Manipulando dados

- Constantes;
- nserção de dados;
- Utilização de TOP em uma instrução INSERT;
- OUTPUT;
- Atualização e exclusão de dados;
- UPDATE;
- DELETE;
- OUTPUT para DELETE e UPDATE;
- Transações.

Esta Leitura Complementar refere-se ao conteúdo das Aulas 10 a 13.

Editora IMPACTA

1.1. Constantes

As constantes, ou literais, são informações fixas que, como o nome sugere, não se alteram no decorrer do tempo. Por exemplo, o seu nome escrito no papel é uma constante e a sua data de nascimento é outra constante. Existem regras para escrever constantes no SQL:

- **Constantes de cadeia de caracteres (CHAR e VARCHAR)**

São sequências compostas por quaisquer caracteres existentes no teclado. Este tipo de constante deve ser escrito entre apóstrofos:

```
'IMPACTA TECNOLOGIA', 'SQL-SERVER', 'XK-1808/2',
'CAIXA D''AGUA'
```

Se o conteúdo do texto possuir o caractere apóstrofo, ele deve ser colocado duas vezes, como mostrado em CAIXA D'AGUA.

- **Cadeias de caracteres Unicode**

Semelhante ao caso anterior, estas constantes devem ser precedidas pela letra maiúscula **N** (identificador):

```
N'IMPACTA TECNOLOGIA', N'SQL-SERVER', N'XK-1808/2'
```

- **Constantes binárias**

São cadeias de números hexadecimais e apresentam as seguintes características:

  - Não são incluídas entre aspas;
  - Possuem o prefixo **0x**.

Veja um exemplo:

```
0xff, 0x0f, 0x01a0
```

- **Constantes datetime**

Utilizam valores de data incluídos em formatos específicos. Devem ser incluídos entre aspas simples:

```
'2009.1.15', '20080115', '01/15/2008', '22:30:10', '2009.1.15
22:30:10'
```

O formato da data pode variar dependendo de configurações do SQL. Podemos também utilizar o comando **SET DATEFORMAT** para definir o formato durante uma seção de trabalho.

- **Constantes bit**

Não incluídas entre aspas, as constantes **bit** são representadas por **0** ou **1**. Uma constante desse tipo será convertida em **1**, caso um número maior do que **1** seja utilizado.

```
0, 1
```

- **Constantes float e real**

São constantes representadas por notação científica:

```
2.53E4   2.53 x 10⁴        2.53 x 10000      25300
4.5E-2   4.5 / 10²         4.5 / 100         0.045
```

- **Constantes integer**

São representadas por uma cadeia de números sem pontos decimais e não incluídos entre aspas. As constantes **integer** não aceitam números decimais, somente números inteiros:

```
1528
817215
5
```

Nunca utilize o separador de milhar.

- **Constantes decimal**

São representadas por cadeias numéricas com ponto decimal e não incluídas entre aspas:

```
162.45
5.78
```

O separador decimal sempre será o ponto, independentemente das configurações regionais do Windows.

- **Constantes uniqueidentifier**

É uma cadeia de caracteres que representa um GUID. Pode ser especificada como uma cadeia de binários ou em um formato de caracteres:

```
0xff19966f868b11d0b42d00c04fc964ff
'6F9619FF-8B86-D011-B42D-00C04FC964FF'
```

- **Constantes money**

São precedidas pelo caractere cifrão ($). Este tipo de dado sempre reserva quatro posições para a parte decimal. Os algarismos além da quarta casa decimal serão desprezados.

```
$1543.56
$12892.6534
$56.275639
```

No último exemplo, será armazenado apenas 56.2756.

1.2. Inserindo dados

Para acrescentar novas linhas de dados em uma tabela, utilize o comando **INSERT**, que possui a seguinte sintaxe:

```
INSERT [INTO] <nome_tabela>
[ ( <lista_de_colunas> ) ]
{ VALUES ( <lista_de_expressoes1> )
      [, (<lista_de_expressoes2>)] [,...] |
<comando_select> }
```

Em que:

- **<lista_de_colunas>**: É uma lista de uma ou mais colunas que receberão dados. Os nomes das colunas devem ser separados por vírgula e a lista deve estar entre parênteses;

- **VALUES (<lista_de_expressoes>)**: Lista de valores que serão inseridos em cada uma das colunas especificadas em **<lista_de_colunas>**.

