10

Mãos na massa: Auto-incremento, padrões e Triggers

Chegou a hora de você pôr em prática o que foi visto na aula. Para isso, execute os passos listados abaixo.

1) Crie uma tabela com o campo auto-incremento, conforme mostrado abaixo:

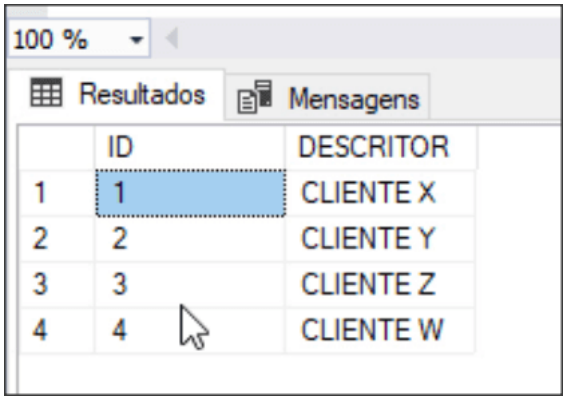
```
CREATE TABLE TAB_IDENTITY (
    ID INT IDENTITY (1,1) NOT NULL,
    DESCRITOR VARCHAR(20) NULL
)
```

2) Insira alguns registros, note que nos comandos de `INSERT` abaixo, não foi especificado o valor para o campo de auto-incremento:

```
INSERT INTO TAB_IDENTITY (DESCRITOR) VALUES ('CLIENTE X')
INSERT INTO TAB_IDENTITY (DESCRITOR) VALUES ('CLIENTE Y')
INSERT INTO TAB_IDENTITY (DESCRITOR) VALUES ('CLIENTE Z')
INSERT INTO TAB_IDENTITY (DESCRITOR) VALUES ('CLIENTE W')
INSERT INTO TAB_IDENTITY (DESCRITOR) VALUES ('CLIENTE A')
```

3) Se você selecionar dados da tabela, verá o valor do campo auto-incremento que foi incluído automaticamente pelo SQL Server:

```
SELECT * FROM TAB_IDENTITY
```



4) Exclua um dos registros da tabela acima:

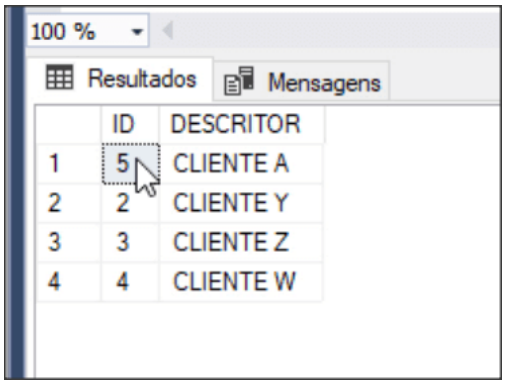
```
DELETE FROM TAB_IDENTITY WHERE ID = 1
```

5) Se você incluir um novo registro:

```
INSERT INTO TAB_IDENTITY (DESCRITOR) VALUES ('CLIENTE A')
```

6) O auto-incremento continuará a crescer seu valor:

```
SELECT * FROM TAB_IDENTITY
```



7) Você pode criar um campo de auto-incremento com valor inicial e intervalo de crescimento diferente de `(1,1)`. Para isso, elimine a tabela:

```
DROP TABLE TAB_IDENTITY
```

8) Crie novamente a tabela, mas agora o campo auto-incremento com as propriedades `(100,5)`:

```
CREATE TABLE TAB_IDENTITY (
    ID INT IDENTITY (100,5) NOT NULL,
    DESCRITOR VARCHAR(20) NULL
)
```

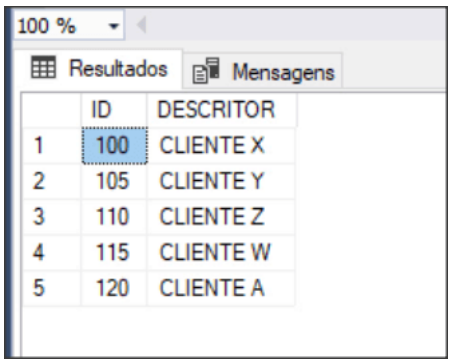
9) Inclua novamente os registros:

```
INSERT INTO TAB_IDENTITY (DESCRITOR)
    VALUES ('CLIENTE X')
INSERT INTO TAB_IDENTITY (DESCRITOR)
    VALUES ('CLIENTE Y')
INSERT INTO TAB_IDENTITY (DESCRITOR)
    VALUES ('CLIENTE Z')
INSERT INTO TAB_IDENTITY (DESCRITOR)
    VALUES ('CLIENTE W')
INSERT INTO TAB_IDENTITY (DESCRITOR)
    VALUES ('CLIENTE A')
```

