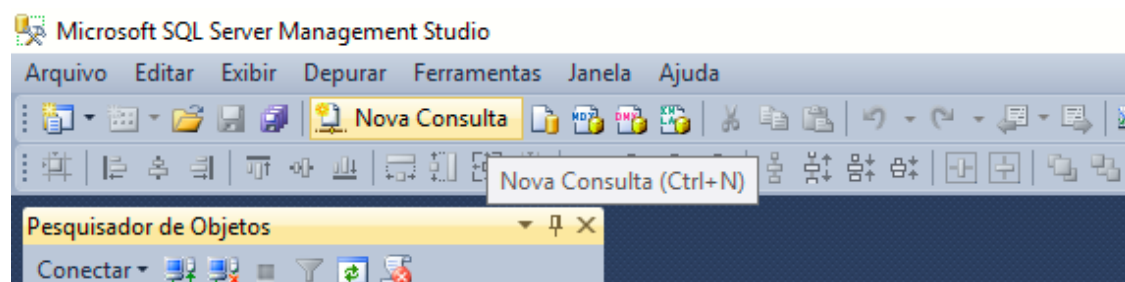


Figura 03. Botão *New Query* ou *Nova Consulta*. Elaborado pelo autor.

Observe o *Management Studio* com uma janela *query* já aberta:

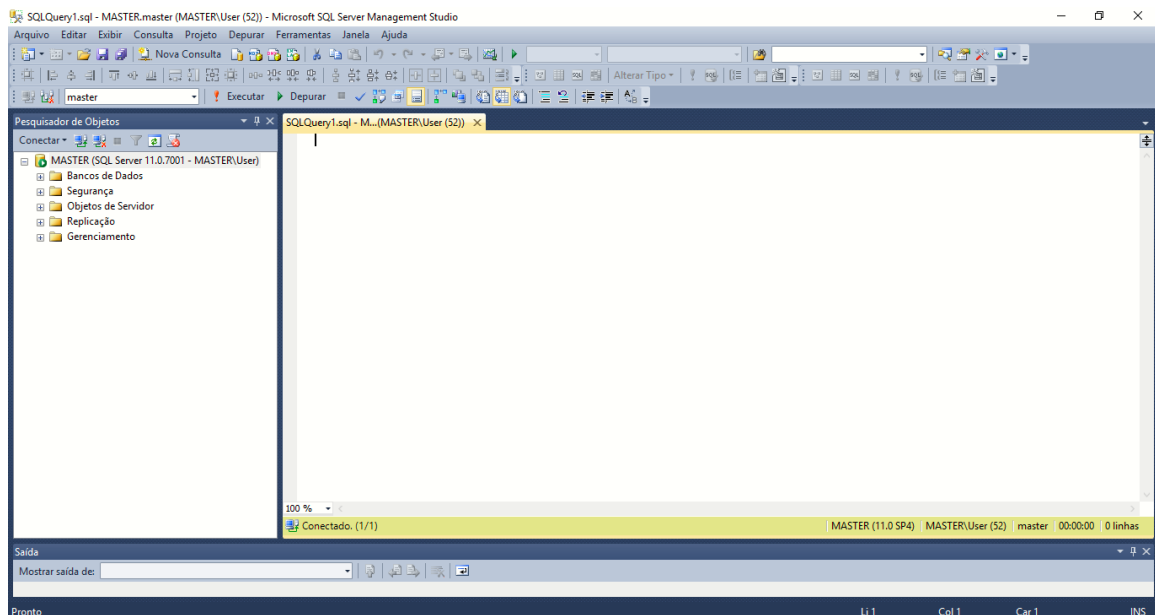


Figura 04. Janela de Consulta. Elaborado pelo autor.

O editor de *query* possibilita localizar, trechos de códigos pelo *menu Edit\Find\and replace* ou pelo atalho CTRL+F. É possível também salvar *queries* em arquivos com terminação *SQL* e abri-los posteriormente.

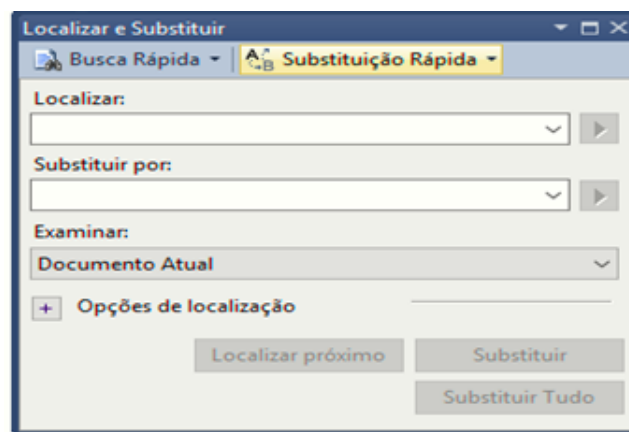


Figura 05. Janela Localizar e Substituir. Elaborado pelo autor.

1.4 Database Selection

É um *ListBox* localizado na barra de botões que listam todos os Bancos de Dados criados na instância, de sistema ou criados pelo usuário.

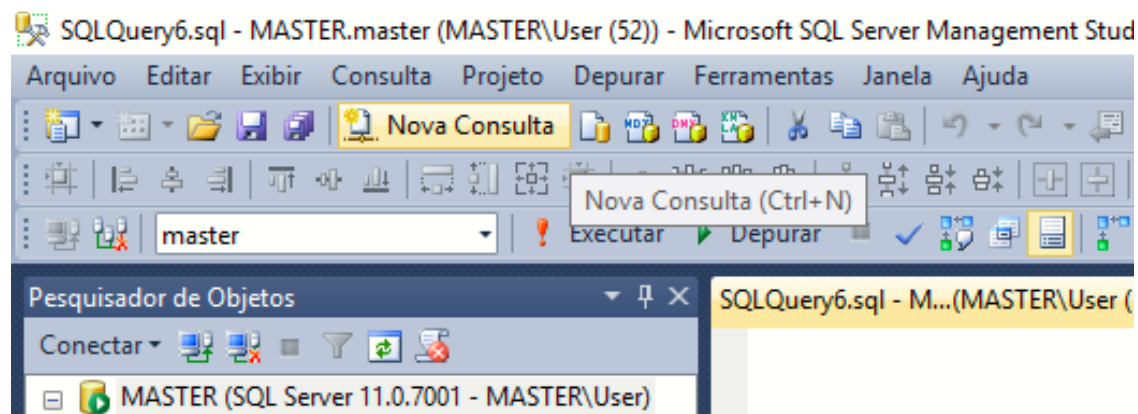


Figura 06. *Database Selection*. Elaborado pelo autor.

Preste sempre muita atenção nesse campo, pois todas as consultas e os scripts de atualização executados nas janelas de consultas são contra o Banco de Dados selecionado nesse campo.

1.5. Os Bancos de Dados de Sistema

Clicar no ícone de expansão do *SQL Server* que deve mostrar uma subpasta chamada *System Databases*. Expanda essa pasta também. Na sua tela deve aparecer a informação apresentada em seguida.

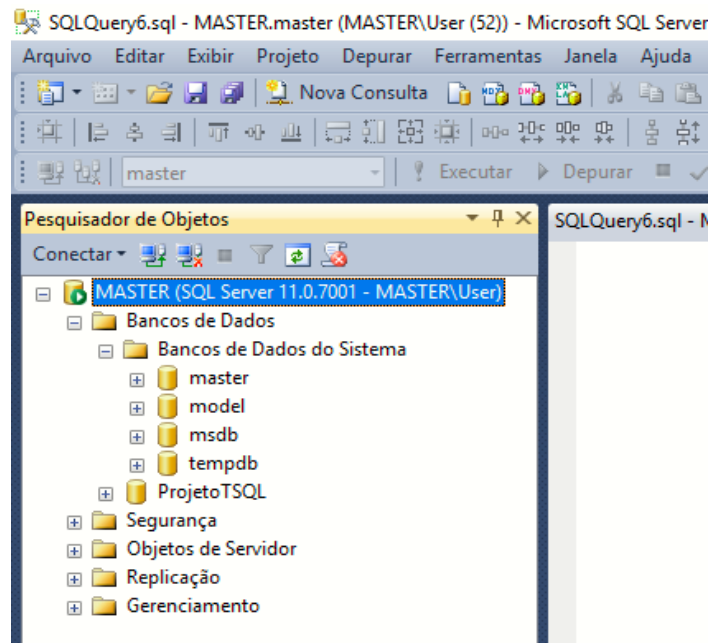


Figura 06. System Databases. Elaborado pelo autor.

Esses são os Bancos de Dados de sistema do *SQL Server*, criados durante o processo de instalação do produto.

Cada um deles tem uma função específica no funcionamento no *SGBD*.

1.5.1. *master*

O banco de dados *master* registra todas as informações de sistema sobre a instância do *SQL Server*. Abrange os *metadados* dos bancos de dados da instância, contas de *login*, informações de servidores instalados à instância, também chamados de *linked servers*, e configurações de sistemas.

1.5.2. *model*

O banco de dados *model* é utilizado como um *template* para a criação de outros bancos de dados, todos os bancos de dados são criados a partir desse modelo.

1.5.3. *msdb*

O banco de dados *msdb* é usado pelo *SQL Server Agent* para armazenar informações sobre o agendamento de *jobs* e configurações de algumas funcionalidades do *Manegement Studio*.

1.5.4. *tempdb*

O *tempdb* é o banco de dados que armazena todos os objetos que são criados temporariamente pelo *SQL Server*.

Trabalharemos com alguns desses objetos mais á frente, por exemplo, com as tabelas temporárias.

Outros objetos são de uso interno do *SQL Server*, como tabelas de trabalho, registro de cursores, etc.

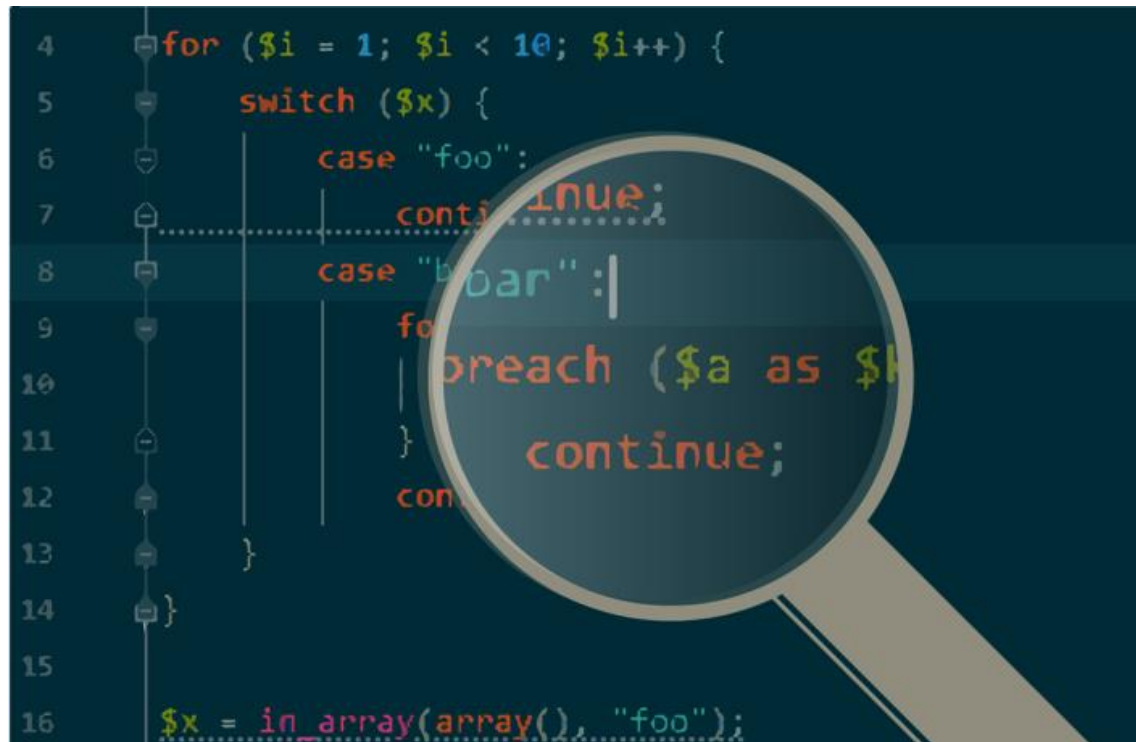
Toda vez que a instância é reiniciada, o *tempdb* é recriado e utiliza como *template* de recriação o banco de dados *model*.

#SAIBA MAIS#

Caro(a) aluno(a), dando continuidade no aprimoramento profissional, deixo a sugestão de leitura deste artigo "Trabalhando com a linguagem *T-SQL*", que tenho certeza que te ajudará a sanar muitas dúvidas e lhe passará uma visão mais ampla da linguagem *T-SQL*. Serão discutidos alguns recursos que podem ser usados no desenvolvimento e manipulação na consulta usando a ferramenta *SQL Server*.

<https://www.devmedia.com.br/trabalhando-com-a-linguagem-t-sql/38126>

2. PRIMEIROS SCRIPTS EM T-SQL



A linguagem *Transact-SQL*, ou simplesmente *T-SQL*, é uma extensão proprietária do SQL comum desenvolvida pela *Microsoft*.

Assim como o SQL Server padrão, é uma linguagem declarativa, mas com alguns elementos procedurais.

Abra o *SQL Server Management Studio*, ou simplesmente o *Management Studio* e conecte-se à instância instalada no seu computador. Na tela principal clique no botão "*New Query*". Uma janela *query* deve se abrir.

Digite o trecho de código seguinte:

```
SELECT 'Hello World! T-SQL na Prática!'
```

Agora clique no botão **Execute** ou aperte o *F5*. O seguinte resultado deve aparecer:

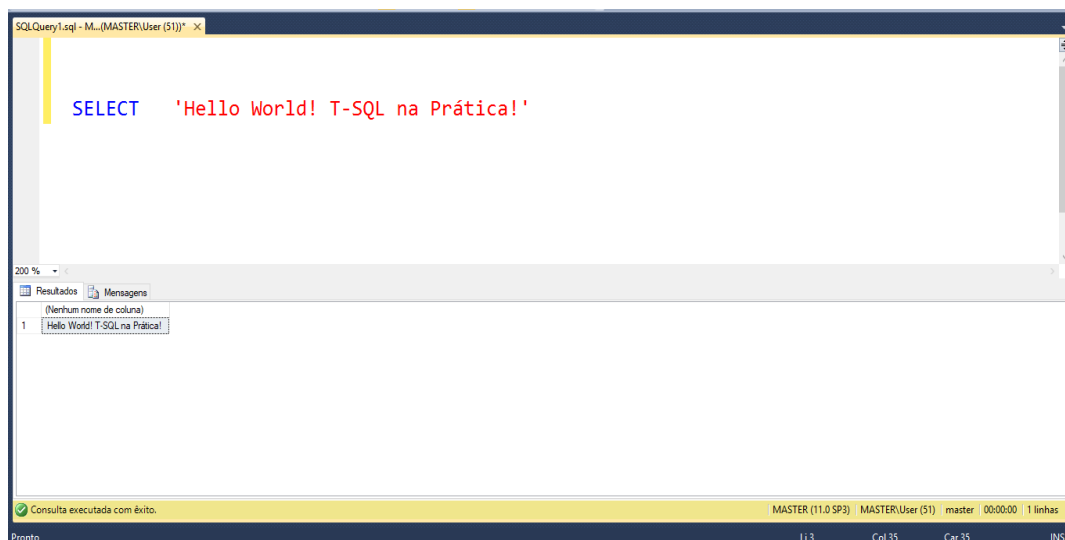


Figura 07. Resultado de uma consulta. Elaborado pelo autor.

Se o resultado foi igual a este, parabéns!

A *query* é bem simples e envolve apenas o comando *SELECT* e a String *'Hello World! T-SQL na prática!'*. No SQL Server, *strings* são indicados com apóstrofes.

2.1. Como Salvar o Script

Clique no botão do script localizado no painel superior.

A tela para a escolha do nome do arquivo deve se abrir em seguida. Informe o nome "*Script1*", selecione uma pasta da sua escolha para salvar o arquivo e clique *Ok*.

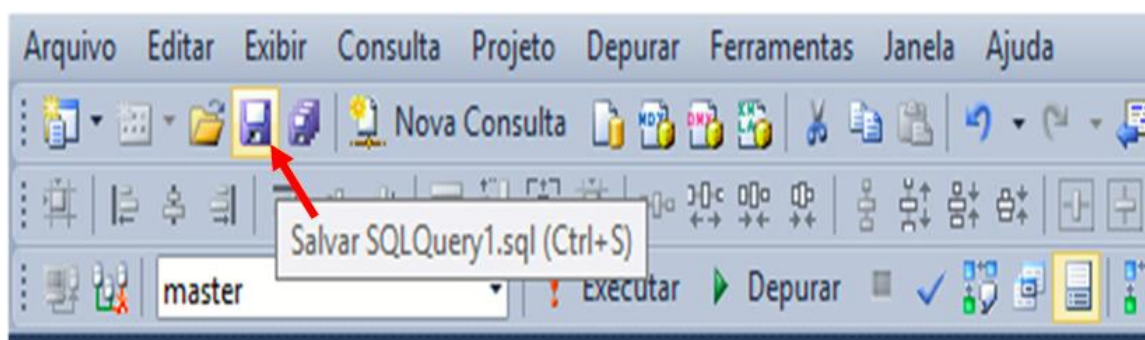


Figura 08. Exemplo de como salvar um arquivo. Elaborado pelo autor.

#FICA A DICA#



Título: Primeiros passos no SQL

• **Ano:** 2017.

• **Sinopse:** SQL é a linguagem padrão para acesso a dados em bancos relacionais. Por conta disso, é através dela que iniciamos nossos primeiros diálogos com os SGBDs, sistemas gerenciadores de bancos de dados. No DevCast abaixo, conversamos sobre a importância da Structured Query Language em diferentes momentos da carreira de um programador.

Referência: <https://www.devmedia.com.br/sql/>

3. VARIÁVEIS NO T-SQL E TIPOS DE DADOS



Com o *T-SQL* é possível criar variáveis que recebem e retornem valores, utilizando os tipos de dados disponíveis no *SQL Server*.

Os valores atribuídos às variáveis são armazenados apenas na memória. Eles não são armazenados fisicamente em nenhum de dados e assim que a sessão é encerrada, os dados atribuídos a essas variáveis são pedidos.