Para inserir uma única linha em uma tabela, o código é o seguinte:

```
   --Caso a tabela não tenha sido criada
   CREATE TABLE TB_ALUNO
   (        COD_ALUNO               INT IDENTITY PRIMARY KEY,
            NOME                    VARCHAR(30),
            DATA_NASCIMENTO         DATETIME,
            IDADE                   TINYINT,
            E_MAIL             VARCHAR(50),
            FONE_RES                CHAR(9),
            FONE_COM                CHAR(9),
            FAX                     CHAR(9),
            CELULAR                 CHAR(9),
            PROFISSAO               VARCHAR(40),
            EMPRESA                 VARCHAR(50) );
   GO

   INSERT INTO TB_ALUNO
(NOME, DATA_NASCIMENTO, IDADE, E_MAIL, FONE_RES, FONE_COM, FAX,
 CELULAR, PROFISSAO, EMPRESA )
   VALUES
   ('CARLOS MAGNO', '1959.11.12', 53, 'magno@magno.com',
    '23456789','23459876','', '998765432',
    'ANALISTA DE SISTEMAS', 'IMPACTA TECNOLOGIA');

   -- Consultar os dados inseridos na tabela
   SELECT * FROM TB_ALUNO;
```

Podemos inserir várias linhas em uma tabela com o uso de vários comandos **INSERT** ou um único:

```
   INSERT INTO TB_ALUNO
   (NOME, DATA_NASCIMENTO, IDADE, E_MAIL,
   FONE_RES, FONE_COM, FAX, CELULAR, PROFISSAO, EMPRESA)
   VALUES
   ('André da Silva', '1980.1.2', 33, 'andre@silva.com',
    '23456789','23459876','', '998765432',
    'ANALISTA DE SISTEMAS', 'SOMA INFORMÁTICA'),
    ('Marcelo Soares', '1983.4.21', 30, 'marcelo@soares.com',
    '23456789','23459876','', '998765432',
    'INSTRUTOR', 'IMPACTA TECNOLOGIA');

   -- Consultar os dados da tabela
   SELECT * FROM TB_ALUNO;
```

Podemos também fazer **INSERT** de **SELECT**:

```
CREATE TABLE ALUNOS2
 (       NUM_ALUNO               INT,
         NOME                    VARCHAR(30),
         DATA_NASCIMENTO         DATETIME,
         IDADE                   TINYINT,
         E_MAIL             VARCHAR(50),
         FONE_RES                CHAR(8),
         FONE_COM                CHAR(8),
         FAX                     CHAR(8),
         CELULAR                 CHAR(9),
       PROFISSAO                 VARCHAR(40),
       EMPRESA              VARCHAR(50) );

INSERT INTO ALUNOS2
SELECT * FROM TB_ALUNO;
```

Não é necessário determinar os nomes das colunas na sintaxe do comando **INSERT** quando os valores forem inseridos na mesma ordem física das colunas no banco de dados. Já para valores inseridos aleatoriamente, é preciso especificar exatamente a ordem das colunas. Esses dois modos de utilização do comando **INSERT** são denominados **INSERT** posicional e **INSERT** declarativo.

1.2.1. INSERT posicional

O comando **INSERT** é classificado como posicional quando não especifica a lista de colunas que receberão os dados de **VALUES**. Nesse caso, a lista de valores precisa conter todos os campos, exceto o **IDENTITY**, na ordem física em que foram criadas no comando **CREATE TABLE**. Veja o exemplo a seguir:

```
INSERT INTO TB_ALUNO
VALUES
('MARIA LUIZA', '1997.10.29', 15, 'luiza@luiza.com',
 '23456789','23459876','', '998765432',
 'ESTUDANTE', 'COLÉGIO MONTE VIDEL');

 -- Consultando os dados
SELECT * FROM TB_ALUNO;
```

1.2.2. INSERT declarativo

O **INSERT** é classificado como declarativo quando especifica as colunas que receberão os dados da lista de valores. Veja o próximo exemplo:

```
INSERT INTO TB_ALUNO
(NOME, DATA_NASCIMENTO, IDADE, E_MAIL,
 FONE_RES, FONE_COM, FAX, CELULAR,
 PROFISSAO, EMPRESA )
VALUES
('PEDRO PAULO', '1994.2.5', 19, 'pedro@pedro.com',
 '23456789','23459876','', '998765432',
 'ESTUDANTE', 'COLÉGIO MONTE VIDEL');

-- Consultando os dados
SELECT * FROM TB_ALUNO;
```

Quando utilizamos a instrução **INSERT** dentro de aplicativos, stored procedures ou triggers, deve ser usado o **INSERT** declarativo, pois, se houver alteração na estrutura da tabela (inclusão de novos campos), ele continuará funcionando, enquanto que o **INSERT** posicional provocará erro.

1.3. Utilizando TOP em uma instrução INSERT

A cláusula **TOP** em uma instrução **INSERT** define a quantidade ou a porcentagem de linhas que serão inseridas em uma tabela. Isso é muito utilizado para preencher rapidamente tabelas novas com informações existentes.