10) Selecionando os dados:

```
SELECT * FROM TAB_IDENTITY
```

Você terá:



	ID	DESCRITOR
1	100	CLIENTE X
2	105	CLIENTE Y
3	110	CLIENTE Z
4	115	CLIENTE W
5	120	CLIENTE A

11) Crie uma nova consulta no Management Studio e digite os comandos abaixo para criar uma nova tabela. Execute o comando:

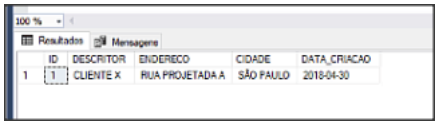
```
CREATE TABLE TAB_PADRAO (
    ID INT IDENTITY (1,1) NOT NULL,
    DESCRITOR VARCHAR(20) NULL,
    ENDERECO VARCHAR(200) NULL,
    CIDADE VARCHAR(50) DEFAULT 'Cidade não definida',
    DATA_CRIACAO DATE DEFAULT GETDATE()
)
```

12) Agora digite um novo comando de inclusão de registros:

```
INSERT INTO TAB_PADRAO (DESCRITOR, ENDERECO, CIDADE, DATA_CRIACAO)
    VALUES ('CLIENTE X', 'RUA PROJETADA A', 'SÃO PAULO', '2018-04-30')
```

13) Fazendo a seleção dos registros, você terá:

```
SELECT * FROM TAB_PADRAO
```

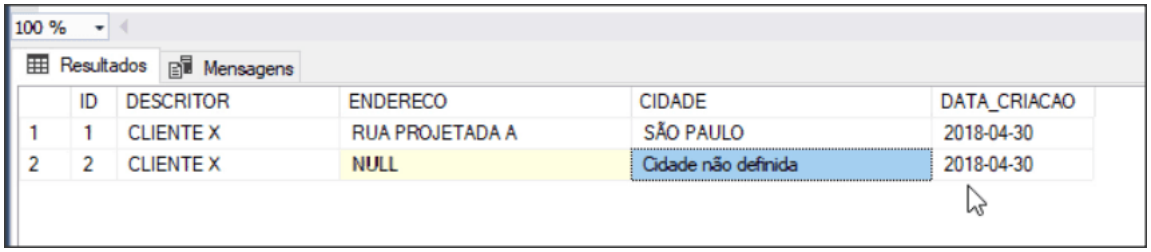


	ID	DESCRITOR	ENDERECO	CIDADE	DATA_CRIACAO
1	1	CLIENTE X	RUA PROJETADA A	SÃO PAULO	2018-04-30

14) Agora, inclua novo registro, omitindo uma série de valores:

```
INSERT INTO TAB_PADRAO (DESCRITOR) VALUES ('CLIENTE X')
```

15) Se você observar a tabela, verá uma série de valores nulos para os campos omitidos, exceto no campo `CIDADE`, que possui um valor padrão definido na criação da tabela:



	ID	DESCRITOR	ENDERECO	CIDADE	DATA_CRIACAO
1	1	CLIENTE X	RUA PROJETADA A	SÃO PAULO	2018-04-30
2	2	CLIENTE X	NULL	Cidade não definida	2018-04-30

16) Crie uma nova consulta no Management Studio e digite os comandos abaixo para criar uma nova tabela::

```
CREATE TABLE TAB_FATURAMENTO (
    DATA_VENDA DATE NULL,
    TOTAL_VENDA FLOAT
)
```

17) Agora, crie uma `TRIGGER` que irá apagar todo o conteúdo da tabela criada anteriormente e calcular o faturamento total de todas as notas fiscais por dia:

```
CREATE TRIGGER TG_ITENS_VENDIDOS
ON [ITENS VENDIDOS]
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
DELETE FROM TAB_FATURAMENTO;
INSERT INTO TAB_FATURAMENTO (DATA_VENDA, TOTAL_VENDA)
SELECT A.DATA AS DATA_VENDA,
    SUM(B.QUANTIDADE * B.[PREÇO]) AS TOTAL_VENDA
FROM NOTAS A INNER JOIN [ITENS VENDIDOS] B
ON A.NÚMERO = B.NÚMERO
GROUP BY A.DATA;
END;
```