Clique em *new query* novamente e escreva seguinte código na janela de consulta:

```
DECLARE @HELLO VARCHAR(100)
```

```
SET @HELLO = 'HELLO WORLD! T-SQL na prática'
```

```
SELECT @HELLO
```

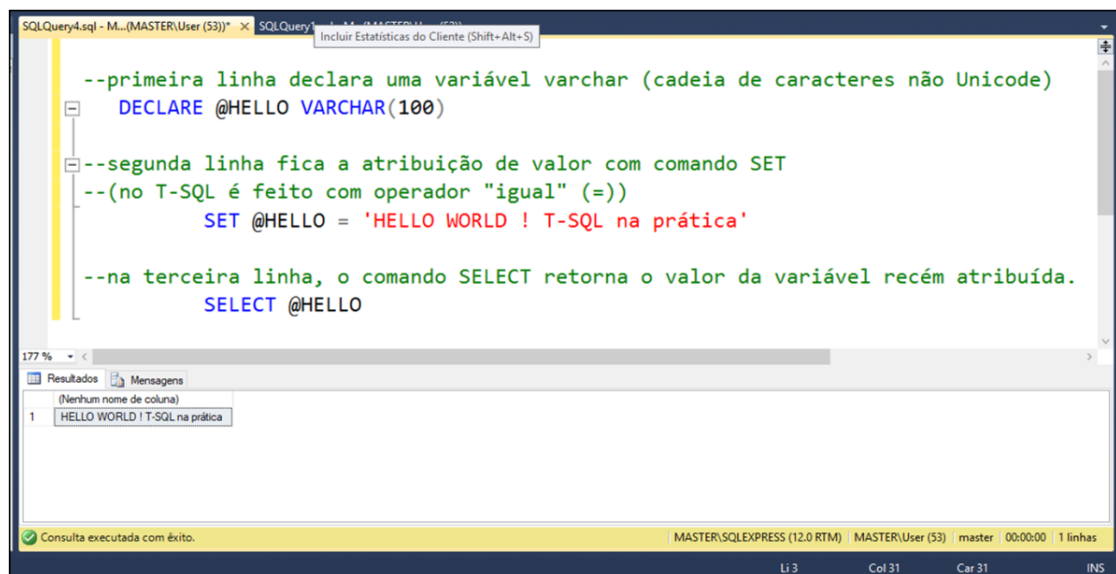


Figura 09. Exemplo de atribuição de valores a variáveis. Elaborado pelo autor.

Verifique no canto inferior direito do *Management Studio*, na barra de status do *Query Editor*, a mensagem "1 rows", indicando o número de linhas retornado pela *Query*.

Toda declaração de variável no *T-SQL* deve ser "*tipada*", ou seja, deve utilizar um dos tipos de dados disponíveis no *SQL Server*, que são:

Tabela 1. Tipos de dados Numéricos Exatos. Elaborado pelo autor.

Tipos Numéricos Exatos	
<i>bigint</i>	<i>numeric</i>
<i>bit</i>	<i>smallint</i>
<i>decimal</i>	<i>smallmoney</i>
<i>int</i>	<i>tinyint</i>
<i>money</i>	

Tabela 2. Tipos de dados Numéricos Aproximados. Elaborado pelo autor.

Tipos Numéricos Aproximados	
<i>float</i>	<i>real</i>

Tabela 3. Tipos de dados de Data e hora. Elaborado pelo autor.

Data e hora	
<i>date</i>	<i>datetimeoffset</i>
<i>datetime2</i>	<i>smalldatetime</i>
<i>datetime</i>	<i>time</i>

Tabela 4. Tipos de dados Cadeia de Caracteres. Elaborado pelo autor.

Cadeia de Caracteres	
<i>char</i>	<i>varchar</i>
<i>text</i>	

Tabela 5. Tipos de dados Cadeia de Caracteres Unicode. Elaborado pelo autor.

Cadeia de Caracteres Unicode	
<i>nchar</i>	<i>nvarchar</i>
<i>text</i>	

Tabela 6. Tipos de dados Binários. Elaborado pelo autor.

Binários	
<i>binary</i>	<i>varbinary</i>
<i>image</i>	

Tabela 7. Outros tipos de dados. Elaborado pelo autor.

Outros Tipos de Dados	
<i>cursor</i>	<i>timestamp</i>
<i>hierarchyid</i>	<i>uniqueidentifier</i>
<i>SQL_variant</i>	<i>xml</i>
<i>table</i>	

3.1 *GETDATE()* e *CURRENT_TIMESTAMP*

A função *GETDATE()* e a função *CURRENT_TIMESTAMP* retorna a data atual do Sistema Operacional em que se está sendo executada. As funções não esperam nenhum argumento, e retorna como resultado um valor do tipo *DATETIME*.

A função pode ser executada com um simples *SELECT*:

SELECT

GETDATE() AS [DATA COM GETDATE]

,*CURRENT_TIMESTAMP* AS [DATA COM CURRENT_TIMESTAMP]

Você deve estar se perguntando "mas, qual é a diferença entre elas, se as duas funções retornam o mesmo valor e se têm a mesma metodologia de uso"?

A diferença entre *GETDATE()* e *CURRENT_TIMESTAMP* é:

GETDATE() é uma extensão do *T-SQL*.

CURRENT_TIMESTAMP é padrão ANSI (American National Standards Institute).

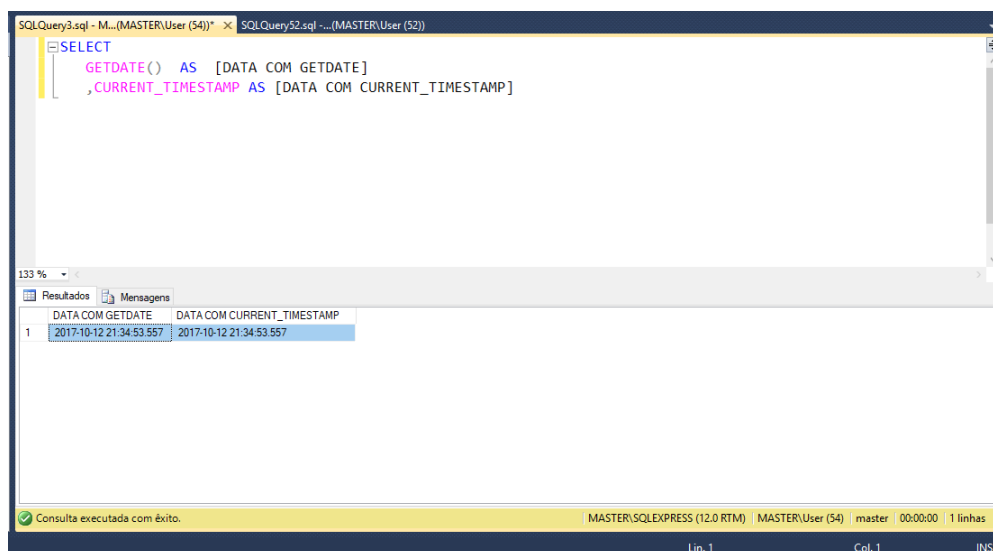


Figura 10. Exemplo de consulta em *GETDATE()* e *CURRENT_TIMESTAMP*.

Elaborado pelo autor.

3.2 CAST e CONVERT

As funções de conversão de tipos de dado do *T-SQL* são o *CAST* e o *CONVERT*.

O *CAST* é uma função padrão do *ANSI* e o *CONVERT* é uma extensão do *SQL Server*.

DECLARE

 @VALOR INT

SET @VALOR = 1000

PRINT 'VALOR É ' + @VALOR

Se você executou o código acima entrou a seguinte mensagem de erro:

Mensagem 245, Nível 16, Estado 1, Linha 6

Falha ao converter o varchar valor 'VALOR É ' para o tipo de dados int.

O erro ocorreu porque, ao tentar concatenar a *string* 'VALOR É' com a variável do tipo inteira @VALOR, houve um erro de conversão. Converta o valor no momento da concatenação. Altere da seguinte forma:

```
DECLARE
```

```
    @VALOR INT
```

```
SET    @VALOR = 1000
```

```
PRINT 'VALOR É ' + CAST(@VALOR AS VARCHAR)
```

O comando *CAST* está sendo usado para converter a variável @VALOR em *Varchar*, o que faz com o *script* "rode" sem erros. Este mesmo código pode ser escrito com a função *CONVERT*, que tem uma sintaxe diferente:

```
DECLARE
```

```
    @VALOR INT
```

```
SET    @VALOR = 1000
```

```
PRINT 'VALOR É ' + CONVERT(VARCHAR,@VALOR)
```

Com o *CONVERT* é necessário informar primeiramente o tipo de dados e depois o valor a ser convertido.

O *CONVERT* permite um terceiro argumento opcional, que é muito útil para conversão de datas.

É possível que datas se encontrem em formatos diversos e a conversão em *DATETIME* diretamente pode não ser possível. A função *CONVERT* é muito para estas situações:

SELECT

```
CONVERT(DATETIME, '10/25/2025', 101) AS [mm/dd/yyyy]
,CONVERT(DATETIME, '2025.10.25', 102) AS [yyyy.mm.dd]
,CONVERT(DATETIME, '25/10/2025', 103) AS [dd/mm/yyyy]
,CONVERT(DATETIME, '25.10.2025', 104) AS [dd.mm.yyyy]
,CONVERT(DATETIME, '25-10-2025', 105) AS [dd-mm-yyyy]
```

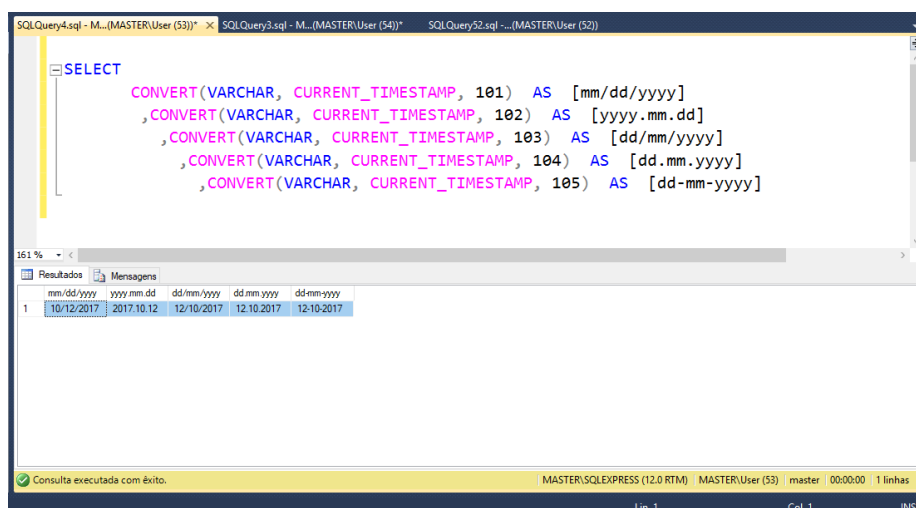


Figura 11. Exemplo de consulta em *CAST* e *CONVERT*. Elaborado pelo autor.

O terceiro argumento da função *CONVERT*, denominado *STYLE*, serve para que a função assimile o formato de entrada do segundo argumento, possibilitando que função execute a função.

É possível executar também o processo inverso, ou seja, converter valores do tipo *DATETIME* em uma cadeia de caracteres com formato desejado. Observe este exemplo em que se usa a função

CURRENT_TIMESTAMP retorna a data e a hora do servidor em que a instância está instalada:

SELECT

```
CONVERT(VARCHAR, CURRENT_TIMESTAMP, 101) AS [mm/dd/yyyy]
,CONVERT(VARCHAR, CURRENT_TIMESTAMP, 102) AS [yyyy.mm.dd]
,CONVERT(VARCHAR, CURRENT_TIMESTAMP, 103) AS [dd/mm/yyyy]
,CONVERT(VARCHAR, CURRENT_TIMESTAMP, 104) AS [dd.mm.yyyy]
,CONVERT(VARCHAR, CURRENT_TIMESTAMP, 105) AS [dd-mm-yyyy]
```

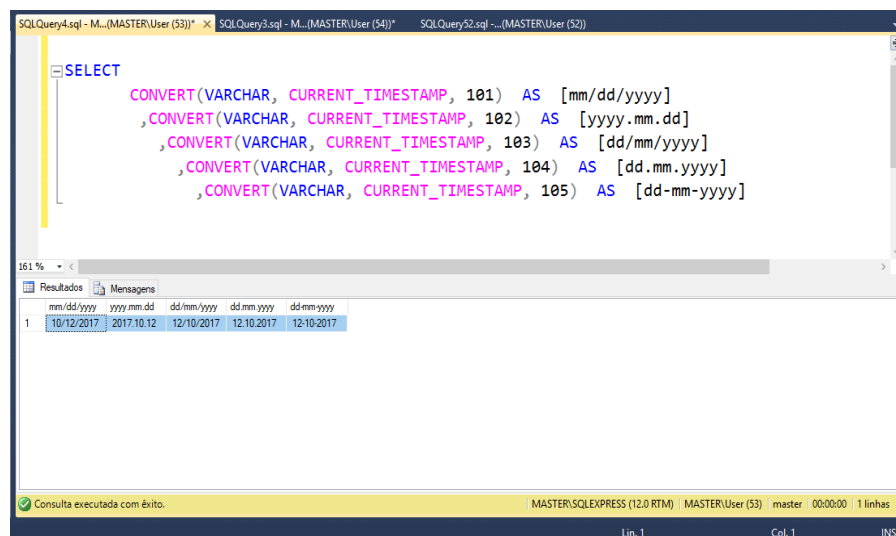
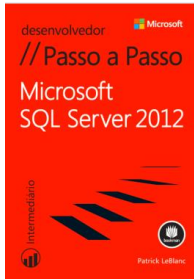


Figura 12. Exemplo de conversão de dados com *CONVERT*. Elaborado pelo autor.



#FICA A DICA#

- **Título:** Microsoft SQL Server 2012 - Passo a Passo
- **Autor:** Patrick LeBlanc
- **Editora:** Bookmam Editora Ltda.
- **Sinopse:** Aprenda a usar o *SQL Server 2012* – um passo de cada vez. Ideal para administradores e desenvolvedores iniciantes, este tutorial fornece exercícios práticos que vão ajudá-lo a projetar e gerenciar bancos de dados, desenvolver relatórios, utilizar sistemas de *business intelligence* e muito mais. Descubra como: Instalar, configurar e atualizar o *SQL Server*; Trabalhar com componentes e ferramentas essenciais; Economizar espaço em disco e obter um melhor desempenho com compactação; Utilizar *T-SQL* para alterar seus dados; Simplificar a escrita de consultas (*queries*) e melhorar o acesso a dados com código reutilizável; Gerenciar a carga de trabalho e o consumo de recursos; Gerenciar, fazer *backup*, recuperar e manter a segurança de bancos de dados; Construir soluções de banco de dados eficientes para a sua empresa.

CONSIDERAÇÕES FINAIS

Caro (a) aluno (a), chegamos ao final desta unidade que foi preparada com muito carinho para você. Nesta unidade foram apresentados alguns conceitos da linguagem T-SQL como as variáveis e os tipos de dados, as funções *GETDATE()* e *CURRENT_TIMESTAMP* retorna a data atual do Sistema Operacional.

O conteúdo que foi apresentado nesta unidade, foi somente a ponta do Iceberg, foi superficial, mas nas próximas unidades você poderá se aprofundar nos conceitos e prática da linguagem SQL.

Estaremos juntos na próxima unidade.

Até lá!

REFERÊNCIAS

- *LeBlanc, Patrick. **Microsoft SQL Server 2012 - Passo a Passo**. Porto Alegre. Bookmam Editora Ltda. 2014.*
- *Gonçalves, Rodrigo Ribeiro. **T-SQL com Microsoft SQL Server 2012 Express - na Prática**. Editora Érica Ltda. 2013.*
- *Gonçalves, Eduardo. **SQL - Uma abordagem para banco de dados Oracle**. Editora Casa do Código Ltda.*
- *Mistry, Ross; Misner, Stacia. **Introducing Microsoft SQL Server 2012**. Microsoft Press. 2012.*

UNIDADE 2

Professor Mestre Ricardo Vieira

Plano de Estudo:

- 1. CONCATENAÇÃO DE STRINGS**
- 2. OPERADORES ARITMÉTICOS**
- 3. OPERADORES LÓGICOS**
- 4. CONTROLE DE FLUXO COM T-SQL**
- 5. ADIÇÃO DE COMENTÁRIOS**
- 6. CONSIDERAÇÕES FINAIS**
- 7. REFERÊNCIAS**

Objetivos de Aprendizagem:

- Conceituar e contextualizar a utilização da concatenação de strings.
- Compreender os tipos de operadores Lógicos e aritméticos.
- Estabelecer a importância dos controladores de fluxos com o *T-SQL*.
- Conceituar e contextualizar a utilização da Adição de Comentários no *SQL*.
- Conceituar e contextualizar a realização de Operações aritméticas básicas.

INTRODUÇÃO

Caro (a) aluno (a), esta unidade inicia os estudos com uma breve introdução ao conceito de concatenação de Strings, dentro da linguagem SQL e suas características. Na segunda parte, serão tratados os assuntos relacionados a uma introdução aos Operadores Aritméticos.

Você poderá ver os tipos de Operadores Lógicos e os Controladores de Fluxo e muito mais.

Então vamos em frente. Bons estudos e tenha um excelente aproveitamento!