O exemplo adiante demonstra o uso de **TOP** em uma instrução **INSERT**. Criamos a tabela **CLIENTES_MG**, copiamos 20 registros da tabela **TB_CLIENTE** para a tabela **CLIENTES_MG** e, por fim, exibimos esta última:

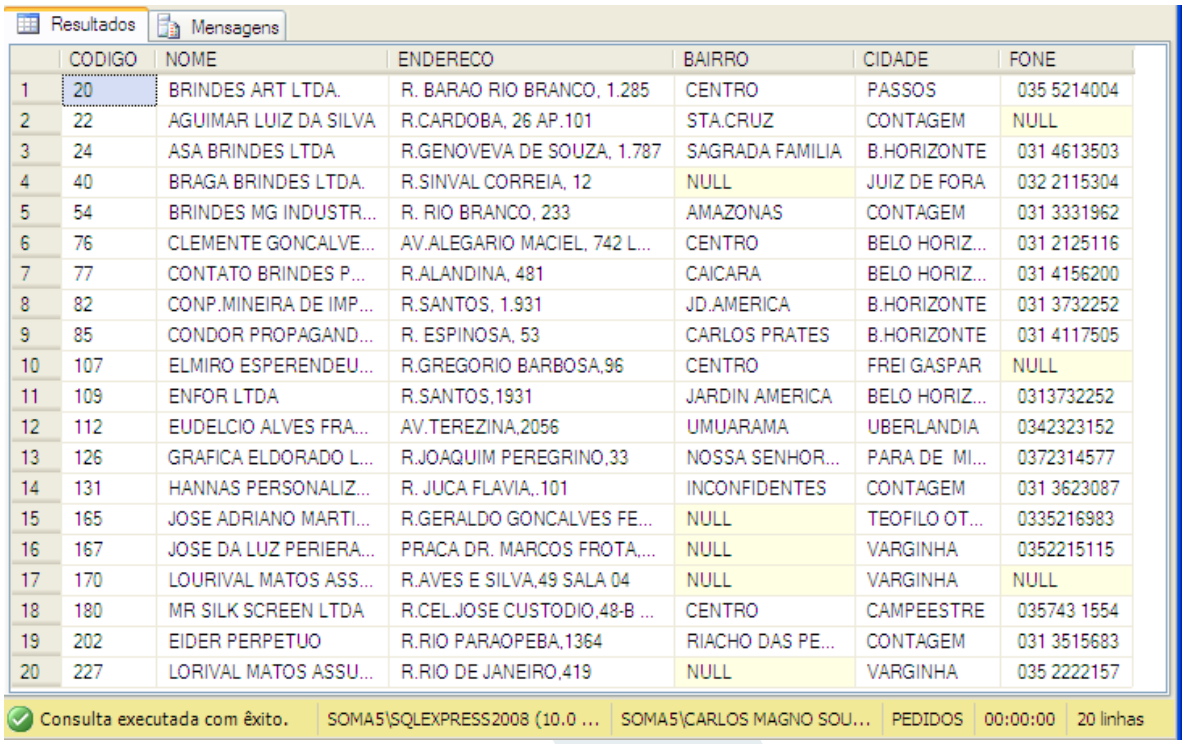
```
-- Colocar o banco de dados PEDIDOS em uso
USE PEDIDOS;

-- Criar a tabela
CREATE TABLE CLIENTES_MG
( CODIGO INT         PRIMARY KEY,
  NOME   VARCHAR(50),
  ENDERECO           VARCHAR(60),
  BAIRRO VARCHAR(30),
  CIDADE VARCHAR(30),
  FONE   VARCHAR(18) )

-- Copiar 20 registros da tabela TB_CLIENTE
-- para a tabela CLIENTES_MG
INSERT TOP( 20 ) INTO CLIENTES_MG
SELECT CODCLI, NOME, ENDERECO, BAIRRO, CIDADE, FONE1
FROM TB_CLIENTE
WHERE ESTADO = 'MG'

-- Consultar CLIENTES_MG
SELECT * FROM CLIENTES_MG
```

O resultado do código anterior é o seguinte:



Resultados | Mensagens

	CODIGO	NOME	ENDERECO	BAIRRO	CIDADE	FONE
1	20	BRINDES ART LTDA.	R. BARAO RIO BRANCO, 1.285	CENTRO	PASSOS	035 5214004
2	22	AGUIMAR LUIZ DA SILVA	R.CARDOBA, 26 AP.101	STA.CRUZ	CONTAGEM	NULL
3	24	ASA BRINDES LTDA	R.GENOVEVA DE SOUZA, 1.787	SAGRADA FAMILIA	B.HORIZONTE	031 4613503
4	40	BRAGA BRINDES LTDA.	R.SINVAL CORREIA, 12	NULL	JUIZ DE FORA	032 2115304
5	54	BRINDES MG INDUSTR...	R. RIO BRANCO, 233	AMAZONAS	CONTAGEM	031 3331962
6	76	CLEMENTE GONCALVE...	AV.ALEGARIO MACIEL, 742 L...	CENTRO	BELO HORIZ...	031 2125116
7	77	CONTATO BRINDES P...	R.ALANDINA, 481	CAICARA	BELO HORIZ...	031 4156200
8	82	CONP.MINEIRA DE IMP...	R.SANTOS, 1.931	JD.AMERICA	B.HORIZONTE	031 3732252
9	85	CONDOR PROPAGAND...	R. ESPINOSA, 53	CARLOS PRATES	B.HORIZONTE	031 4117505
10	107	ELMIRO ESPERENDEU...	R.GREGORIO BARBOSA,96	CENTRO	FREI GASPAR	NULL
11	109	ENFOR LTDA	R.SANTOS,1931	JARDIN AMERICA	BELO HORIZ...	0313732252
12	112	EUDELCIO ALVES FRA...	AV.TEREZINA,2056	UMUARAMA	UBERLANDIA	0342323152
13	126	GRAFICA ELDORADO L...	R.JOAQUIM PEREGRINO,33	NOSSA SENHOR...	PARA DE MI...	0372314577
14	131	HANNAS PERSONALIZ...	R. JUCA FLAVIA,.101	INCONFIDENTES	CONTAGEM	031 3623087
15	165	JOSE ADRIANO MARTI...	R.GERALDO GONCALVES FE...	NULL	TEOFILO OT...	0335216983
16	167	JOSE DA LUZ PERIERA...	PRACA DR. MARCOS FROTA,...	NULL	VARGINHA	0352215115
17	170	LOURIVAL MATOS ASS...	R.AVES E SILVA,49 SALA 04	NULL	VARGINHA	NULL
18	180	MR SILK SCREEN LTDA	R.CEL.JOSE CUSTODIO,48-B ...	CENTRO	CAMPEESTRE	035743 1554
19	202	EIDER PERPETUO	R.RIO PARAOPEBA,1364	RIACHO DAS PE...	CONTAGEM	031 3515683
20	227	LORIVAL MATOS ASSU...	R.RIO DE JANEIRO,419	NULL	VARGINHA	035 2222157