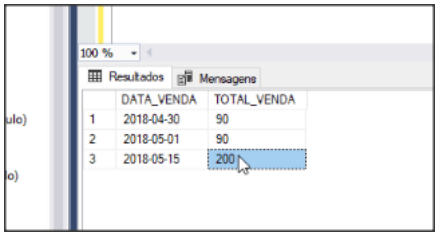
18) insira uma nota fiscal e alguns itens:

```
INSERT INTO [NOTAS] ([NÚMERO], DATA, CPF, MATRICULA, IMPOSTO)
    VALUES ('0100', '2018-05-15', '1471156710', '235', 0.18)
INSERT INTO [ITENS VENDIDOS] ([NÚMERO],[CÓDIGO],[QUANTIDADE],[PREÇO])

    VALUES ('0100', '1000889', 100, 1)
INSERT INTO [ITENS VENDIDOS] ([NÚMERO],[CÓDIGO],[QUANTIDADE],[PREÇO])
    VALUES ('0100', '1002334', 100, 1)
```

19) Após a inclusão desta nota fiscal, verifique a tabela com os faturamentos consolidados:

```
SELECT * FROM TAB_FATURAMENTO
```



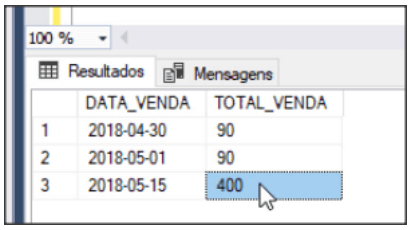
20) Insira uma nova nota fiscal com seus respectivos itens:

```
INSERT INTO [NOTAS] ([NÚMERO], DATA, CPF, MATRICULA, IMPOSTO)
    VALUES ('0101', '2018-05-15', '1471156710', '235', 0.18)
INSERT INTO [ITENS VENDIDOS] ([NÚMERO],[CÓDIGO],[QUANTIDADE],[PREÇO])
```

```
    VALUES ('0101', '1000889', 100, 1)
INSERT INTO [ITENS VENDIDOS] ([NÚMERO],[CÓDIGO],[QUANTIDADE],[PREÇO])
    VALUES ('0101', '1002334', 100, 1)
```

21) Verificando a tabela de faturamentos consolidados, você verá novos valores:

```
SELECT * FROM TAB_FATURAMENTO
```

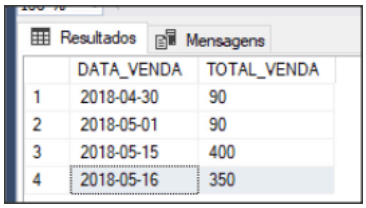


22) Mais uma nota, agora para o dia 16, e você terá novos valores:

```
INSERT INTO [NOTAS] ([NÚMERO], DATA, CPF, MATRICULA, IMPOSTO)
    VALUES ('0102', '2018-05-16', '1471156710', '235', 0.18)
INSERT INTO [ITENS VENDIDOS] ([NÚMERO],[CÓDIGO],[QUANTIDADE],[PREÇO])
    VALUES ('0102', '1000889', 100, 1.5)
INSERT INTO [ITENS VENDIDOS] ([NÚMERO],[CÓDIGO],[QUANTIDADE],[PREÇO])
    VALUES ('0102', '1002334', 200, 1)
```

Depois:

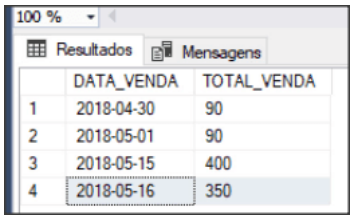
```
SELECT * FROM TAB_FATURAMENTO
```



23) Qualquer alteração na tabela irá mudar a tabela de faturamentos, já que você colocou a propriedade **`AFTER INSERT, UPDATE, DELETE`**. Se você executar o comando de exclusão de um item de nota:

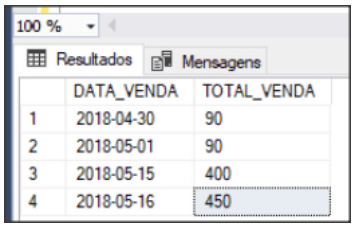
```
DELETE FROM [ITENS VENDIDOS] WHERE [NÚMERO] = '0102' AND [CÓDIGO] = '1002334'
```

Terá novos valores na tabela de faturamento consolidado:



24) Também haverá mudanças ao executar um **`UPDATE`**:

```
UPDATE [ITENS VENDIDOS] SET [QUANTIDADE] = 300
    WHERE [NÚMERO] = '0102' AND [CÓDIGO] = '1000889'
```