#REFLITA

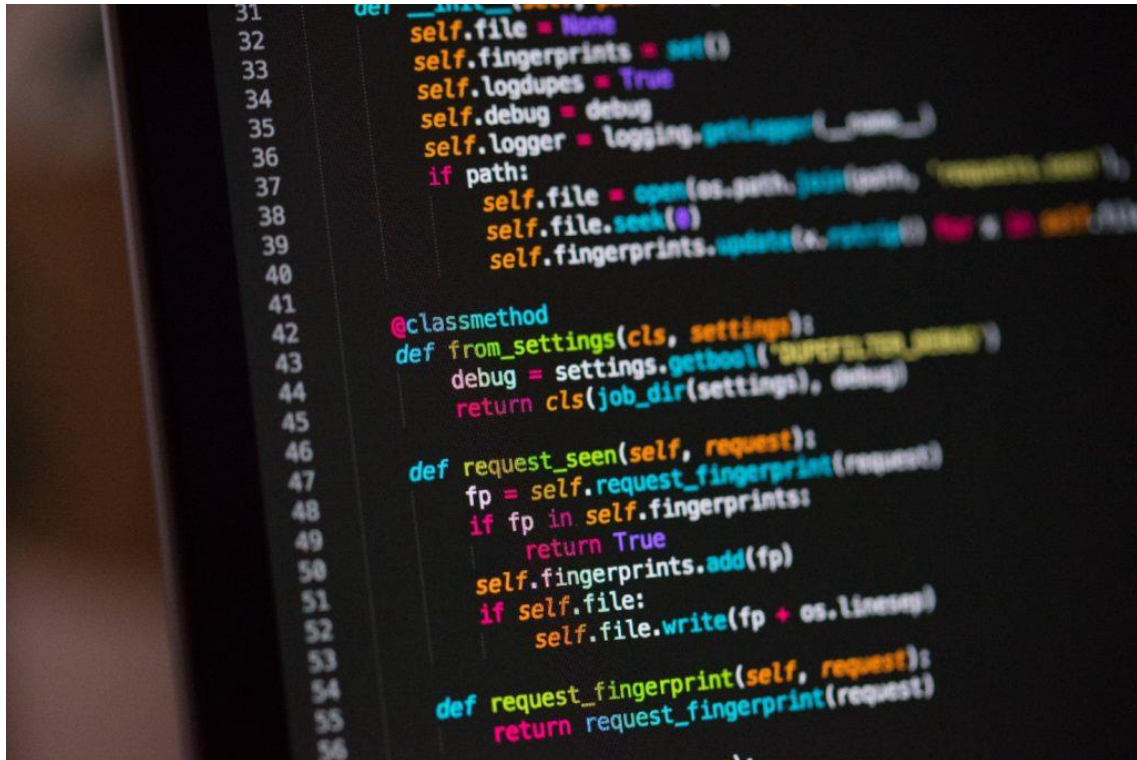
"Algumas pessoas acham que foco significa dizer sim para a coisa em que você vai se focar. Mas não é nada disso. Significa dizer não às centenas de outras boas ideias que existem. Você precisa selecionar cuidadosamente."
(Steve Jobs).

Caro (a) aluno (a), inspirado em uma das maiores mentes da computação, *Steve Jobs*, hoje é o dia certo para estudar, trabalhar, buscar algo novo. Mas, não devemos fazer isso tentando abraçar o mundo, ou seja, tentando fazer tudo ao mesmo tempo, mas sim, um de cada vez, porém de forma completa e totalmente envolvida. Assim, priorizar é a competência que v. deve aprimorar, somente assim será capaz de ir além e fazer coisas que outros não podem.

Hoje é o dia! Faça o hoje valer a pena, fazendo boas escolhas!

Referência: <https://www.tecmundo.com.br/internet/3145-frases-impactantes-sobre-tecnologia-e-informatica.htm>

1. CONCATENAÇÃO DE STRINGS



Concatenação é o processo de acrescentar uma cadeia de caracteres ao final de outra cadeia de caracteres. Você concatena cadeias de caracteres usando o operador `+`. Para literais de cadeia de caracteres e constantes de cadeia de caracteres, a concatenação ocorre em tempo de compilação; não ocorre nenhuma concatenação de tempo de execução. Para variáveis de cadeia de caracteres, a concatenação ocorre somente em tempo de execução.

Uma conversão explícita para dados de caracteres deve ser usada ao concatenar cadeias binárias e quaisquer caracteres entre as cadeias binárias.

Podemos concatenar *strings* de todos os tipos, como variáveis declaradas com tipos de caracteres variáveis e *strings* definidos em tempo de execução.

Para concatenar tipos distintos de dados é necessário usar a conversão de tipo `CAST` ou `CONVERT`. Menos tipos `image`, `ntext` ou `text`.

Veja o exemplo:

DECLARE

```

        @HELLO VARCHAR(100)
SET      @HELLO = 'Hello World! T-SQL na prática'
SELECT  @HELLO + ' - Concatenando Strings'

```

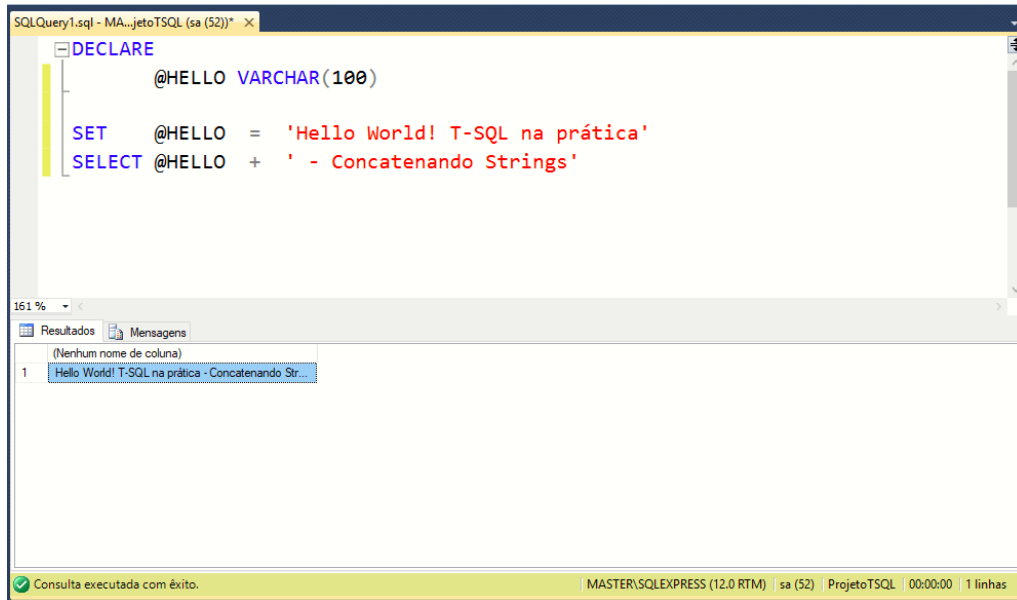


Figura 13. Exemplo de concatenação de *Strings* com *SET*. Elaborada pelo autor.

Vale observar que a atribuição de valor de variáveis pode ser feita com *SET* ou *SELECT*, levando ao mesmo resultado. A atribuição com *SET* está dentro dos padrões *ANSI*, mas com *SELECT* permite atribuição múltipla. Veja o exemplo:

```

DECLARE
    @HELLO  VARCHAR(100)
    ,@HELLO2 VARCHAR(100)

SELECT
    @HELLO = 'HELLO WORLD! T-SQL NA PRATICA - PRIMEIRA VARIABEL'
    ,@HELLO2 = 'HELLO WORLD! T-SQL NA PRATICA - SEGUNDA VARIABEL'

SELECT
    @HELLO
    ,@HELLO2

```

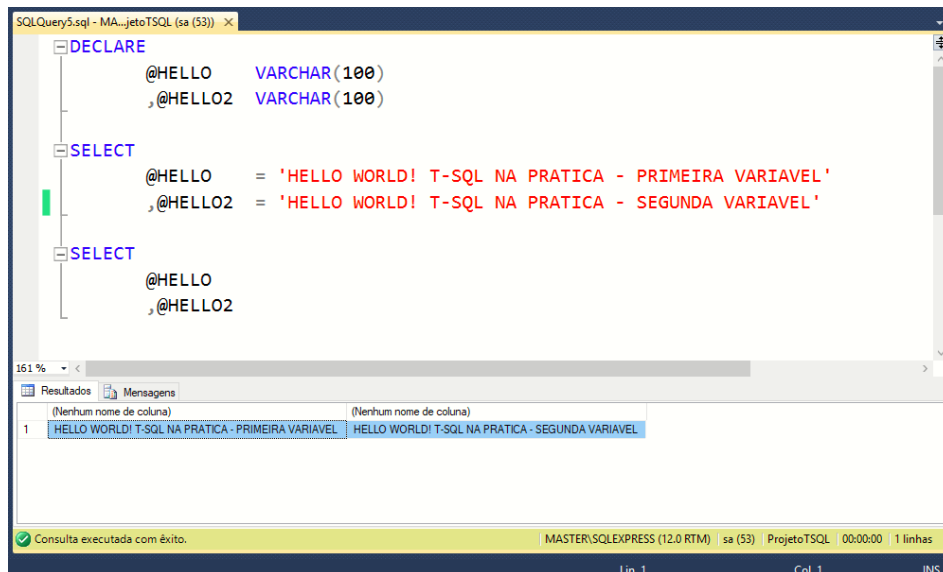



Figura 14. Exemplo de concatenação de *Strings* com *SELECT*. Elaborada pelo autor.

#SAIBA MAIS#

Caro(a) aluno(a), dando continuidade no aprimoramento profissional, deixo a sugestão de leitura deste artigo que tenho certeza que te ajudará a sanar dúvidas te dará uma visão mais ampla da Linguagem *Transact-Sql*.

Esta série de artigos apresentará uma introdução à linguagem *T-SQL*, a linguagem para desenvolvimento em banco de dados criados no *Microsoft SQL Server*. Através de exemplos práticos veremos como podemos utilizar o *SQL Server Management Studio* para aprender os fundamentos e conceitos de desenvolvimento em banco de dados.

Referencia: <https://www.devmedia.com.br/transact-sql/17682>

2. OPERADORES ARITMÉTICOS

Você pode usar operadores aritméticos em *MultiDimensional Expressions* (MDX) para cálculos aritméticos, incluindo adição, subtração, multiplicação e divisão.

O MDX suporta os operadores aritméticos listados na tabela 8.

Tabela 8. Tabela de Operadores Aritméticos. Elaborada pelo autor.

Operação	Operador
Adição	+
Subtração	-
Multiplicação	*
Divisão	/
Mod	%

- O resto da divisão é o que chamamos de "Mod".
- A potenciação (a^n), que é a função matemática que indica a multiplicação da base **a** por ela mesma tantas vezes quanto indicar o expoente **n**, pode ser utilizada pelo programador através de uma função matemática do Transact-SQL chamada `power(x,y)`.

Clique em *New Query* para abrir uma nova janela do *query* digite os seguintes *scripts*:

DECLARE

@VALOR1 INT
, @VALOR2 INT

SELECT

@VALOR1 = 12
, @VALOR2 = 6

SELECT

@VALOR1 + @VALOR2 AS [Resultado da Adição]

Três detalhes importantes sobre o *script* anterior:

1. Criaram-se duas variáveis usando o tipo *int* para a realização dessa operação aritmética.
2. A atribuição de valores à variável foi realizada com *SELECT*.
3. Utilizou-se a palavra-chave "AS" para definir um nome para a coluna de resultado desta operação.

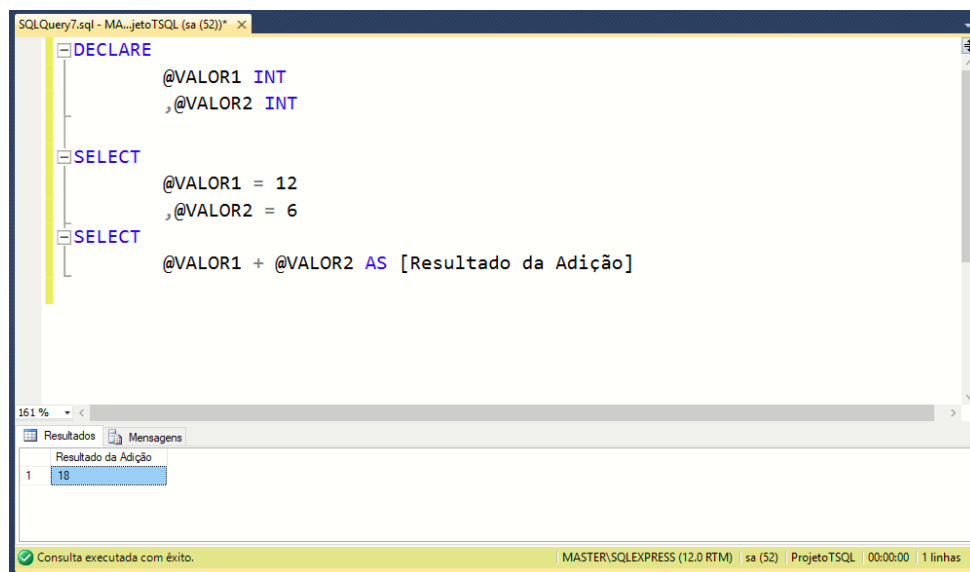


Figura 15. Exemplo de operação de adição. Elaborada pelo autor.

2.1. Ordem de Precedência

As regras a seguir determinam a ordem de precedência para operadores aritméticos em uma expressão *MDX*:

- Expressões entre parênteses têm precedência sobre todas as outras operações;
- Quando todos os operadores aritméticos em uma expressão têm o mesmo nível de precedência, a ordem de execução é da esquerda para a direita;
- Quando há mais de um operador aritmético em uma expressão, o *MDX* executa primeiro, multiplicação e divisão, seguido de subtração e adição.

3. EXPRESSÕES LÓGICAS

As expressões lógicas são operadores e comparadores que são utilizados em store procedures, consultas em SQL e T-SQL para realizar comparações e promover desvios condicionais dentro de uma lógica de programação. Na seção sobre controle de fluxo estas expressões serão apresentadas dentro de um contexto de programação.

3.1. OPERADORES LÓGICOS

Os operadores lógicos testam a legitimidade de algumas condições. Os operadores lógicos retornam um tipo de dados Boolean com um valor TRUE, FALSE ou UNKNOWN.

Tabela 9. Tabela de Operadores Lógicos. Elaborada pelo autor.

Operador	Significado	Exemplo
AND	e	expressão AND expressão
OR	ou	expressão OR expressão
NOT	não (negação)	NOT (expressão)
ALL	TRUE se tudo em um conjunto de comparações for TRUE	ALL(2, 3, 4) = resultado
ANY	TRUE se qualquer conjunto de comparações for TRUE	ANY(2, 3, 4) = resultado
EXISTS	TRUE se uma subconsulta tiver qualquer linha	EXISTS (SELECT 1 FROM CLIENTE)
BETWEEN	TRUE se o operando estiver dentro de um intervalo	WHERE BETWEEN 90 AND 200
IN	TRUE se o operando for igual a um de uma lista de expressões	@AUX IN (1, 2, 3)
LIKE	TRUE se o operando corresponder a um padrão	WHERE coluna LIKE '%RUA%'
SOME	TRUE se algum conjunto de comparações for TRUE	SOME(2, 3, 4) = resultado

3.2. COMPARADORES LÓGICOS

Tabela 10. Tabela de Comparadores Lógicos. Elaborada pelo autor.

Operador	Definição
=	Igual a
>	Maior que
<	Menor que
>=	Maior ou igual que
<=	Menor ou igual que
<>	Diferente que

= (Igual a)

Para argumentos não nulos, retorna *TRUE* se o argumento da esquerda for igual ao argumento da direita; caso contrário, *FALSE*.

Se um ou ambos os argumentos avaliarem para um valor nulo, o operador retornará um valor nulo, a menos que a comparação *0 = null* seja feita, caso em que o booleano contém *TRUE*.

> (Maior que)

Para argumentos não nulos, retorna *TRUE* se o argumento esquerdo tiver um valor maior que o argumento correto; caso contrário, *FALSE*.

Se um ou ambos os argumentos avaliarem para um valor nulo, o operador retornará um valor nulo.

< (Menor que)

Para argumentos não nulos, retorna *TRUE* se o argumento esquerdo tiver um valor menor que o argumento correto; caso contrário, *FALSE*.

Se um ou ambos os argumentos avaliarem para um valor nulo, o operador retornará um valor nulo.

>= (Maior ou igual a)

Para argumentos não nulos, retorna *TRUE* se o argumento da esquerda tiver um valor maior ou igual ao argumento da direita; caso contrário, *FALSE*.

Se um ou ambos os argumentos avaliarem para um valor nulo, o operador retornará um valor nulo.

<= (Menor ou igual a)

Para argumentos não nulos, retorna *TRUE* se o argumento esquerdo tiver um valor menor ou igual ao argumento direito; caso contrário, *FALSE*.

Se um ou ambos os argumentos avaliarem para um valor nulo, o operador retornará um valor nulo.

<> (Diferente que)

Para argumentos não nulos, retorna *TRUE* se o argumento da esquerda não for igual ao argumento da direita; caso contrário, *FALSE*.

Se um ou ambos os argumentos avaliarem para um valor nulo, o operador retornará um valor nulo.

Alguns operadores que o *T-SQL* apresenta são extensões, ou seja, não são padrões *SQL ISO*. Por exemplo, o operador (**<>**) (**Diferente que**) pode ser substituído por (**!=**). O operador (**!<**) pode ser usado para verificar se o número é "**não menor**" que outro, e (**!>**) verifica se o número é "**não maior**" que outro.

4. CONTROLE DE FLUXO COM T-SQL

Assim como muitas linguagens de programação utilizam operadores de condição, o *SQL* não poderia ficar de fora. Ele trabalha com esses elementos, também denominado de controle de fluxo, permitindo assim ao desenvolvedor criar lógicas para as mais variadas situações e regras de negócio de seu sistema.

#FICA A DICA



- **Título:** Aprendendo SQL

- **Autor:** Alan Beaulieu

- **Editora:** Novatec Editora Ltda.

- **Sinopse:** Cada capítulo apresenta uma lição independente sobre um conceito chave de SQL, com várias ilustrações e exemplos comentados. Exercícios no final de cada capítulo permitem praticar as habilidades aprendidas. Com este livro você irá: Passar rapidamente pelo básico de SQL e aprender inúmeras funcionalidades avançadas. Usar instruções de dados SQL para gerar, manipular e recuperar dados. Criar objetos de banco de dados, como tabelas, índices e restrições, usando instruções de esquema SQL. Aprender como os conjuntos de dados interagem com as consultas e entender a importância das subconsultas. Converter e manipular dados usando funções nativas SQL e usar lógica condicional em instruções de dados.

4.1. Utilização do *IF/ELSE*

Impõe condições na execução de uma instrução *Transact-SQL*.

- A instrução *Transact-SQL* que segue uma palavra-chave *IF* e sua condição é executada se a condição for satisfeita:
 - A expressão booleana retorna *TRUE*.
- A palavra-chave *ELSE* opcional introduz outra instrução *Transact-SQL* que é executada quando a condição *IF* não é satisfeita:
 - A expressão booleana retorna *FALSE*.

Uma construção *IF/ELSE* pode ser usada em lotes, em procedimentos armazenados e em consultas *ad hoc*. Quando essa construção é usada em um procedimento armazenado, ela é frequentemente usada para testar a existência de algum parâmetro.