Consulta executada com êxito. | SOMA5\SQLEXPRESS2008 (10.0 ... | SOMA5\CARLOS MAGNO SOU... | PEDIDOS | 00:00:00 | 20 linhas

1.4. OUTPUT

Para verificar se o procedimento executado pelo comando **INSERT**, **DELETE** ou **UPDATE** foi executado corretamente, podemos utilizar a cláusula **OUTPUT** existente nesses comandos. Essa cláusula mostra os dados que o comando afetou.

Usaremos os prefixos **deleted** ou **inserted** para acessar os dados de antes ou depois da operação:

COMANDO	deleted (antes)	inserted (depois)
DELETE	SIM	NÃO
INSERT	NÃO	SIM
UPDATE	SIM	SIM

A cláusula **OUTPUT** é responsável por retornar resultados com base em linhas que tenham sido afetadas por uma instrução **INSERT**, **UPDATE**, **DELETE** ou **MERGE**. Os resultados retornados podem ser usados por um aplicativo como mensagens, bem como podem ser inseridos em uma tabela ou variável de tabela.

A cláusula **OUTPUT** garante que qualquer uma dessas instruções, mesmo que possua erros, retorne linhas ao cliente. Contudo, é importante ressaltar que o resultado não deve ser usado caso ocorra um erro ao executar a instrução.

1.4.1. OUTPUT em uma instrução INSERT

Em uma instrução **INSERT**, a cláusula **OUTPUT** retorna informações das linhas afetadas pela instrução, ou seja, linhas inseridas. Isso pode ser útil para retornar o valor de identidade ou as colunas computadas na instrução. Os resultados retornados também podem ser usados como mensagens de um aplicativo.

A cláusula **OUTPUT** não pode ser utilizada em uma instrução **INSERT** caso o alvo da instrução seja uma tabela remota, expressão de tabela comum ou visualização. Também não pode possuir ou ser referenciada por uma constraint **FOREIGN KEY**.

A seguir, temos um exemplo que demonstra o uso de **OUTPUT** em uma instrução **INSERT**. Primeiramente, vamos criar uma cópia da tabela **TB_EMPREGADO** chamada **EMP_TEMP**:

```
IF OBJECT_ID('EMP_TEMP','U') IS NOT NULL
   DROP TABLE EMP_TEMP;

CREATE TABLE EMP_TEMP
( CODFUN    INT PRIMARY KEY,
  NOME      VARCHAR(30),
  COD_DEPTO INT,
  COD_CARGO    INT,
  SALARIO      NUMERIC(10,2) );
```

O próximo passo é inserir dados na tabela criada e exibir os registros inseridos:

```
INSERT INTO EMP_TEMP OUTPUT INSERTED.*
SELECT CODFUN, NOME, COD_DEPTO, COD_CARGO, SALARIO
FROM TB_EMPREGADO;
GO
```

Agora, excluiremos todos os registros da tabela **EMP_TEMP** e, em seguida, acrescentaremos novos dados e exibiremos algumas colunas:

```
DELETE FROM EMP_TEMP;

INSERT INTO EMP_TEMP
OUTPUT INSERTED.CODFUN, INSERTED.NOME, INSERTED.COD_DEPTO
SELECT CODFUN, NOME, COD_DEPTO, COD_CARGO, SALARIO
FROM TB_EMPREGADO WHERE COD_DEPTO = 2;
GO
```

Depois, declararemos uma variável tabular, acrescentaremos dados e os armazenaremos na variável criada:

```
-- Declarar variável tabular
DECLARE @REG_INSERT TABLE (     CODFUN    INT,
                           NOME       VARCHAR(30),
                           COD_DEPTO INT,
                           COD_CARGO   INT,
                           SALARIO     NUMERIC(10,2) );

-- Inserir dados e armazenar em variável tabular
INSERT INTO EMP_TEMP
OUTPUT INSERTED.* INTO @REG_INSERT
SELECT CODFUN, NOME, COD_DEPTO, COD_CARGO, SALARIO
FROM TB_EMPREGADO WHERE COD_DEPTO = 3;
```