	DATA_VENDA	TOTAL_VENDA
1	2018-04-30	90
2	2018-05-01	90
3	2018-05-15	400
4	2018-05-16	450

25) Crie uma nova consulta no Management Studio e digite os comandos abaixo para criar uma nova tabela:

```
CREATE TABLE TAB_CHECK (
    ID INT NOT NULL,
    NOME VARCHAR(20) NULL,
    IDADE INT NULL,
    CIDADE VARCHAR(20) NULL,
    CONSTRAINT CHK_PESSOA CHECK (IDADE >= 18)
)
```

26) Na tabela acima, a idade foi restringida para maior de 18 anos. Se você executar o comando:

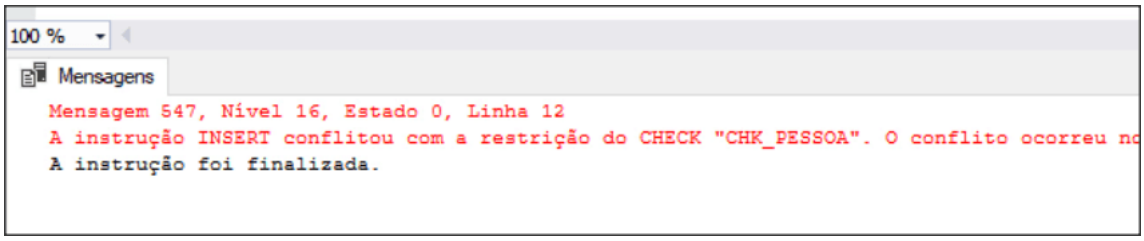
```
INSERT INTO TAB_CHECK (ID, NOME, IDADE, CIDADE)
    VALUES (1, 'JOÃO', 19, 'RIO DE JANEIRO')
```

Ele será incluído na tabela.

27) Mas se você executar outro **INSERT**, conforme mostrado abaixo:

```
INSERT INTO TAB_CHECK (ID, NOME, IDADE, CIDADE)
    VALUES (2,'PEDRO', 17, 'SÃO PAULO')
```

Terá a mensagem:



28) Se corrigir para:

```
INSERT INTO TAB_CHECK (ID, NOME, IDADE, CIDADE)
    VALUES (2,'PEDRO', 20, 'SÃO PAULO')
```

Agora sim a informação é incluída.

29) A condição do `CHECK` pode ser composta com `AND` ou `OR`, por exemplo. Crie outra tabela:

```
CREATE TABLE TAB_CHECK2 (
    ID INT NOT NULL,
    NOME VARCHAR(20) NULL,
    IDADE INT NULL,
    CIDADE VARCHAR(20) NULL,
    CONSTRAINT CHK_PESSOA2
        CHECK (IDADE >= 18 AND CIDADE = 'RIO DE JANEIRO')
)
```

30) Ao incluir um novo registro:

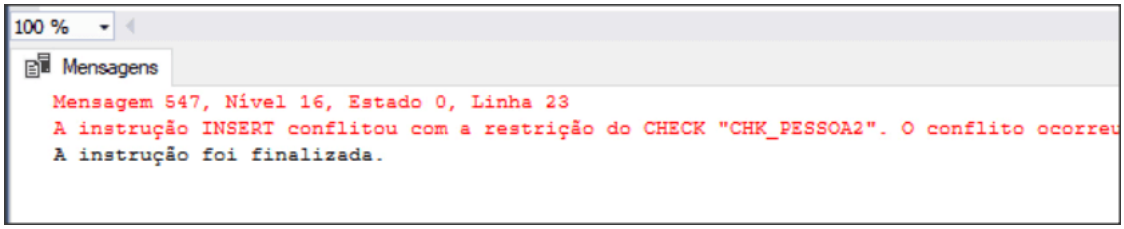
```
INSERT INTO TAB_CHECK2 (ID, NOME, IDADE, CIDADE)
    VALUES (1, 'JOÃO', 19, 'RIO DE JANEIRO')
```

Não haverá problemas.

31) Mas se você digitar:

```
INSERT INTO TAB_CHECK2 (ID, NOME, IDADE, CIDADE)
    VALUES (2, 'PEDRO', 20, 'RIO DE JANEIRO')
```

O problema de integridade será apresentado, já que a idade está correta, mas a cidade não:



32) A restrição é testada não somente durante o `INSERT`, mas também durante o `UPDATE`:

```
UPDATE TAB_CHECK2 SET CIDADE = 'SÃO PAULO' WHERE ID = 2
```