Os testes *IF* podem ser aninhados após outro *IF* ou após um *ELSE*. O limite para o número de níveis aninhados depende da memória disponível.

É possível criar uma divisão de fluxo com base em uma condição com *IF*, da seguinte forma:

```
DECLARE
    @VALOR1 INT
    , @VALOR2 INT
SELECT
    @VALOR1 = 1
    , @VALOR2 = 3

IF (@VALOR1 < @VALOR2)
    PRINT 'VALOR1 É MENOR QUE VALOR2'
ELSE
    PRINT 'VALOR2 É MENOR QUE VALOR1'
```

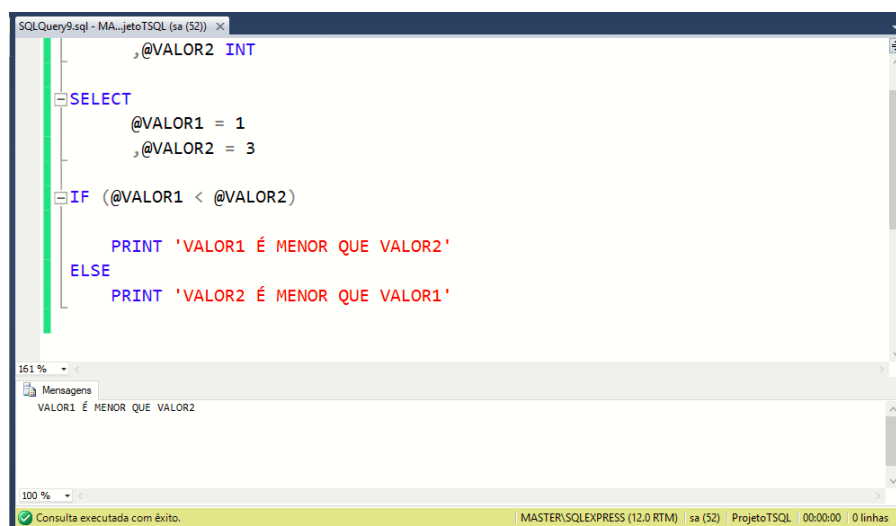


Figura 16. Exemplo de utilização do *IF/ELSE*. Elaborada pelo autor.

4.2. Uso do *WHILE*

Define uma condição para a execução repetida de uma instrução *SQL* ou bloco de instruções. As instruções são executadas repetidamente desde que a condição especificada seja verdadeira. A execução de instruções no *loop WHILE* pode ser controlada de dentro do loop com as palavras-chave *BREAK* e *CONTINUE*.

Se dois ou mais loops *WHILE* estiverem aninhados, o *BREAK* interno sai para o próximo loop mais externo. Todas as instruções após o final do loop interno são executados primeiro e, em seguida, o próximo loop externo é reiniciado.

De uma maneira simplificada, o comando *WHILE* é uma estrutura de controle que, baseado em uma condição lógica, determina a execução de uma instrução específica repetidamente.

DECLARE

@COUNT INT

,@MENSAGEM VARCHAR

SET @COUNT = 0

WHILE @COUNT < 10

BEGIN

SET @COUNT += 1

SET @MENSAGEM = @COUNT

IF @COUNT%2 !=0

PRINT 'LINHA ÍMPAR ' + @MENSAGEM

ELSE

PRINT 'LINHA PAR ' + @MENSAGEM

END

```
SQLQuery10.sql - M...ojetoTSQL (sa (52)) ×  
-- DECLARE  
-- @COUNT INT  
-- ,@MENSAGEM VARCHAR  
  
SET @COUNT = 0  
  
WHILE @COUNT < 10  
BEGIN  
    SET @COUNT += 1  
  
    SET @MENSAGEM = @COUNT  
  
    IF @COUNT%2 !=0  
        PRINT 'LINHA ÍMPAR ' + @MENSAGEM  
    ELSE  
        PRINT 'LINHA PAR ' + @MENSAGEM  
  
END  
  
100 %  
Mensagens  
LINHA ÍMPAR 1  
LINHA PAR 2  
LINHA ÍMPAR 3  
LINHA PAR 4  
LINHA ÍMPAR 5  
LINHA PAR 6  
LINHA ÍMPAR 7  
LINHA PAR 8  
LINHA ÍMPAR 9  
LINHA PAR 10  
91 %  
Consulta executada com êxito. MASTER\SQLEXPRESS (12.0 RTM) sa (52) ProjetoTSQL 00:00:00 0 linhas
```

Figura 17. Exemplo de uso do *WHILE*. Elaborada pelo autor.

4.2.1. Observações sobre o uso do *WHILE*

No incremento da variável de controle é realizada utilizando **Operador de Atribuição de Adição "+="**.

A instrução "SET @COUNT += 1" é equivalente a "SET @COUNT = @COUNT + 1".

1. Se você tem alguma noção de C++ ou C#, com certeza terá alguma familiaridade com esse tipo de atribuição.
2. Outra peculiaridade é a condição de divisão de fluxo, em que se verifica o *MOD* da divisão entre a variável de controle e o número 2.
3. Divisão de um número par por dois, o resto é sempre zero e números ímpares não.
4. Desta forma é possível distinguir números pares de ímpares.
5. Foi declarada a variável @mensagem como um *varchar*, porém atribuiu-se a esta variável o valor da variável @count que é um inteiro.
6. Não ocorre erro quando o código é executado. Isso é possível no T-SQL. Este tipo de situação chama-se *Conversão Implícita*.

5. ADIÇÃO DE COMENTÁRIOS

Comentários são informações que adicionamos ao código que desenvolvemos para documentar os scripts, permitindo que os outros desenvolvedores compreendam melhor o que escrevemos.

5.1. Comentários em Linha

Indica o texto fornecido pelo usuário. Os comentários podem ser inseridos em uma linha separada, aninhados no final de uma linha de comando *Transact-SQL* ou em uma instrução *Transact-SQL*. O servidor não avalia o comentário.

Use dois hífens (-) para comentários de linha única ou aninhados. Comentários inseridos com - são terminados pelo caractere de nova linha. Não há tamanho máximo para comentários. A tabela a seguir lista os atalhos de teclado que você pode usar para comentar e descomentar o texto.

--OPERADORES ARITMÉTICOS

DECLARE

 @VALOR1 *INT*

 ,@VALOR2 *INT*

SELECT

 @VALOR1 = 12

 ,@VALOR2 = 6

SELECT

 @VALOR1 + @VALOR2 AS [Resultado da Adição]

-- Quando usamos o prefixo "--", o *SQL Server* não interpreta aquela linha.

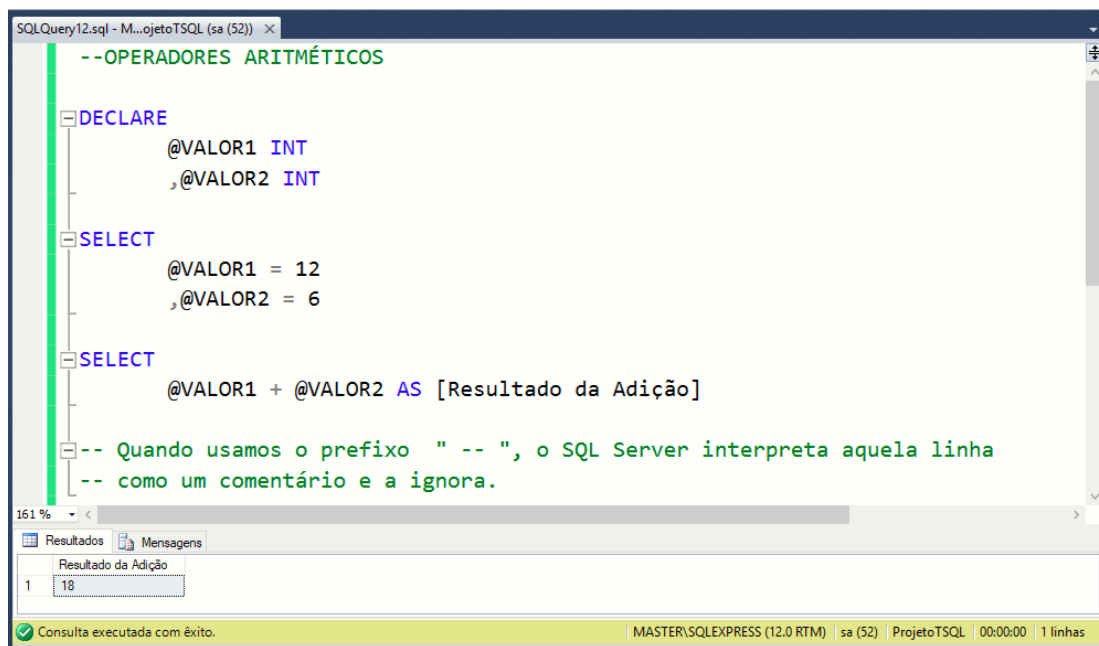


Figura 18. Exemplo de comentários em linha. Elaborada pelo autor.

5.2. Comentários com Múltiplas Linhas

Outra forma de adicionar comentários é através de barras com asteriscos, que permitem comentários com múltiplas linhas.

Indica o texto fornecido pelo usuário. O texto entre o `/ *` e `* /` não é avaliado pelo servidor.

Os comentários podem ser inseridos em uma linha separada ou em uma instrução *Transact-SQL*. Comentários de várias linhas devem ser indicados por `/ *` e `* /`. Uma convenção estilística geralmente usada para comentários de várias linhas é iniciar a primeira linha com `/ *`, linhas subsequentes com `**` e terminar com `* /`.

Não há tamanho máximo para comentários.

Comentários aninhados são suportados. Se o padrão de caractere `/ *` ocorrer em qualquer lugar dentro de um comentário existente, ele será tratado como o início de um comentário aninhado e, portanto, exigirá uma marca de fechamento de `* /` comentário. Se a marca de comentário de fechamento não existir, um erro será gerado.

```

/*
ESTE É UM COMENTÁRIO
COM VÁRIAS LINHAS
*/

DECLARE
    @VALOR1 INT
    ,@VALOR2 INT

SELECT
    @VALOR1 = 12
    ,@VALOR2 = 6

SELECT
    @VALOR1 + @VALOR2 AS [Resultado da Adição]

```

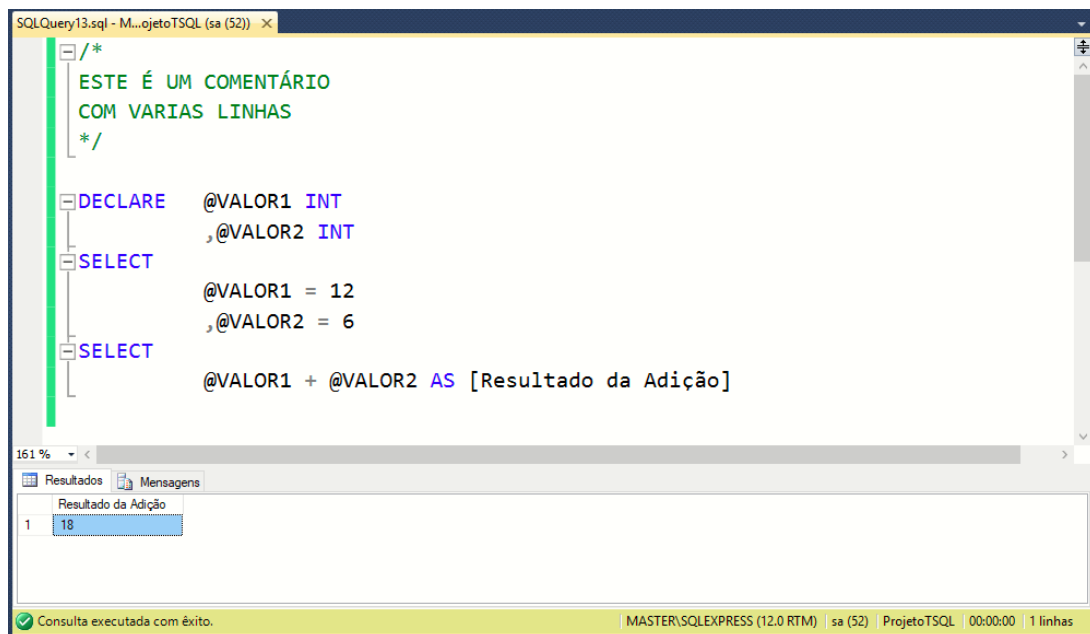


Figura 19. Exemplo de comentários em bloco. Elaborada pelo autor.

#FICA A DICA



- **Título:** Curso de SQL - Introdução e grupos de comandos - SQL Server #1
- **Ano:** 2013.
- **Sinopse:** O que é *SQL – Structured Query Language* SQL é uma Linguagem de Consulta Estruturada padrão para acesso a Bancos de Dados. Usada em inúmeros sistemas, como *MySQL*, *SQL Server*, *Oracle*, *Sybase*, *Access*, *DB2*, *PostgreSQL*, etc. Cada um desses sistemas pode utilizar um “dialetto” diferente do *SQL*, como *T-SQL* (*SQL Server*), *PL/SQL* (*Oracle*), *JET SQL* (*Access*), etc. Os comandos *SQL* podem ser divididos em quatro grupos principais: Comandos *DDL*, Comandos *DML*, Comandos *DCL* e Comandos *DQL*.

CONSIDERAÇÕES FINAIS.

Caro(a) Aluno(a), chegamos ao final desta unidade II com satisfação de tê-lo aqui conosco, com o sentimento de grande satisfação de saber que você, caro(a) aluno(a), conseguiu chegar até o fim. Parabéns!

Nesta unidade, você recebeu uma introdução à Concatenação de Strings, aos Operadores Aritméticos aos tipos de Operadores Lógicos, a utilização dos controladores de fluxos *IF/ELSE* e *WHILE* e muito mais.

Caro(a) Aluno(a), mais uma vez gostaria de parabenizá-lo pelo esforço e a dedicação para chegar até aqui. Desejo todo o sucesso que você puder conquistar em sua vida pessoal e profissional.

Sucesso e um grande abraço!

REFERÊNCIAS

- *LeBlanc, Patrick. **Microsoft SQL Server 2012 - Passo a Passo**. Porto Alegre. Bookmam Editora Ltda. 2014.*
- *Gonçalves, Rodrigo Ribeiro. **T-SQL com Microsoft SQL Server 2012 Express - na Prática**. Editora Érica Ltda. 2013.*
- *Gonçalves, Eduardo. **SQL - Uma abordagem para banco de dados Oracle**. Editora Casa do Código Ltda.*
- *Mistry, Ross; Misner, Stacia. **Introducing Microsoft SQL Server 2012**. Microsoft Press. 2012.*

UNIDADE 3

Professor Mestre Ricardo Vieira

Plano de Estudo:

1. INTEGRIDADE REFERENCIAL E RELACIONAMENTOS
2. CHAVES ESTRANGEIRAS
3. CRIAÇÃO DE CAMPOS CALCULADOS
4. JOINS
5. CASE, GROUP BY E FUNÇÕES DE AGREGAÇÃO
6. CONSIDERAÇÕES FINAIS
7. REFERÊNCIAS

Objetivos de Aprendizagem:

- Conceituar e contextualizar a Integridade Referencial relacionada aos valores de uma coluna em uma tabela de banco de dados.
- Compreender o conceito de Chaves Primárias do banco de dados.
- Estabelecer a importância da Criação de Campos Calculados com a virtualização das colunas que por padrão não estão armazenadas fisicamente no banco de dados.
- Conceituar e contextualizar a utilização *JOIN* para recuperar dados de duas ou mais tabelas com base em relações lógicas entre as tabelas.

INTRODUÇÃO

Caro(a) aluno(a), esta unidade inicia os estudos com uma breve introdução ao conceito a Integridade Referencial relacionada aos valores de uma coluna em uma tabela de banco de dados. Na primeira parte você poderá encontrar os assuntos de grande importância como as Restrições para Garantir a Integridade Referencial.

Na segunda parte, serão tratados os assuntos relacionados às Chaves Estrangeiras, com uma breve introdução às Chaves Primárias. Você poderá ver que a utilização da Chave Estrangeira 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.

Na terceira parte, será tratado a respeito da Criação de Campos Calculados que são colunas virtuais que por padrão não estão armazenadas fisicamente no banco de dados. Estão dispostas de maneira lógica.

E por último, mas não menos importante, trataremos sobre o conceito e a utilização do comando *JOIN*. Será apresentado os quatro tipos de comandos que são: *INNER JOINS*, *OUTER JOINS*, *LEFT JOIN* e o *RIGHT JOIN*.

Então vamos em frente, bons estudos e tenha um excelente aproveitamento!

#REFLITA

"A arte da educação deve ser cultivada em todos os aspectos, para se tornar uma ciência construída a partir do conhecimento profundo da natureza humana." - *Johann Heinrich Pestalozzi*.

Caro(a) aluno(a), deixo esta mensagem com o propósito de inspirar vocês a buscarem em cada momento uma oportunidade de aprendizagem e aperfeiçoamento intelectual e moral. Transformando assim o mundo com educação e conhecimento.

Referência: <https://citacoes.in/citacoes/102625-johann-heinrich-pestalozzi-a-arte-da-educacao-deve-ser-cultivada-em-todos-os/>

1. INTEGRIDADE REFERENCIAL E RELACIONAMENTOS



É a propriedade relacionada aos valores de uma coluna em uma tabela de banco de dados. Todos os valores dessa coluna têm uma correspondência em outra tabela. Se esta condição estiver satisfeita, podemos dizer que há uma integridade referencial entre as duas tabelas.

Quando colocamos uma coluna como chave estrangeira em uma tabela, assumimos responsabilidade com o banco de dados por assim dizer. As colunas pertencentes à chave estrangeira da tabela **A**, deve ter o mesmo domínio das colunas pertencentes à chave primária da tabela **B**. O valor de uma chave estrangeira em uma tabela A deve ser de chave primária da tabela **B**, ou então ser nulo. Sintetizando, uma tabela contém uma chave estrangeira, então o valor dessa chave só pode ser:

- Nulo. Neste caso pode, pois representa a inexistência de referência para uma linha da tabela.
- Igual ao valor de alguma chave primária na tabela referenciada.

Você pode perguntar como ficaria uma tabela com chave estrangeira nula. Vejamos:

Tabela 10. Exemplo de tabela Funcionário com chave estrangeira nula. Elaborada pelo autor.