Para exibir os registros inseridos, utilizamos a seguinte instrução:

```
SELECT * FROM @REG_INSERT;
```

Já para exibir todos os registros, a instrução utilizada é a seguinte:

```
SELECT * FROM EMP_TEMP;
GO
```

1.5. Atualizando e excluindo dados

Os comandos da categoria Data Manipulation Language, ou DML, são utilizados não apenas para consultar (**SELECT**) e inserir (**INSERT**) dados, mas também para realizar alterações e exclusões de dados presentes em registros. Os comandos responsáveis por alterações e exclusões são, respectivamente, **UPDATE** e **DELETE**.

Para executarmos diferentes comandos ao mesmo tempo, podemos escrevê-los em um só script. Porém, esse procedimento dificulta a correção de eventuais erros na sintaxe. Portanto, recomenda-se executar cada um dos comandos separadamente.

É importante lembrar que os apóstrofos (‘) devem ser utilizados entre as strings de caracteres, com exceção dos dados numéricos. Os valores de cada coluna, por sua vez, devem ser separados por vírgulas.

1.6. UPDATE

Os dados pertencentes a múltiplas linhas de uma tabela podem ser alterados por meio do comando **UPDATE**.

Quando utilizamos o comando **UPDATE**, é necessário especificar algumas informações, como o nome da tabela que será atualizada e as colunas cujo conteúdo será alterado. Também, devemos incluir uma expressão ou um determinado valor que realizará essa atualização, bem como algumas condições para determinar quais linhas serão editadas.

A sintaxe de **UPDATE** é a seguinte:

```
UPDATE tabela
SET nome_coluna = expressao [, nome_coluna = expressao,...]
 [WHERE condicao]
```

Em que:

- **tabela**: Define a tabela em que dados de uma linha ou grupo de linhas serão alterados;
- **nome_coluna**: Trata-se do nome da coluna a ser alterada;
- **expressao**: Trata-se da expressão cujo resultado será gravado em **nome_coluna**. Também pode ser uma constante;
- **condicao**: É a condição de filtragem utilizada para definir as linhas de tabela a serem alteradas.

> Esta leitura aborda a utilização simples de **UPDATE** sem as cláusulas **FROM** e **JOIN** que foram omitidas da sintaxe do comando. Em uma leitura posterior, veremos como utilizar esse comando com as cláusulas **FROM/JOIN**.

Nas expressões especificadas com o comando **UPDATE**, costumamos utilizar operadores aritméticos, os quais realizam operações matemáticas, e o operador de atribuição =. A tabela a seguir descreve esses operadores:

Operador	Descrição
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Retorna o resto inteiro de uma divisão
=	Atribuição

Tais operadores podem ser combinados, como descreve a próxima tabela:

Operadores	Descrição
+=	Soma e atribui
-=	Subtrai e atribui
*=	Multiplica e atribui
/=	Divide e atribui
%=	Obtém o resto da divisão e atribui

Veja alguns exemplos da utilização desses operadores:

```
-- Operadores
DECLARE @A INT = 10;
SET @A += 5;   -- O mesmo que SET @A = @A + 5;
PRINT @A;
SET @A -= 2;   -- O mesmo que SET @A = @A - 2;
PRINT @A;
SET @A *= 4;   -- O mesmo que SET @A = @A * 4;
PRINT @A;
SET @A /= 2;   -- O mesmo que SET @A = @A / 2;
PRINT @A;
GO
```

1.6.1. Alterando dados de uma coluna

O exemplo a seguir descreve como alterar dados de uma coluna:

```
-- Alterar dados de uma coluna
USE PEDIDOS;
```

```
-- Aumentar o salário de todos os funcionários em 20%
UPDATE TB_EMPREGADO
SET SALARIO = SALARIO * 1.2;
-- OU
UPDATE TB_EMPREGADO
SET SALARIO *= 1.2;
```

```
-- Somar 2 na quantidade de dependentes do funcionário de
-- código 5
UPDATE TB_EMPREGADO
SET NUM_DEPEND = NUM_DEPEND + 2
WHERE CODFUN = 5;
-- OU
UPDATE TB_EMPREGADO
SET NUM_DEPEND += 2
WHERE CODFUN = 5;
```

1.6.2. Alterando dados de diversas colunas

O exemplo a seguir descreve como alterar dados de várias colunas. Será corrigido o endereço do cliente de código **5**:

```
SELECT * FROM TB_CLIENTE WHERE CODCLI = 5;
```

```
-- Alterar os dados do cliente de código 5
UPDATE TB_CLIENTE
SET ENDERECO = 'AV. PAULISTA, 1009 - 10 AND',
    BAIRRO   = 'CERQUEIRA CESAR',
    CIDADE   = 'SÃO PAULO'
WHERE CODCLI = 5;
```

```
-- Conferir o resultado da alteração
SELECT * FROM TB_CLIENTE WHERE CODCLI = 5;
```