NumReg	NomeFunc	DataAdmiso	Sexo	CdCargo
101	Luis José	10/8/2003	M	C3
104	Carlos Paulo	2/3/2004	M	C4
134	Jose Antonio	23/5/2002	M	C5
121	Jose Antonio	10/12/2001	M	C3
123	Pedro Sergio	29/6/2003	M	NULO
115	Roberto Fernando	15/10/2003	M	C3
22	Sergio Noel	10/2/2000	M	C2

Na linha de Pedro Sérgio o valor para *CdCargo* está nulo, o que pode significar que ainda não está alocado a nenhum cargo, ou foi promovido e não foi atualizado ainda no sistema.

O que importa é que ele não tem um cargo assinalado, o que é uma situação válida. O que não pode haver é um valor de chave estrangeira que não exista como chave primária de nenhuma linha da tabela referenciada, no caso a tabela Cargo.

Na definição de uma chave estrangeira devemos nos referenciar a uma chave primária de outra tabela.

1.1. Restrições para Garantir a Integridade Referencial

Existe um conjunto de regras de operação para um banco de dados relacional que coloca restrições, regras nas operações de atualização das tabelas do banco de dados, de forma a garantir e manter a integridade referencial.

- **PARENT DELETE RESTRICT:** (Deleção Restrita) Ao excluir (*deletar*) a tabela pai (*parent*), se ela possuir filhos relacionados (ou seja, se o cargo tiver funcionários registrados), a exclusão é impedida (*RESTRICT*). Isso evita que o banco de dados fique inconsistente, ou seja, linhas de Funcionário com valor de chave estrangeira inexistente como chave primária na tabela associada.

Outras opções para garantir a integridade de referências do banco de dados seriam excluir todos os filhos em cascata (*CASCADE*), fazendo com que todos os funcionários referenciam um cargo padrão, *CdCargo=C3* (Operador), por exemplo, ou fazer com que todos os funcionários fiquem sem cargo, *CdCargo = NULL*.

- **CHILD INSERT RESTRICT:** (Inclusão e Linha Restrita) Ao inserir um funcionário, caso seja obrigatório que já possua cargo associado, verifica se ele está relacionado a um departamento existente na tabela Departamento, senão impede a operação (*RESTRICT*).

- **CHILD UPDATE RESTRICT:** Ao atualizar (*UPDATE*) a chave estrangeira de uma tabela (*CHILD*), deve-se verificar se existe uma linha da tabela associada que possua como chave primária o novo valor da chave estrangeira, senão impede essa operação (*RESTRICT*).

A opção *Cascade* (Cascata) é sempre perigosa de ser utilizada em banco de dados, pois existe o risco de perder todos os dados existentes em uma determinada tabela se optar por apagar as suas linhas que estão associadas a uma determinada linha que será *deletada* da tabela que possui a chave primária referenciada.

2. CHAVES ESTRANGEIRAS

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 um banco de dados. Diante disso, visando apresentar esses conceitos, de uma maneira mais simples, foi colocado aqui, cada um dentro de seu escopo e mostrando como e quando utilizá-los.

2.1. Uma Introdução a 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 será 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:

- Chaves primárias não podem ser nulas;
- Cada registro na tabela deverá possuir uma, e somente uma chave primária;

- 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 do banco de dados que não é obrigatória. Nos casos em que ela (incremento automático) não for definida, é preciso garantir que não haverá valores repetidos nessa coluna;

- 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 integer PRIMARY KEY AUTOINCREMENT,
    Nome varchar (255),
    Endereco varchar (255),
    Cidade varchar (255)
);
```

Como o nosso banco de dados *SQL Server* ele cria a chave primária automaticamente, não será necessário acrescentar o comando *AUTOINCREMENT*, caso faça a inserção do comando, será apresentado erro na janela de consultas pois este não é o comando correto para fazer auto incremento no *SQL SERVER*. Este banco de dados utiliza o comando *IDENTITY()* para realizar este auto incremento. Nota: é permitido somente um campo *IDENTITY* por tabela

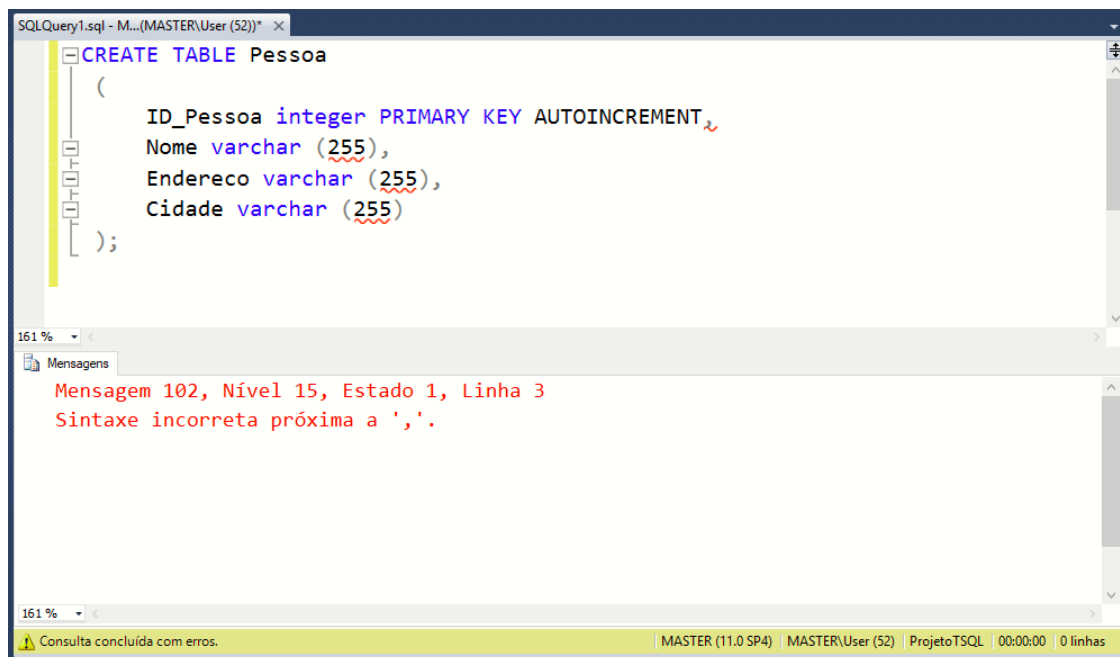


Figura 21. Gerando tabela Pessoa com chave primária com auto incremento que dá erro.
Elaborado pelo autor.

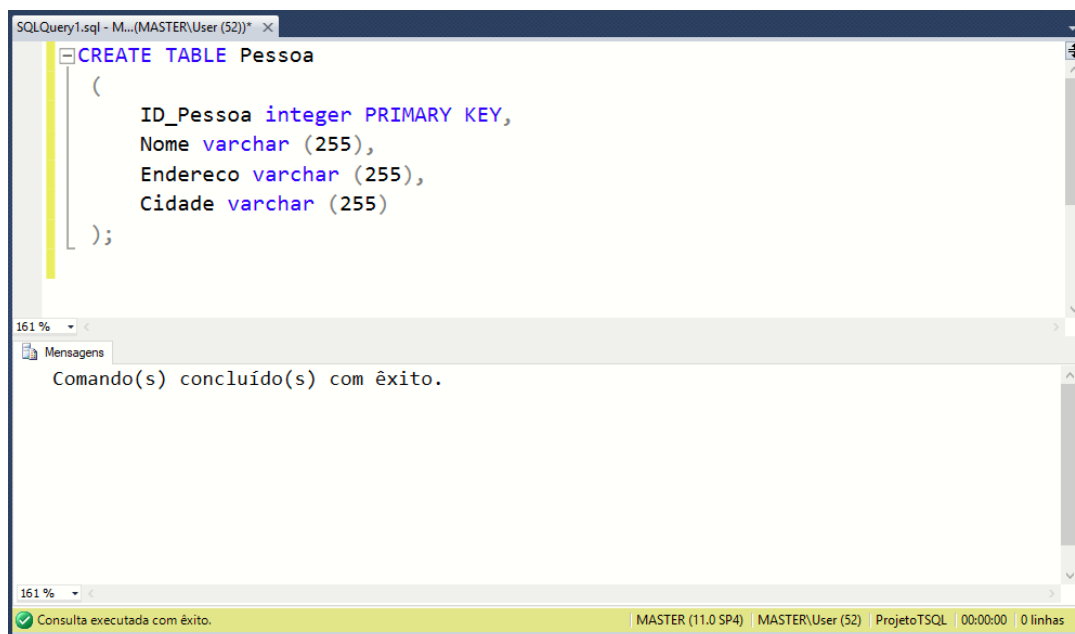


Figura 22. Gerando tabela Pessoa com chave primária. Elaborado pelo autor.

Linha 01: comando SQL para criação da tabela Pessoa na base de dados. Quando executado, irá criar a tabela com os campos definidos nas linhas 03 à 06;

Linha 03: 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;

Linhas 04 a 06: definição dos demais campos da tabela Pessoa.

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.

2.2. Chave Estrangeira, Propriamente Dita

A chave estrangeira, ou *foreign key*, é um conceito 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:

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

Um alerta muito importante 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 Figura 23 mostra a adição da chave estrangeira diretamente quando criamos a tabela:

```
CREATE TABLE Carro
(
    ID_Carro integer PRIMARY KEY,
    Nome varchar (255),
    Marca varchar (255),
    ID_Pessoa integer,
    CONSTRAINT fk_PesCarro FOREIGN KEY (ID_Pessoa) REFERENCES
Pessoa (ID_Pessoa)
);
```

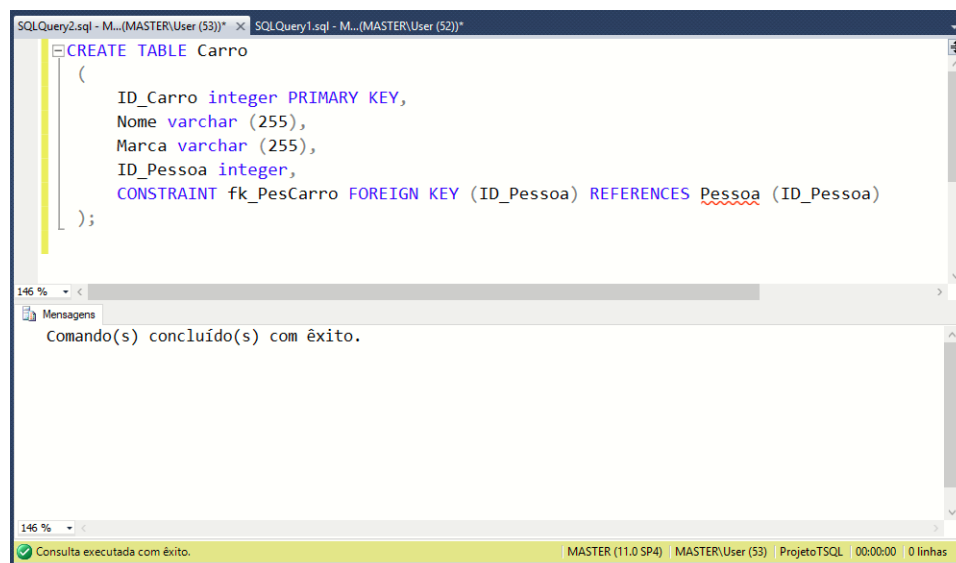


Figura 23. Chave estrangeira especificada no *CREATE TABLE*. Elaborada pelo autor.

Linha 01: comando *SQL* para criação da tabela *Carro* na base de dados. Quando executado, irá criar a tabela com os campos definidos nas linhas 03 a 07;

Linha 03: gera a chave primária da tabela *Carro*.

Linhas 04 e 05: inserção dos campos *Nome* e *Marca* do *Carro* na tabela;

Linha 06: criação do campo *ID_Pessoa*, do tipo inteiro (*integer*). Esse campo representa a chave estrangeira, e, portanto, receberá o valor do campo *ID_Pessoa* (a chave primária) da Pessoa "dona" do Carro, na modelagem de nossa base de dados exemplo;

Linha 07: definição da chave estrangeira propriamente dita. Para isso, 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.

Na Figura 24, abaixo, vemos a utilização do comando *ALTER TABLE* para inserir essa *CONSTRAINT* (restrição) em uma tabela já existente.

```
CREATE TABLE Carro
```

```
(  
    ID_Carro integer PRIMARY KEY,  
    Nome varchar(255),  
    Marca varchar(255),  
    ID_Pessoa integer,  
);
```

```
ALTER TABLE Carro
```

```
ADD CONSTRAINT fk_PesCarro FOREIGN KEY (ID_Pessoa)  
REFERENCES Pessoa (ID_Pessoa)
```

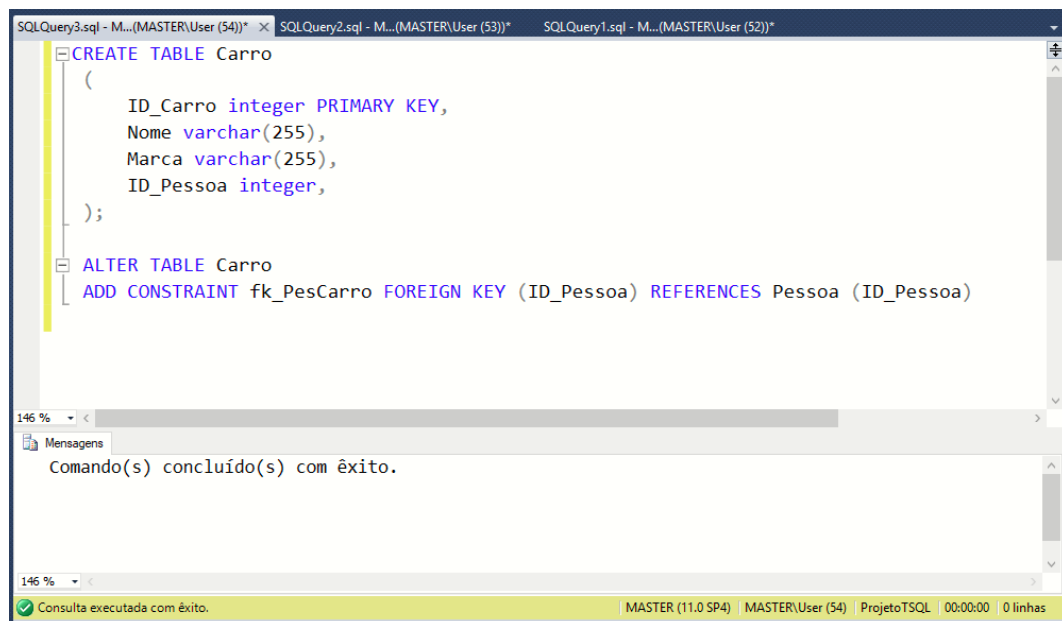


Figura 24. Chave estrangeira adicionada com o comando *ALTER TABLE*.

Elaborada pelo autor.

Linhas 01 a 07: comando para criação da tabela Carro sem a definição da chave estrangeira;

Linhas 09 e 10: comando para alterar a tabela Carro para adição da constraint, do tipo chave estrangeira;

Linha 10: comando para adição da constraint com nome "*fk_PesCarro*", do tipo *FOREIGN KEY*, para o campo *ID_Pessoa* da tabela Carro. Observe, também, a referência (palavra-chave *REFERENCES*) ao campo *ID_Pessoa* da tabela Pessoa.

Os comandos das Figuras 23 e 24, fazem as mesmas coisas, mas foram criadas por caminhos diferentes. A chave estrangeira gerada é a mesma, bem como o relacionamento entre as tabelas Pessoa e Carro.

Um ponto digno de nota é que tanto as chaves estrangeiras quanto as chaves primárias poderão ser compostas, ou seja, envolverem mais de um campo da tabela. Esse tipo de situação, no entanto, não é muito comum, pois prejudica principalmente a *performance* do acesso aos dados através de comandos *select*.

3. CRIAÇÃO DE CAMPOS CALCULADOS

Campos calculados são colunas virtuais que por padrão não estão armazenadas fisicamente no banco de dados. Estão dispostas de maneira lógica.

Uma coluna ou campo calculado poderá ter diversas aplicações, como listar por extenso uma data ou calcular o restante de uma operação aritmética entre dois ou mais campos de uma tabela. No caso da tabela vendas, um cálculo interessante é o produto da quantidade de itens de venda e o valor da venda. Para criar essa coluna, observe o seguinte exemplo:

ALTER TABLE Vendas

ADD TotalVendas **AS** (Quantidade * ValorVenda)

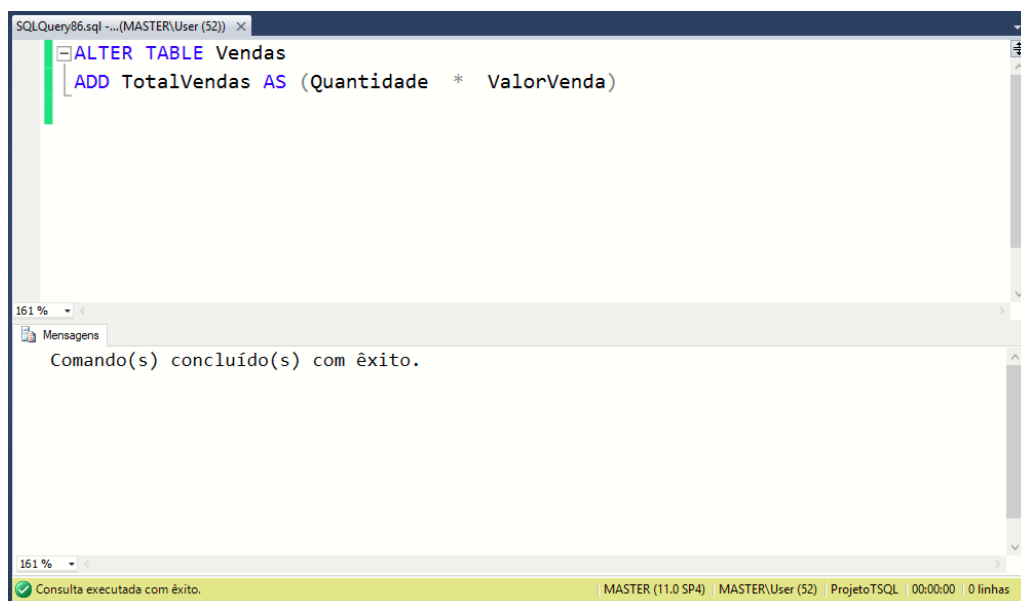


Figura 25. Exemplo de *ALTER TABLE*. Elaborado pelo autor.

Observe a sintaxe do comando *ALTER TABLE*. Logo depois do comando *ADD*, deverão ser informados o nome do campo e em seguida o seu cálculo, depois da instrução *AS*. Ao selecionar todas as colunas da tabela Vendas, ou selecionando a coluna diretamente pelo seu nome (*TotalVendas*), você poderá visualizar o produto da quantidade de itens e o valor da venda.

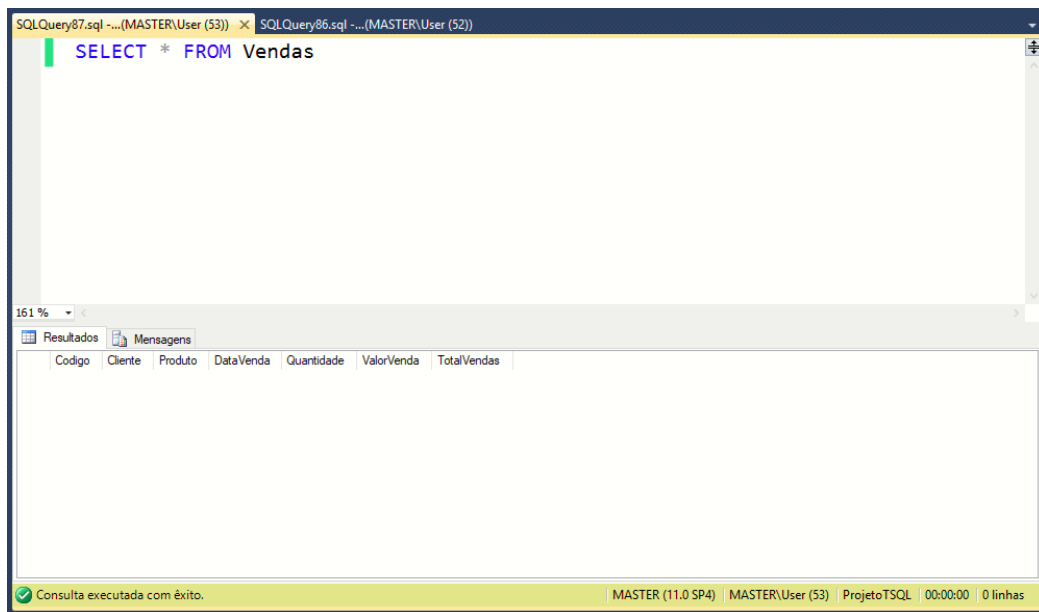


Figura 26. Exemplo de uma consulta com *SELECT*. Elaborado pelo autor.

4. JOINS

Usando *JOINS*, é possível recuperar dados de duas ou mais tabelas com base em relações lógicas entre as tabelas. *JOINS* indicam como o *Microsoft SQL Server* deve usar dados de uma tabela para selecionar as linhas em outra tabela.

Uma condição de *JOINS* define o modo como duas tabelas são relacionadas em uma consulta por:

- Especificando a coluna de cada tabela a ser usada para a junção. Uma condição de *JOINS* típica especifica uma chave estrangeira de uma tabela e sua chave associada na outra tabela.
- Especificando um operador lógico (por exemplo, = ou <>) a ser usado na comparação de valores das colunas.

#SAIBA MAIS#

Caro(a) aluno(a), dando continuidade no aprimoramento profissional, deixo como sugestão de leitura deste artigo que tenho certeza que te ajudará a sanar dúvidas te dará uma visão mais ampla da Linguagem *SQL*.

Este artigo mostra algumas técnicas para tornar o código armazenado em *scripts SQL* mais "polido", ou seja, mais adequado para receber modificações no futuro.

Referencia: <https://www.devmedia.com.br/linguagem-sql-torne-seu-codigo-sql-mais-legivel/38062>

4.1. Comando *INNER JOINS*

O padrão *ANSI SQL* define quatro tipos de *JOIN*, sendo *INNER*, *OUTER*, *LEFT* e *RIGHT*. O *INNER JOIN* consulta os registros de duas tabelas, verificando todos os registros de cada uma e seleciona os que têm valores em comum, com base no critério estabelecido no *JOIN*.

Observe o exemplo apresentado a seguir para que você possa entender melhor. No banco de dados *ProjetoTSQL*, criamos três tabelas sendo clientes, produtos e vendas. Estabelecemos dois relacionamentos entre Clientes e Vendas, e entre Produtos e Vendas, desta forma não é possível inserir vendas com clientes que não existem, mas é possível que existam clientes que não realizam vendas.

Para listar o nome de cada cliente e data da venda também de cada cliente, podemos usar o comando *INNER JOIN*:

```
SELECT NOME,  
       DataVenda  
FROM Clientes C  
INNER JOIN Vendas V ON (C.Codigo = V.Cliente)
```

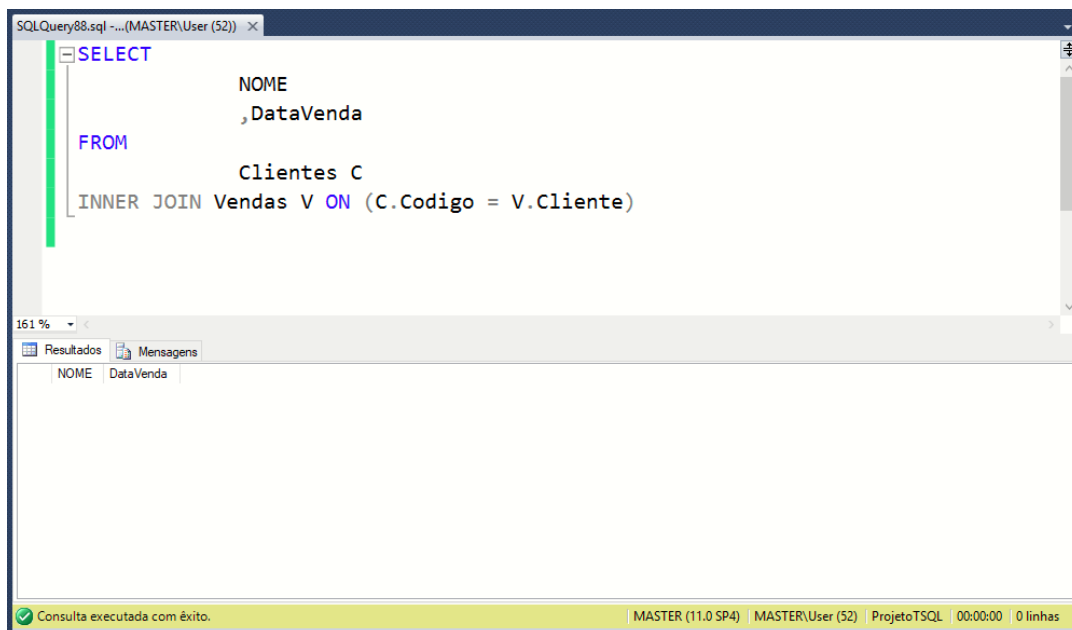


Figura 27. Exemplo de utilização do INNER JOIN. Elaborado pelo autor.

O *INNER JOIN* é considerado o tipo "padrão". Podemos executar a *query* anterior com a mesma sintaxe e eliminando o *INNER* e obtendo o mesmo resultado.

```

SELECT NOME,
       DataVenda
FROM Clientes C
JOIN Vendas V ON (C.Codigo = V.Cliente)

```

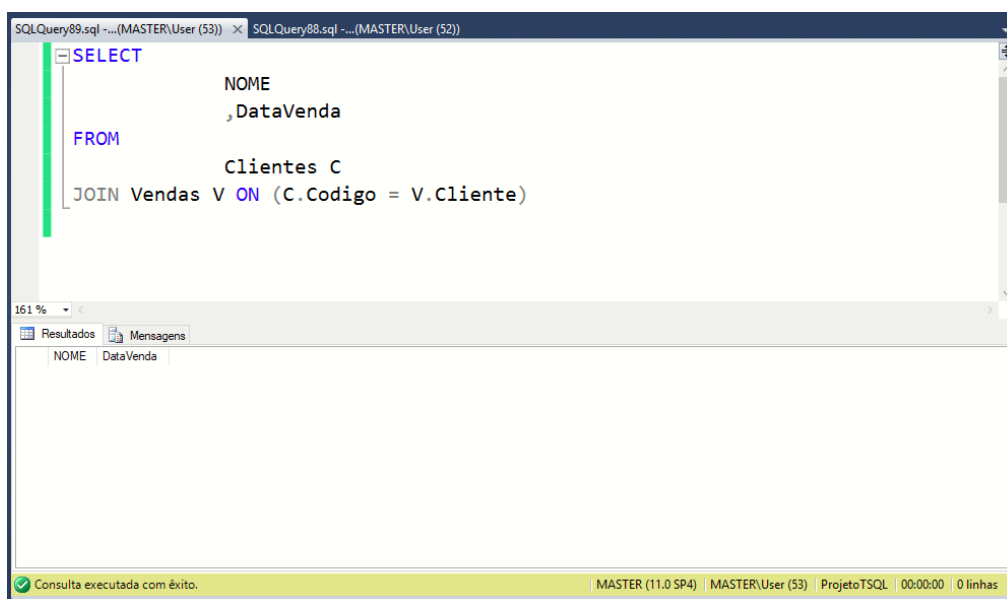


Figura 28. Exemplo de utilização do JOIN. Elaborado pelo autor.

Com o *JOIN* estamos trazendo a intersecção dos registros da tabela Clientes (ou conjunto A) e da tabela Vendas (conjunto B).

Comando OUTER JOINS

Podem ser especificadas apenas na cláusula *FROM*. As condições de *JOINS* combinam-se com as condições de pesquisa *WHERE* e *HAVING* para controlar as linhas selecionadas das tabelas base referenciadas na cláusula *FROM*.

Especificar as condições de *JOINS* na cláusula *FROM* ajuda a separá-las de qualquer outro critério de pesquisa que possa ser especificado em uma cláusula *WHERE* e é o método recomendado para a especificação de *JOINS*.

4.3. Comando LEFT JOIN

O comando *LEFT JOIN*, entre duas tabelas hipotéticas A e B, vai trazer todos os registros da tabela A independente do critério estabelecido no predicado do *JOIN*. OU seja, se a tabela A contém 100 registros e nenhum deles tem correspondente na outra, baseado no critério de comparação, a *query* ainda assim vai trazer 100 registros, porém onde a correspondência existir, os dados correspondentes serão resgatados.

Para listar todos os clientes, independentemente de haver vendas para eles, podemos usar o seguinte *script*:

```
SELECT NOME,  
       DataVenda  
FROM Clientes C  
LEFT JOIN Vendas V ON (C.Codigo = V.Cliente)
```

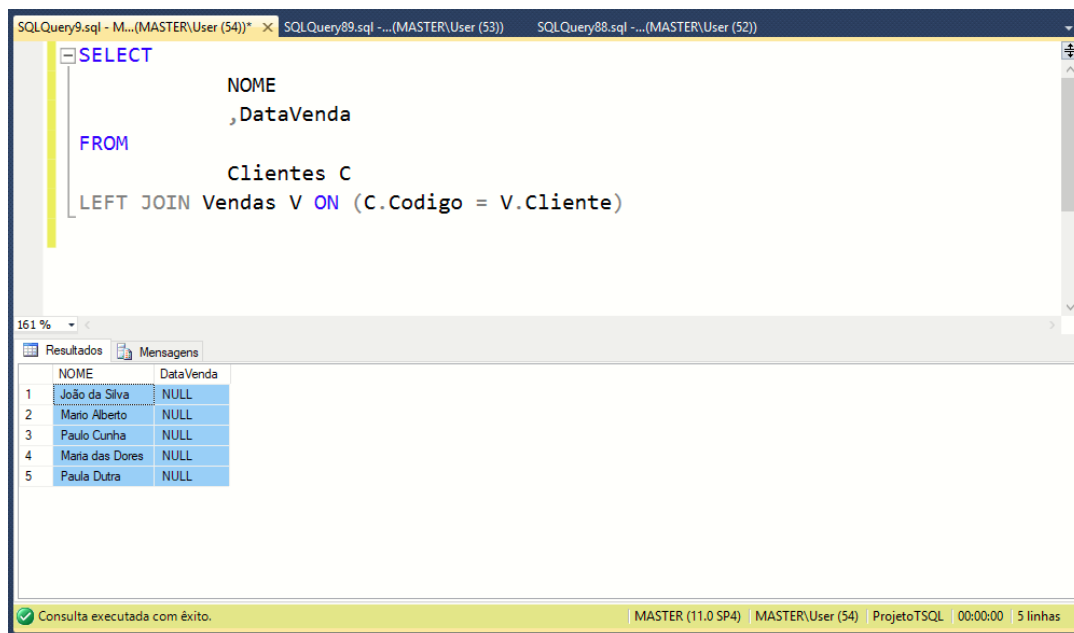



Figura 29. Exemplo de utilização do *LEFT JOIN*. Elaborado pelo autor.

4.4. Comando *RIGHT JOIN*

O comando *RIGHT JOIN* produz um resultado semelhante ao *LEFT JOIN*, porém com a inversão da comparação. Para que você possa entender melhor este conceito, em nosso banco de dados criamos uma tabela chamada *Produtos* que se relaciona com a tabela *Vendas*. Podemos listar a descrição de todos os produtos e datas da venda de cada um deles junto com os produtos que não realizam vendas através de uma *query* com o comando *RIGHT JOIN*.

```

SELECT DESCRICAO,
        DataVenda
FROM Vendas V
RIGHT JOIN Produtos P ON (P.Codigo = V.Cliente)
  
```

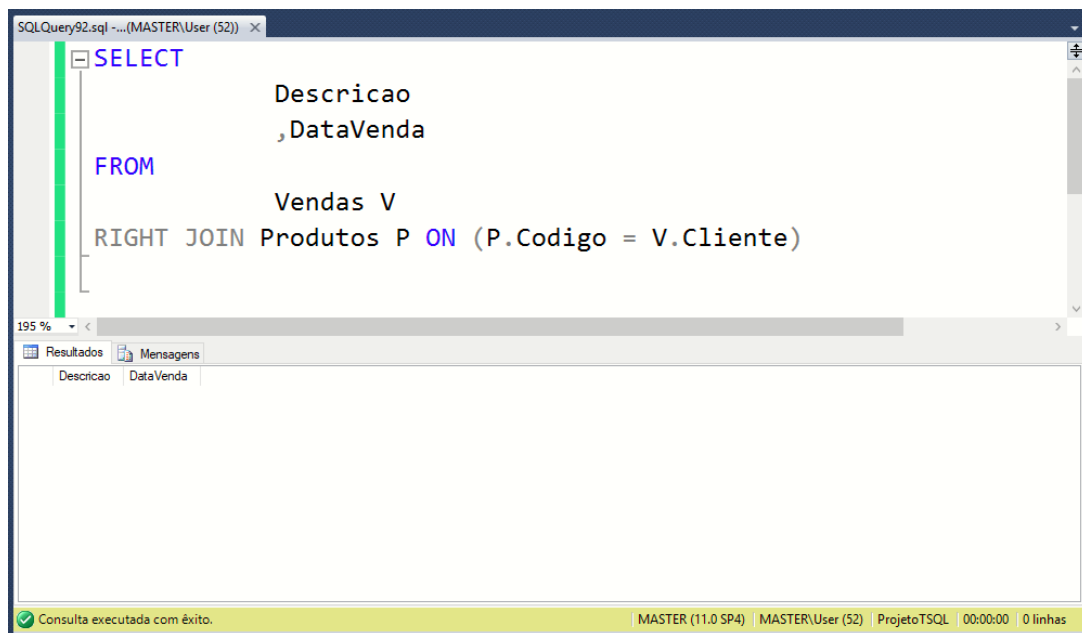


Figura 30. Exemplo de utilização do *RIGHT JOIN*. Elaborado pelo autor.

Podemos chegar exatamente ao mesmo resultado, usando *LEFT JOIN* e invertendo a posição das tabelas.