No exemplo a seguir, serão corrigidos dados de um grupo de produtos:

```
SELECT * FROM TB_PRODUTO
WHERE COD_TIPO = 5;
```

```
-- Alterar os dados do grupo de produtos
UPDATE TB_PRODUTO SET QTD_ESTIMADA = QTD_REAL,
                      CLAS_FISC = '96082000',
                      IPI = 8
WHERE COD_TIPO = 5;
```

```
-- A linha a seguir confere o resultado da alteração
SELECT * FROM TB_PRODUTO
WHERE COD_TIPO = 5;
```

1.6.3. Utilizando TOP em uma instrução UPDATE

Utilizada em uma instrução **UPDATE**, a cláusula **TOP** define uma quantidade ou porcentagem de linhas que serão atualizadas, conforme o exemplo a seguir, o qual multiplica por 10 o valor de salário de 15 registros da tabela **EMP_TEMP**:

```
-- Consultar
SELECT * FROM EMP_TEMP;
-- Multiplicar por 10 o valor do SALARIO de 15 registros da tabela
UPDATE TOP(15) EMP_TEMP SET SALARIO = 10*SALARIO;
-- Consultar
SELECT * FROM EMP_TEMP;
```

1.7. DELETE

O comando **DELETE** deve ser utilizado quando desejamos excluir os dados de uma tabela. Sua sintaxe é a seguinte:

```
DELETE [FROM] tabela
 [WHERE condicao]
```

Em que:

- **tabela**: É a tabela cuja linha ou grupo de linhas será excluído;
- **condicao**: É a condição de filtragem utilizada para definir as linhas de tabela a serem excluídas.

Uma alternativa ao comando **DELETE**, sem especificação da cláusula **WHERE**, é o uso de **TRUNCATE TABLE**, que também exclui todas as linhas de uma tabela. No entanto, este último apresenta as seguintes vantagens:

- Não realiza o log da exclusão de cada uma das linhas, o que acaba por consumir pouco espaço no log de transações;
- A tabela não fica com páginas de dados vazias;
- Os valores originais da tabela, quando ela foi criada, são restabelecidos, caso haja uma coluna de identidade.

A sintaxe dessa instrução é a seguinte:

```
TRUNCATE TABLE <nome_tabela>
```

1.7.1. Excluindo todas as linhas de uma tabela

Podemos excluir todas as linhas de uma tabela com o comando **DELETE**. Nesse caso, não utilizamos a cláusula **WHERE**.

1. Primeiramente, gere uma cópia da tabela **TB_EMPREGADO**:

```
SELECT * INTO EMPREGADOS_TMP FROM TB_EMPREGADO;
```

2. Agora, exclua os empregados que ganham mais do que **5000**:

```
SELECT * FROM EMPREGADOS_TMP WHERE SALARIO > 5000;
--
DELETE FROM EMPREGADOS_TMP WHERE SALARIO > 5000;
--
```

A linha a seguir verifica se os empregados que ganham mais do que **5000** realmente foram excluídos:

```
SELECT * FROM EMPREGADOS_TMP WHERE SALARIO > 5000;
```

3. A seguir, exclua os empregados de código **3**, **5** e **7**:

```
SELECT * FROM EMPREGADOS_TMP WHERE CODFUN IN (3,5,7);
--
DELETE FROM EMPREGADOS_TMP WHERE CODFUN IN (3,5,7);
```

A linha a seguir verifica se os empregados dos referidos códigos realmente foram excluídos:

```
SELECT * FROM EMPREGADOS_TMP WHERE CODFUN IN (3,5,7);
```

4. Já com o código adiante, elimine todos os registros da tabela **EMPREGADOS_TMP**:

```
DELETE FROM EMPREGADOS_TMP;
-- OU
TRUNCATE TABLE EMPREGADOS_TMP;
--
```

A linha a seguir verifica se todos os registros da referida tabela foram eliminados:

```
SELECT * FROM EMPREGADOS_TMP;
```

1.7.2. Utilizando TOP em uma instrução DELETE

Em uma instrução **DELETE**, a cláusula **TOP** define uma quantidade ou porcentagem de linhas a serem removidas de uma tabela, como demonstrado no exemplo adiante, que exclui 10 linhas da tabela **CLIENTES_MG**:

```
-- Colocar o banco de dados PEDIDOS em uso
USE PEDIDOS;
-- Criar a tabela a partir do comando SELECT INTO
SELECT * INTO CLIENTE_MG FROM TB_CLIENTE;
-- Consultar CLIENTE_MG
SELECT * FROM CLIENTE_MG;

DELETE TOP(10) FROM CLIENTE_MG;
-- Consultar
SELECT * FROM CLIENTE_MG;
```

O resultado do código anterior é o seguinte:

Resultados | Mensagens

	CODIGO	NOME	ENDERECO	BAIRRO	CIDADE	FONE
1	109	ENFOR LTDA	R.SANTOS,1931	JARDIN AMERICA	BELO HORIZ...	0313732252
2	112	EUDELCIO ALVES FRANCO	AV.TEREZINA,2056	UMUARAMA	UBERLANDIA	0342323152
3	126	GRAFICA ELDORADO LTDA	R.JOAQUIM PEREGRINO,33	NOSSA SENHOR...	PARA DE MI...	0372314577
4	131	HANNAS PERSONALIZACAO	R. JUCA FLAVIA,.101	INCONFIDENTES	CONTAGEM	031 3623087
5	165	JOSE ADRIANO MARTINS MA...	R.GERALDO GONCALVES FERREIRA,31	NULL	TEOFILO OT...	0335216983
6	167	JOSE DA LUZ PERIERA ME	PRACA DR. MARCOS FROTA,220	NULL	VARGINHA	0352215115
7	170	LOURIVAL MATOS ASSUNCAO	R.AVES E SILVA,49 SALA 04	NULL	VARGINHA	NULL
8	180	MR SILK SCREEN LTDA	R.CEL.JOSE CUSTODIO,48-B CX.POSTAL 59	CENTRO	CAMPEESTRE	035743 1554
9	202	EIDER PERPETUO	R.RIO PARAOPEBA,1364	RIACHO DAS PED...	CONTAGEM	031 3515683
10	227	LORIVAL MATOS ASSUNCAO	R.RIO DE JANEIRO,419	NULL	VARGINHA	035 2222157

Consulta executada com êxito. | SOMA5\SQLEXPRESS2008 (10.0 ... | SOMA5\CARLOS MAGNO SOU... | PEDIDOS | 00:00:00 | 10 linhas

1.8. OUTPUT para DELETE e UPDATE

A sintaxe é a seguinte:

```
DELETE [FROM] <tabela> [OUTPUT deleted.<nomeCampo>|*[,...]]
WHERE <condição>
```

```
UPDATE <tabela> SET <campo1> = <expr1> [,...]
[OUTPUT deleted|inserted.<nomeCampo> [,...]]
[WHERE <condição>]
```

Veja alguns exemplos de **OUTPUT**:

```
-- Colocar o banco PEDIDOS em uso
USE PEDIDOS;

-- Gerar uma cópia da tabela TB_EMPREGADO chamada EMP_TEMP
CREATE TABLE EMP_TEMP
( CODFUN    INT PRIMARY KEY,
  NOME      VARCHAR(30),
  COD_DEPTO INT,
  COD_CARGO    INT,
  SALARIO      NUMERIC(10,2) )
GO
```

```
-- Inserir dados e exibir os registros inseridos
INSERT INTO EMP_TEMP OUTPUT INSERTED.*
SELECT CODFUN, NOME, COD_DEPTO, COD_CARGO, SALARIO
FROM TB_EMPREGADO;
GO
```

```
-- Deletar registros e mostrar os registros deletados
DELETE FROM EMP_TEMP OUTPUT DELETED.*
WHERE COD_DEPTO = 2
GO
```

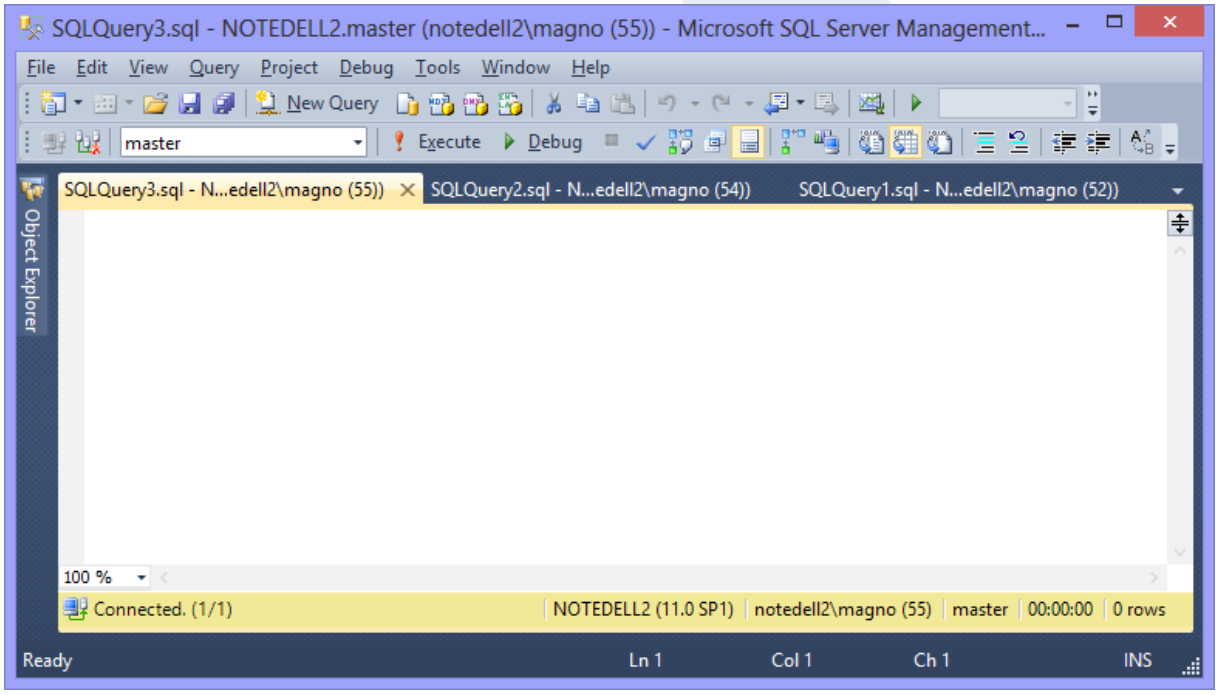
```
-- Alterar registros e mostrar os dados antes e depois da
-- alteração
UPDATE EMP_TEMP SET SALARIO *= 1.5
OUTPUT
   INSERTED.CODFUN, INSERTED.NOME, INSERTED.COD_DEPTO,
   DELETED.SALARIO AS SALARIO_ANTIGO,
   INSERTED.SALARIO AS SALARIO_NOVO
WHERE COD_DEPTO = 3;
GO
```