```
SELECT DESCRICAO,
       DataVenda
FROM Produtos P
LEFT JOIN Vendas V ON (P.Codigo = V.Cliente)
```

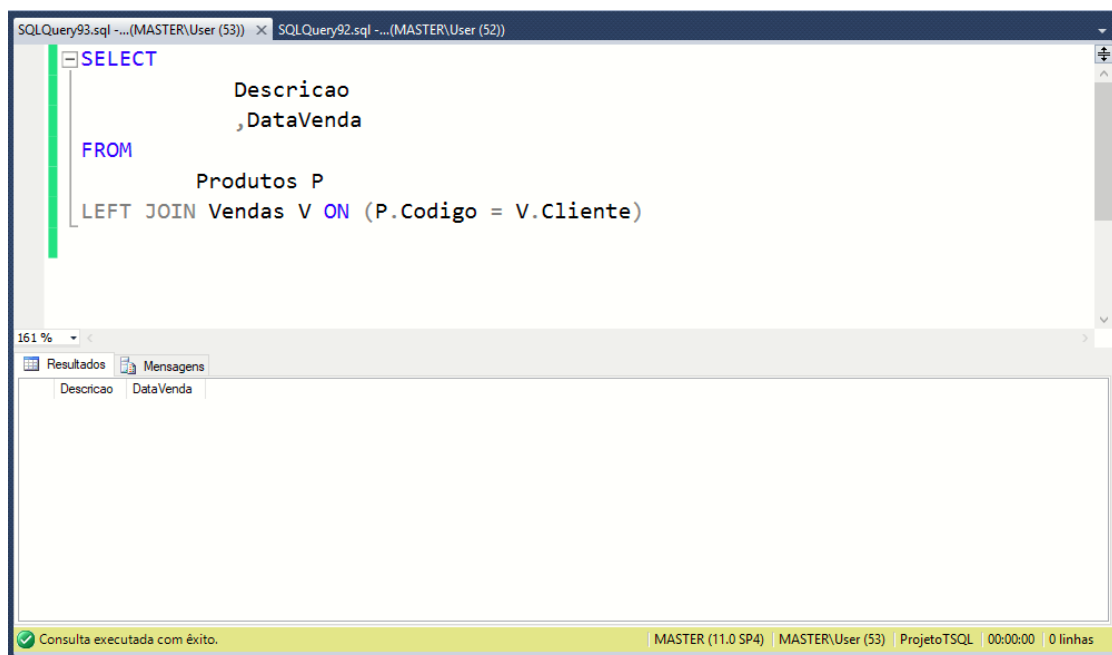
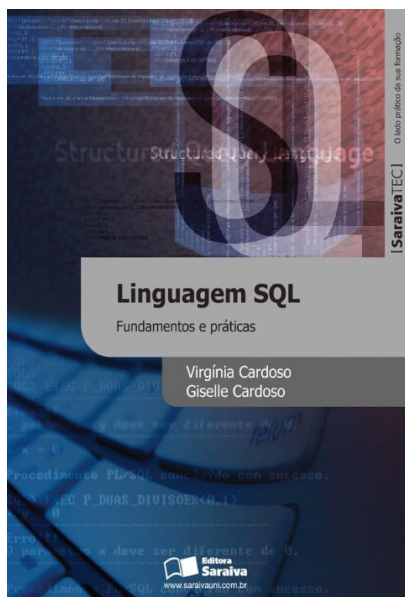


Figura 31. Exemplo de utilização do *JOIN*. Elaborado pelo autor.

#FICA A DICA#



- **Título:** LINGUAGEM SQL - fundamentos e prática
- **Autor:** Virgínia Cardoso, Giselle Cardoso.
- **Editora:** Editora Saraiva
- **Sinopse:** Este livro aborda os principais conceitos e fundamentos da linguagem de bancos de dados, a SQL. Entre os tópicos abordados estão: *DDL? Data Definition Language*, *DML? Data Manipulation Language* e consultas em SQL. Para um aprofundamento são apresentados tópicos da linguagem SQL avançada como consultas complexas, visões e *triggers*. Além de tratar também da

Visão geral de um *SGBD* e de Banco de dados e as novas tecnologias. Ao apresentar o assunto de maneira didática e com exercícios, as autoras direcionam os alunos a desenvolverem seu próprio banco de dados utilizando a linguagem SQL.

5. CASE, GROUP BY E FUNÇÕES DE AGREGAÇÃO

5.1. Comando CASE

O comando *CASE* comum em diversas linguagens de programação, avalia uma lista de condições verificadas em um ou mais campos e retorna apenas um de vários resultados possíveis.

A sintaxe do *CASE* não é complexa. Depois do comando *CASE*, as condições são avaliadas uma a uma com argumentos *WHEN* e *THEN*. Observe a sintaxe da construção do *CASE*:

```

SELECT
    CASE
        WHEN <Expressao logica 1>
        THEN <Resultado da Expressao 1>
        WHEN <Expressao logica 2>
        THEN <Resultado da Expressao 2>
        ELSE <Resultado fora das condicoes listadas>
    END
FROM <Tabela>

```

Veja um exemplo para que você possa entender melhor com, o usar o *CASE*. Observe o *script* seguinte:

```

SELECT Codigo
      ,Descricao
      ,CASE WHEN ValorVenda between 0 AND 100 THEN 'Bronze'
            WHEN ValorVenda between 100 AND 200 THEN 'Prata'
            WHEN ValorVenda between 200 AND 300 THEN 'Ouro'
            WHEN ValorVenda > 300 THEN
'Platina'
            WHEN ValorVenda IS NULL THEN
'Platina'
            ELSE 'Não classificado'
      END AS [Tipo Produto]
FROM Produtos

```

Por meio do exemplo anterior criamos uma classificação baseada no valor de venda de cada produto. Observe que na avaliação lógica, entre o comando *WHEN* e *THEN*, podemos ter qualquer condição e neste exemplo foi utilizado o comparador *BETWEEN*. Execute o script anterior e observe os resultados. Experimente também mudar as condições e os resultados de cada comparação.

5.2. GROUP BY

A cláusula *GROUP BY* permite agrupar registros baseados em um comando e, um critério estabelecido no argumento da instrução posicionado logo após o comando.

Uma cláusula da instrução *SELECT* que divide o resultado da consulta em grupos de linhas, normalmente, com a finalidade de executar uma ou mais agregações em cada grupo. A instrução *SELECT* retorna uma linha por grupo.

Com o comando *GROUP BY* podemos agrupar os registros e todos os clientes que realizaram mais de uma compra no mesmo dia aparecem agrupados. Execute o *script* a seguir:

```
SELECT NOME,  
       DataVenda  
FROM Clientes C  
INNER JOIN Vendas V ON (C.Codigo = V.Cliente)  
GROUP BY NOME, DataVenda
```

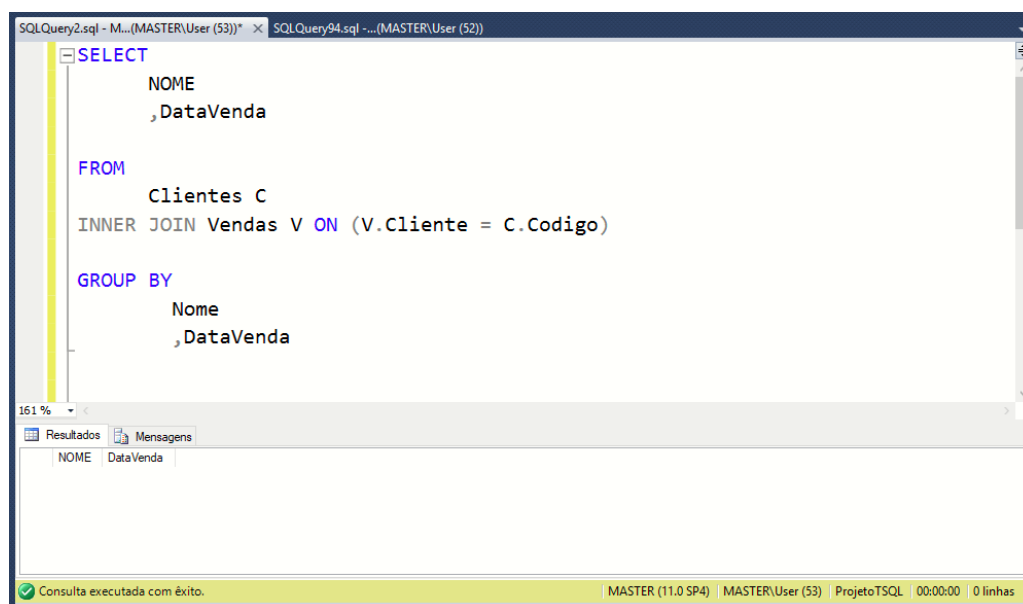


Figura 32. Exemplo de utilização do *GROUP BY*. Elaborado pelo autor.

5.3. Funções de Agregação

Podemos usar funções de agregação em uma coluna específica para realizar algum tipo de cálculo. Existem cinco funções de agregação, contagem de registros - COUNT(), soma de valores - SUM(), escolher um valor mínimo entre um conjunto de valores - MIN(), escolher um valor máximo entre um conjunto de valores - MAX() e média aritmética - AVG(). Observação quanto as funções de agregação, COUNT(), SUM (), AVG (), MIN () e MAX () nunca incluirão valores nulos em seus cálculos.

A função COUNT() pode ser usada para contar registros. Se você não especificar uma coluna e adicionar um asterisco como parâmetro para essa função, ela contará o número de registros retornado pela consulta. Por exemplo, podemos contar a quantidade de pessoas cadastradas em um sistema:

```
SELECT COUNT(*) QTD  
FROM PESSOAS
```

Fazer uma contagem de valores não nulos em uma coluna pode ser útil, portanto, observe que COUNT () também pode cumprir essa finalidade quando aplicada a uma coluna.

Vamos seguir para outras tarefas de agregação. Se você quisesse encontrar a média dos projetos de uma pasta de portfólio, teria que realizar uma consulta como esta:

```
SELECT AVG(INVESTIMENTOPREVISTO) as MEDIA_PROJETOS  
FROM PROJETOS
```

A função SUM() pode ser utilizada, por exemplo, para encontrar a soma de todos os projetos de um portfólio:

```
SELECT SUM(INVESTIMENTOPREVISTO) as SOMA_PROJETOS  
FROM PROJETOS
```

Já as funções MAX() e MIN() poderão ser utilizadas para encontrar os valores maiores e menores, respectivamente, de um conjunto de valores ou de uma coluna específica. Por exemplo, podemos utilizar estas funções para encontrar o maior e o menor valor dos projetos dentro de um portfólio.

```
SELECT MIN(INVESTIMENTOPREVISTO) as MENOR_VALOR_PROJETOS,  
       MAX(INVESTIMENTOPREVISTO) as MAIOR_VALOR_PROJETOS  
FROM PROJETOS
```

#FICA A DICA#



- **Título:** Curso de SQL: Primeiros passos na linguagem SQL
- **Ano:** 2017.
- **Sinopse:** Neste curso conheceremos os primeiros comandos da linguagem SQL (*Structured Query Language*), utilizada na estruturação e consulta de bancos de dados relacionais como MySQL e SQL Server.

CONSIDERAÇÕES FINAIS

Caro(a) Aluno(a)!

Estamos muito felizes porque você chegou até aqui. Sabemos que não foi fácil, pois, abriu mão do tempo livre para se dedicar aos estudos. Tempo esse que está sendo recompensado devido a sua dedicação na busca de um aprimoramento para a sua carreira profissional.

Nesta unidade você recebeu a introdução a respeito da importância da Integridade Referencial e Relacionamentos. Você pode ver também a respeito das Chaves Estrangeiras com uma direção a uma breve introdução a Chaves Primárias. Outro tópico, não menos importante que foi visto, Criação de Campos de agregação que é de extrema importância dentro da linguagem SQL. E por fim, pode ver a utilização dos tipos de *JOIN* seus aspectos básicos e muito mais.

Parabéns pelo seu esforço e dedicação! Continue com força e garra que na próxima unidade você terá muitas informações novas e interessantes, como *System Views* e *Sys.Objects*.

Estaremos juntos na próxima unidade. Até lá!

REFERÊNCIAS

- *LeBlanc, Patrick. **Microsoft SQL Server 2012 - Passo a Passo**. Porto Alegre. Bookmam Editora Ltda. 2014.*
- *Gonçalves, Rodrigo Ribeiro. **T-SQL com Microsoft SQL Server 2012 Express - na Prática**. Editora Érica Ltda. 2013.*
- *Gonçalves, Eduardo. **SQL - Uma abordagem para banco de dados Oracle**. Editora Casa do Código Ltda.*
- *Mistry, Ross; Misner, Stacia. **Introducing Microsoft SQL Server 2012**. Microsoft Press. 2012.*

UNIDADE 4

Professor Mestre Ricardo Vieira

Plano de Estudo:

1. SYSTEM VIEWS

1.1. SYS.OBJECTS

Objetivos de Aprendizagem:

- Conceituar e contextualizar a utilização do *System Views*.
- Compreender os tipos de *Sys.Objects*, suas características e suas usabilidades.

INTRODUÇÃO

Caro(a) aluno(a), esta unidade inicia os estudos com uma breve introdução ao conceito e a utilização do System Views dentro do SQL Server. Na segunda parte, será tratada os assuntos relacionados ao acesso ao banco de dados, listando os objetos utilizando Sys.Objects. Apresentando suas funções e suas características.

Então vamos em frente, bons estudos e tenha um excelente aproveitamento!

#REFLITA

"O sucesso não acontece por acaso. É trabalho duro, perseverança, aprendizado, estudo, sacrifício e, acima de tudo, amor pelo que você está fazendo ou aprendendo a fazer." (Pele).

Dessa forma, fica a reflexão para todos nós, com inspiração do "Rei" Pelé, se quisermos alcançar o sucesso devemos abrir mão do nosso conforto e comodidade para realmente superar os desafios, sejam eles através do trabalho, esporte e também dos estudos. Superação não é ser melhor que o seu adversário ou concorrente. Superação é você ser melhor a cada dia. Supere-se!

Um grande abraço!

Sucesso!

1. SYSTEM VIEWS



As *views* de sistema, ou *System Views*, permitem acesso aos metadados dos objetos que criamos durante o desenvolvimento. Acesso a essas informações é importante, pois assim podemos verificar a compatibilidade entre versões de aplicações, a preexistência de um determinado objeto no banco de dados antes de criá-lo, certificar se uma determinada coluna possui o número de caracteres adequado à necessidade do desenvolvimento antes de uma atualização, enfim as aplicações são diversas.

1.1. *Sys.Objects*

A *view Sys.Objects* lista todos os objetos contidos no banco de dados, sejam objetos de sistemas ou criados pelos usuários de banco de dados. Abra uma nova janela de query e execute o script abaixo:

```
SELECT *  
FROM sys.objects  
WHERE TYPE = 'u'
```

Você provavelmente deverá ter obtido o seguinte resultado:

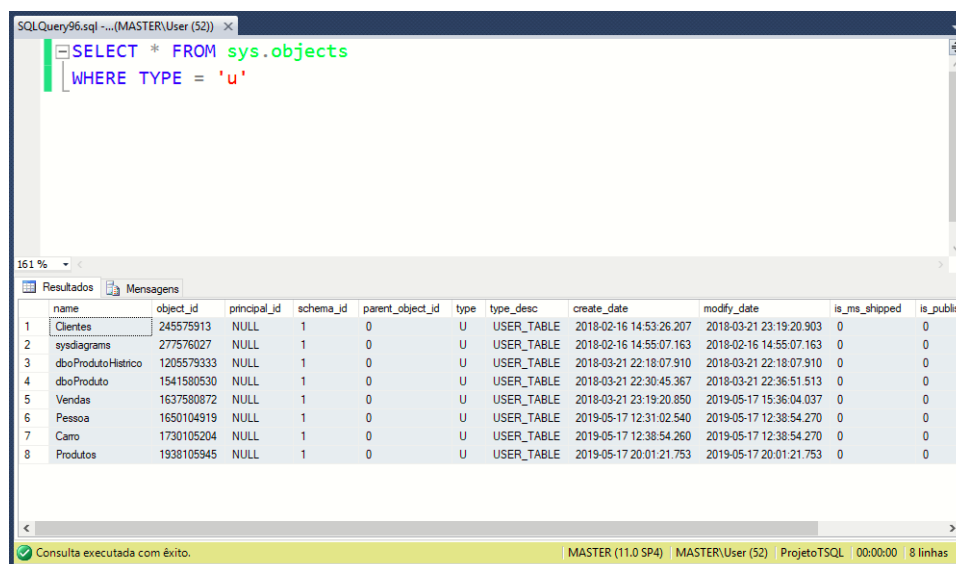


Figura 33. Exemplo de *SELECT* em *Sys.Object*. Elaborado pelo autor.

Essa query lista todas as tabelas criadas pelo usuário no banco de dados. Observe o filtro com o tipo de objeto "*TYPE = 'U'*", que define as tabelas criadas por usuário.

Uma das tabelas listadas é a *Sys.Diagrams*, que você provavelmente não se lembra de ter criado com os exemplos deste livro. Essa tabela foi criada pelo próprio *SQL Server* quando criamos o diagrama de banco de dados.

Na tabela seguinte estão listados todos os tipos de objetos que podemos criar no *SQL Server*, assim você pode filtrar o que precisa listar, assim como as principais colunas dessa view e uma breve descrição de cada uma delas. Destaque para as colunas *Create_Date* e *Modify_Date*, que definem a criação e a última modificação do objeto.

Tabela 11. Lista de objetos que poderá ser criada no *SQL Server*. Fonte docs.microsoft.com/

Nome da coluna	Tipo de dados	Descrição
name	sysname	Nome do objeto.
object_id	int	Número de identificação do objeto. É exclusivo em um banco de dados.
principal_id	int	ID do proprietário individual se for diferente do proprietário do esquema. Por padrão, os objetos contidos no esquema pertencem ao proprietário do esquema. No entanto, um

		<p>proprietário alternativo pode ser especificado usando a instrução <i>ALTER AUTHORIZATION</i> para alterar a propriedade.</p> <p>Será <i>NULL</i> se não houver nenhum proprietário individual alternativo.</p> <p>Será <i>NULL</i> se o tipo de objeto for um dos seguintes:</p> <p>C = Restrição <i>CHECK</i> D = <i>DEFAULT</i> (restrição ou autônomo) F = Restrição <i>FOREIGN KEY</i> PK = Restrição <i>PRIMARY KEY</i> R = Regra (estilo antigo, autônomo) TA = Gatilho de <i>assembly</i> (integração CLR) TR = Gatilho <i>SQL</i> UQ = Restrição <i>UNIQUE</i> EC = restrição de borda</p>
schema_id	int	<p>ID do esquema em que o objeto está contido.</p> <p>Os objetos do sistema de escopo de esquema estão sempre contidos nos esquemas sys ou <i>INFORMATION_SCHEMA</i>.</p>
parent_object_id	int	<p>ID do objeto ao qual este objeto pertence.</p> <p>0 = Não é um objeto filho.</p>
type	char(2)	<p>Tipo de objeto:</p> <p>AF = Função de agregação (<i>CLR</i>) C = Restrição <i>CHECK</i> D = <i>DEFAULT</i> (restrição ou autônomo) F = Restrição <i>FOREIGN KEY</i> FN = Função escalar <i>SQL</i> FS = Função escalar de <i>assembly</i> (<i>CLR</i>) FT = Função avaliada por tabela de <i>assembly</i> (<i>CLR</i>) IF = Função <i>SQL</i> com valor de tabela embutida IT = tabela interna P = Procedimento armazenado <i>SQL</i> PC = procedimento armazenado <i>Assembly</i> (<i>CLR</i>) PG = Guia de plano PK = Restrição <i>PRIMARY KEY</i></p>

		<p>R = Regra (estilo antigo, autônomo)</p> <p>RF = Procedimento de filtro de replicação</p> <p>S = Tabela base do sistema</p> <p>SN = Sinônimo</p> <p>SO = Objeto de sequência</p> <p>U = Tabela (definida pelo usuário)</p> <p>V = Exibição</p> <p>EC = restrição de borda</p> <p>Aplica-se a: do <i>SQL Server 2012</i> (11.x) ao <i>SQL Server 2017</i>.</p> <p>SQ = Fila de serviço</p> <p>TA = Gatilho <i>DML</i> de <i>assembly</i> (<i>CLR</i>)</p> <p>TF = Função com valor de tabela <i>SQL</i></p> <p>TR = Gatilho <i>DML</i> de <i>SQL</i></p> <p>TT = Tipo de tabela</p> <p>UQ = Restrição <i>UNIQUE</i></p> <p>X = Procedimento armazenado estendido</p> <p>Aplica-se ao: <i>SQL Server 2016</i> (13.x) por meio <i>SQL Server 2017</i>, Banco de dados <i>SQL do Azure</i>, <i>Azure SQL Data Warehouse</i>, <i>Parallel Data Warehouse</i>.</p> <p>ET = tabela externa</p>
type_desc	nvarchar(60)	<p>Descrição do tipo de objeto:</p> <p><i>AGGREGATE_FUNCTION</i></p> <p><i>CHECK_CONSTRAINT</i></p> <p><i>CLR_SCALAR_FUNCTION</i></p> <p><i>CLR_STORED_PROCEDURE</i></p> <p><i>CLR_TABLE_VALUED_FUNCTION</i></p> <p><i>CLR_TRIGGER</i></p> <p><i>DEFAULT_CONSTRAINT</i></p> <p><i>EXTENDED_STORED_PROCEDURE</i></p> <p><i>FOREIGN_KEY_CONSTRAINT</i></p> <p><i>INTERNAL_TABLE</i></p> <p><i>PLAN_GUIDE</i></p> <p><i>PRIMARY_KEY_CONSTRAINT</i></p> <p><i>REPLICATION_FILTER_PROCEDURE</i></p> <p><i>RULE</i></p> <p><i>SEQUENCE_OBJECT</i></p> <p>Aplica-se a: do <i>SQL Server 2012</i></p>

		(11.x) ao SQL Server 2017. <i>SERVICE_QUEUE</i> <i>SQL_INLINE_TABLE_VALUED_FUNCTION</i> <i>SQL_SCALAR_FUNCTION</i> <i>SQL_STORED_PROCEDURE</i> <i>SQL_TABLE_VALUED_FUNCTION</i> <i>SQL_TRIGGER</i> <i>SYNONYM</i> <i>SYSTEM_TABLE</i> <i>TABLE_TYPE</i> <i>UNIQUE_CONSTRAINT</i> <i>USER_TABLE</i> <i>VIEW</i>
create_date	datetime	A data em que o objeto foi criado.
modify_date	datetime	A data em que o objeto foi modificado pela última vez com uma instrução <i>ALTER</i> . Se o objeto for uma tabela ou uma exibição, <i>modify_date</i> também será alterado quando um índice <i>clusterizado</i> na tabela ou na exibição for criado ou alterado.
is_ms_shipped	bit	O objeto é criado por um componente interno do SQL Server.
is_published	bit	O objeto é publicado.
is_schema_published	bit	Apenas o esquema do objeto é publicado.

Você pode aplicar a *OBJECT_ID*, *OBJECT_NAME*, e *OBJECTPROPERTY()* funções internas para os objetos mostrados em *sys.Objects*.

Há uma versão desta exibição com o mesmo esquema, chamado *sys.system_objects*, que mostra os objetos do sistema. Não há outra exibição denominada *all_objects* que mostra os objetos de sistema e do usuário. Todas as três exibições do catálogo têm a mesma estrutura.

Nesta versão do SQL Server, um índice estendido, como um índice *XML* ou índice espacial, é considerado uma tabela interna em *sys.objects* (*type = IT* and *type_desc = INTERNAL_TABLE*). Para um índice estendido:

name é o nome interno da tabela de índice.

parent_object_id é o *object_id* da tabela base.

As colunas *is_ms_shipped*, *is_published* e *is_schema_published* são definidas como 0.

Subconjuntos de objetos podem ser exibidos usando exibições do sistema para um tipo específico de objeto, como:

- *sys.tables*
- *sys.views*
- *sys.procedures*

#FICA A DICA



- **Título:** Linguagem SQL Aprendendo a Falar a Língua dos Bancos de Dados
- **Autor:** Fabricio Burch Salvador
- **Editora:** Editora Viena.
- **Sinopse:** A linguagem SQL é hoje a língua do mundo dos bancos de dados. E embora existam variações nas funções disponíveis em cada sistema de banco de dados, e nas sintaxes de alguns comandos, o fato é que após anos (décadas) de desenvolvimento em caminhos diferentes.

1.1.1. Permissões

A visibilidade dos metadados em exibições do catálogo está limitada aos protegíveis que pertencem a um usuário ou para os quais o usuário recebeu permissão. Para obter mais informações, consulte *Metadata Visibility Configuration* em <https://docs.microsoft.com/pt-br/sql/relational-databases/security/metadata-visibility-configuration?view=sql-server-2017>

Uma utilização comum das System Views é a verificação da existência de um objeto antes de criá-lo. Execute o *script* em uma janela de *query*:

```
IF NOT EXISTS (SELECT * FROM SYS.OBJECTS
                WHERE TYPE = 'U'
                AND NAME = 'Clientes')

BEGIN
    CREATE TABLE Clientes(
        Codigo          int IDENTITY(1,1) PRIMARY KEY,
        Nome            VARCHAR (100)      NULL,
        Endereco        VARCHAR (100)      NULL,
        Telefone        VARCHAR (25)       NULL,
        Email           VARCHAR (100)      NULL,
        DataNascimento  DATETIME           NOT NULL
    )
END
```

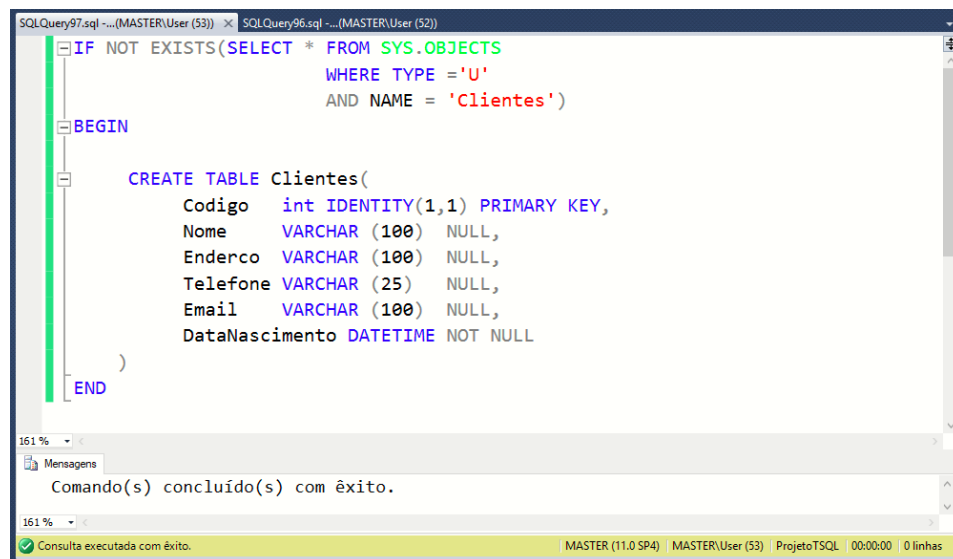


Figura 35. Exemplo verificação da existência de um objeto antes de criá-lo. Elaborado pelo autor.

Você poderá executar esses scripts quantas vezes quiser e nunca obterá erros na execução, tampouco criará outra tabela Clientes, pois o uso da função *EXISTS* acessando a view *Sys.Object* com o filtro pelo tipo "U" e o

name "Clientes" garante que o comando *CREATE TABLE* só será executado se a tabela ainda não existir.

Agora, execute o seguinte script em uma nova janela de *query*:

```
SELECT * FROM SYS.OBJECTS  
WHERE TYPE = 'U'  
AND Create_Date > GETDATE()-10
```

#SAIBA MAIS#

Caro(a) aluno(a), dando continuidade no aprimoramento profissional, deixo a sugestão de leitura deste artigo Processamento de consultas no SQL Server.

Este artigo apresenta como funciona o processamento de uma consulta no SQL Server, identificando os principais mecanismos e componentes desse sistema.

Referencia: <https://www.devmedia.com.br/processamento-de-consultas-no-sql-server/39174>

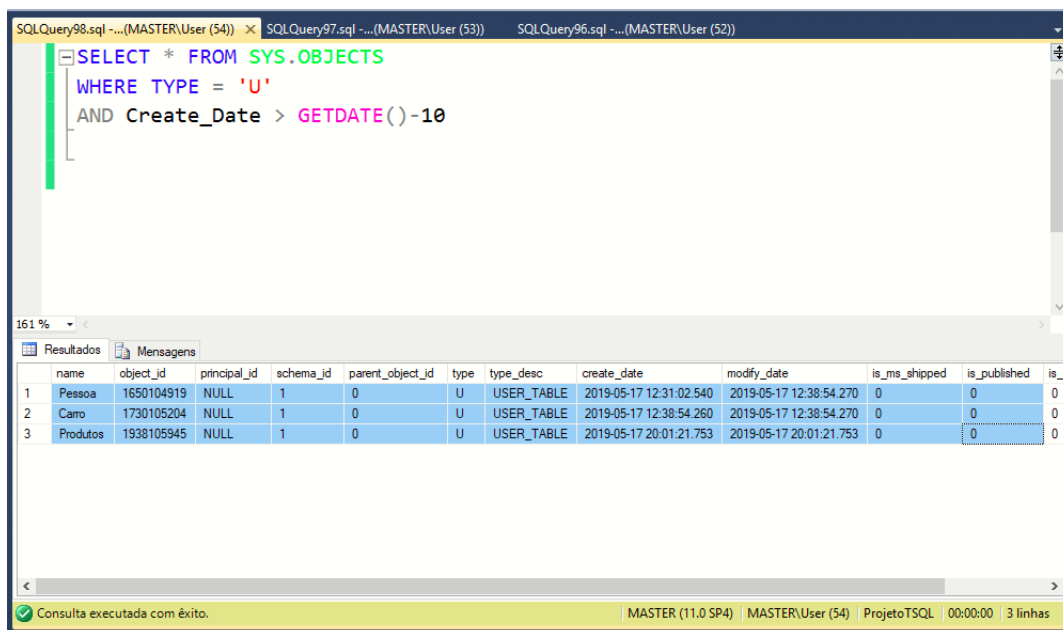


Figura 36. Consulta de todos os objetos criados nos últimos dez dias. Elaborado pelo autor.

O script anterior lista todos os objetos criados nos últimos dez dias. Os resultados vão depender de quando você criou os exemplos do livro e quando executou a query anterior. Se a diferença entre a data de criação e a data de execução da query anterior for menor que dez dias, você verá todas as tabelas criadas no banco de dados.

1.3. Sys.Columns

Sys.Columns é outra *View* de sistema, que lista as colunas de todas as tabelas e seus respectivos tipos de dados, "nulabilidade", a que tabelas estão vinculadas, entre outras informações.

Tabela 12. Tabela *Sys.Columns*, que lista as colunas de todas as tabelas e seus respectivos tipos de dados. Fonte: docs.microsoft.com.

Nome da coluna	Tipo de dados	Descrição
object_id	int	ID do objeto ao qual esta coluna pertence.
nome	sysname	Nome da coluna. É exclusiva no objeto.
column_id	int	ID da coluna. É exclusiva no objeto. Os IDs de coluna podem não ser sequenciais.

system_type_id	tinyint	ID do tipo de sistema da coluna.
user_type_id	int	D do tipo da coluna, como definido pelo usuário. Para retornar o nome do tipo, Junte-se para o Types essa coluna de exibição do catálogo.
max_length	smallint	Comprimento máximo (em bytes) da coluna. -1 = a coluna é do tipo de dados varchar (max), nvarchar (max), varbinary (max), ou xml. Para texto colunas, o valor de max_length será 16 ou o valor definido por sp_tableoption 'text in row'.
precision	tinyint	Precisão da coluna se tiver base numérica; Caso contrário, 0.
scale	tinyint	Escala da coluna se numérica; caso contrário é 0.
collation_name	bit	Nome do agrupamento da coluna se baseados em caracteres; Caso contrário, nulo.
is_nullable	bit	1 = A coluna permite valor nulo.

Você poderá listar todas as colunas e suas respectivas tabelas com o seguinte script:

```
SELECT o.name AS Tabela,
       c.name AS Coluna
FROM Sys.Objects o
JOIN Sys.Columns c ON (O.object_id = C.object_id)
WHERE O.type = 'U'
```

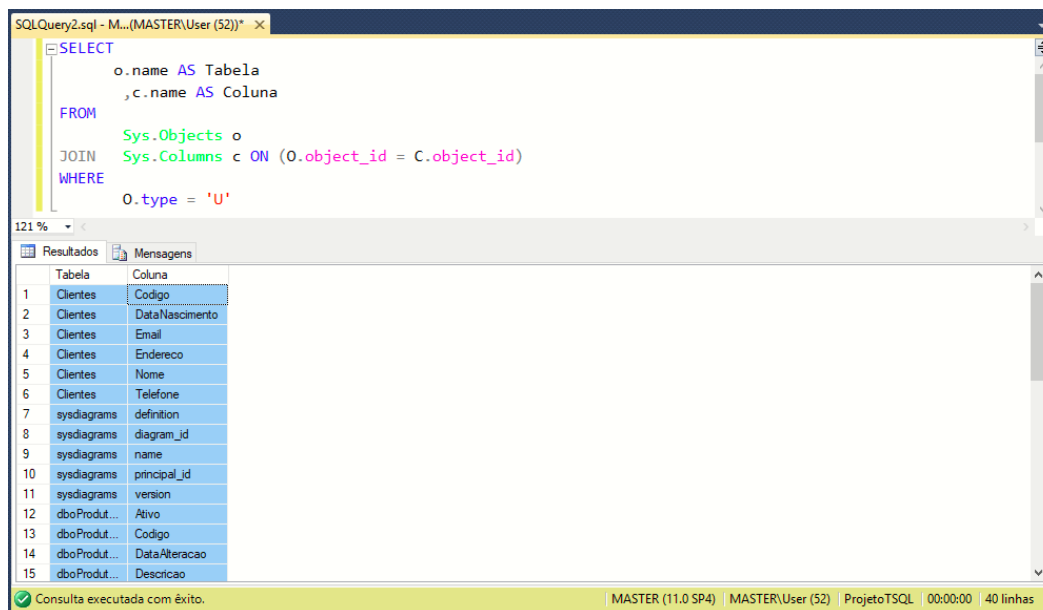


Figura 38. Consulta utilizando *Sys.Columns*. Elaborado pelo autor.

#FICA A DICA#



Título: Curso de Comando *Select* no SQL

• **Ano:** 2018.

• **Sinopse:** Consultas em

SQL são amplamente utilizadas em diversos tipos de sistemas que utilizam bancos de dados relacionais. Neste curso aprenderemos a realizar algumas das consultas mais comuns utilizando a linguagem SQL. Exploraremos as diversas possibilidades de uso da instrução SELECT, em conjunto com outras cláusulas, funções e operadores muito utilizados.

CONSIDERAÇÕES FINAIS

Caro(a) Aluno(a)!

Chegamos ao final desta unidade IV com satisfação de tê-lo aqui conosco, com o sentimento de grande satisfação de saber você querido (a) aluno (a) conseguiu chegar até o fim. Parabéns!

Nesta unidade, você recebeu uma introdução aos conceitos System Views e Sys.Objects.

Procuramos apresentar os conceitos sobre a utilização dessas duas funções existentes no SQL Server, juntamente com as aplicações dos exemplos de cada capítulo.

Caro (a) Aluno (a), mais uma vez gostaria de parabenizá-lo pelo esforço e a dedicação para chegar até aqui. Desejo todo o sucesso que você puder conquistar em sua vida pessoal e profissional.

Sucesso!

Um grande abraço!

REFERÊNCIAS

- *LeBlanc, Patrick. **Microsoft SQL Server 2012 - Passo a Passo**. Porto Alegre. Bookmam Editora Ltda. 2014.*
- *Gonçalves, Rodrigo Ribeiro. **T-SQL com Microsoft SQL Server 2012 Express - na Prática**. Editora Érica Ltda. 2013.*
- *Gonçalves, Eduardo. **SQL - Uma abordagem para banco de dados Oracle**. Editora Casa do Código Ltda.*
- *Mistry, Ross; Misner, Stacia. **Introducing Microsoft SQL Server 2012**. Microsoft Press. 2012.*

CONCLUSÃO

Chegamos ao final do nosso estudo sobre Arquitetura SQL Server. Foi possível entender os conceitos principais da plataforma, sua origem e seus princípios básicos.

Também estudamos a forma de instalação do ambiente de desenvolvimento do SQL SERVER e ferramentas auxiliares. Além disso, pudemos estudar sobre a construção de scripts, uma excelente forma de realizar cargas iniciais em projetos novos. Também aprendemos os comandos essenciais em SQL para definição e manipulação dos dados de nossa aplicação.

Ainda dentro deste contexto, estudamos tipos de operadores e fluxos de controle. Estas ferramentas de desenvolvimento são muito úteis para qualquer linguagem de programação e não diferente para o SQL. Essa técnica deve ser estudada a fundo por você, caro(a) leitor(a), pois pode lhe auxiliar em projetos futuros.

Outro assunto que foi abordado, e se caracteriza como fundamento principal de um banco relacional, é a integridade referencial, chaves e JOINS. Este princípio é indispensável para o bom desenvolvimento de arquiteturas de banco para nossas aplicações. Caso ainda tenha dúvidas sobre este conteúdo, sugiro que revise o material e estudo das dicas que deixei durante a unidade III.

Para finalizar nosso estudo e concretizar o aprendizado dessa incrível ferramenta estudamos alguns objetos específicos do SQL SERVER como o System Views e o Sys.Objects e toda a potencialidade embutidas nestes objetos.

Perceba que agora você está munido de uma grande ferramenta de mercado, capaz de facilitar a vida dos programadores no dia a dia. O poder

dessa ferramenta já é visto e explorado pelas empresas que possuem uma demanda muito grande por armazenamento de informações e centralização de regras de negócio.

E não é somente isso, a demanda extrapola o conceito de produtividade e atinge até as áreas principais da empresa, sempre com foco em redução de custos e aumento de lucratividade, tornando os produtos e serviços disponíveis em tempo real.

Apresentei muitas informações durante o desenvolvimento deste livro e acreditem: essas informações mudarão e todo esse cenário é apresentado a você por uma razão, o desenvolvimento está em constante evolução, principalmente na área de armazenamento de informações.

Então uma dica é nunca parar de estudar o assunto e transformar os novos conhecimentos adquiridos em práticas do dia a dia. Assim, terá a chance de alcançar o sucesso tão almejado. O primeiro passo já foi dado, agora é com você.

Encerro este trabalho com uma pequena reflexão da poetisa Rosana Hermann, a qual afirma que “Quem impede você de experimentar a vida? Alguém? Se for alguém, livre-se da pessoa. Se for você, liberte-se de suas amarras. A vida não tem outro objetivo senão a experiência de estar aqui. Vai, é a sua vez”.

Um abraço e até a próxima!

Prof. Ricardo Vieira.