Observe que podemos conferir se a alteração foi feita corretamente porque é possível verificar se a condição está correta (**COD_DEPTO = 3**), e verificar o campo **SALARIO** antes e depois da alteração.

1.9. Transações

Quando uma conexão ocorre no MS-SQL, ela recebe um número de sessão. Mesmo que a conexão ocorra a partir da mesma máquina e mesmo login, cada conexão é identificada por um número único de sessão.

Observe a figura a seguir, em que temos três conexões identificadas pelas sessões **52**, **54** e **55**:



Sobre as transações, é importante considerar as seguintes informações:

- Um processo de transação é aberto por uma sessão e deve também ser fechado pela mesma sessão;
- Durante o processo, as alterações feitas no banco de dados poderão ser efetivadas ou revertidas quando a transação for finalizada;
- Os comandos **DELETE**, **INSERT** e **UPDATE** abrem uma transação de forma automática. Se o comando não provocar erro, ele confirma as alterações no final, caso contrário, descarta todas as alterações.

1.9.1. Transações explícitas

As transações explícitas são aquelas em que seu início e seu término são determinados de forma explícita. Para definir este tipo de transação, os scripts Transact-SQL utilizam os seguintes comandos:

- **BEGIN TRANSACTION** ou **BEGIN TRAN**

Inicia um processo de transação para a sessão atual.

- **COMMIT TRANSACTION**, **COMMIT WORK** ou simplesmente **COMMIT**

Finaliza o processo de transação atual, confirmando todas as alterações efetuadas desde o início do processo.

- **ROLLBACK TRANSACTION**, **ROLLBACK WORK** ou **ROLLBACK**

Finaliza o processo de transação atual, descartando todas as alterações efetuadas desde o início do processo.

Sobre as transações explícitas, é importante considerar as seguintes informações:

- Se uma conexão for fechada com uma transação aberta, um **ROLLBACK** será executado;
- As operações feitas por um processo de transação ficam armazenadas em um arquivo de log (**.ldf**), que todo banco de dados possui;
- Durante um processo de transação, as linhas das tabelas que foram alteradas ficam bloqueadas para outras sessões.

Veja exemplos de transação:

```
-- Alterar os salários dos funcionários com COD_CARGO = 5
-- para R$950,00
-- Abrir processo de transação
BEGIN TRANSACTION;
-- Verificar se existe processo de transação aberto
SELECT @@TRANCOUNT;
-- Alterar os salários do COD_CARGO = 5
UPDATE TB_EMPREGADO SET SALARIO = 950
OUTPUT inserted.CODFUN, inserted.NOME ,
deleted.salario as Salario_Anterior,
inserted.salario as Salario_Atualizado
WHERE COD_CARGO = 5
-- Conferir os resultados na listagem gerada pela cláusula
-- OUTPUT
-- Se estiver tudo OK...
COMMIT TRANSACTION
-- caso contrário
ROLLBACK TRANSACTION
```

Pontos principais

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

- Para acrescentar novas linhas de dados em uma tabela, utilizamos o comando **INSERT**;
- No **INSERT** posicional não é necessário mencionar o nome dos campos, porém será obrigatório incluir todos os campos, exceto o campo autonumerável. Já no **INSERT** declarativo, os campos deverão ser informados no comando;
- Podemos restringir a quantidade de inserções do comando por meio da cláusula **TOP**;
- Para auditar e mostrar quais dados foram inseridos, utilizamos a cláusula **OUTPUT**;
- Os dados pertencentes a múltiplas linhas de uma tabela podem ser alterados por meio do comando **UPDATE**;
- O comando **DELETE** deve ser utilizado quando desejamos excluir os dados de uma tabela;
- Transações são unidades de programação capazes de manter a consistência e a integridade dos dados. Devemos considerar uma transação como uma coleção de operações que executa uma função lógica única em uma aplicação de banco de dados;
- Todas as alterações de dados realizadas durante a transação são submetidas e tornam-se parte permanente do banco de dados caso a transação seja executada com êxito. No entanto, caso a transação não seja finalizada com êxito por conta de erros, são excluídas quaisquer alterações feitas sobre os dados.