Programação em Banco de Dados

Autoria: Douglas Fugita de Oliveira Cezar

Tema 03

Alteração, Exclusão e o Gerenciamento de Transações

Anhanguera

Tema 03

Alteração, Exclusão e o Gerenciamento de Transações

Autoria: Douglas Fugita de Oliveira Cezar

Como citar esse documento:

CEZAR, Douglas Fugita de Oliveira. *Programação em Banco de Dados*: Alteração, Exclusão e o Gerenciamento de Transações. Caderno de atividades. Valinhos: Anhanguera Educacional, 2014.

Índice

CONVITEÀLEITURA

Neste caderno de atividades você terá o contato com os comandos responsáveis pela manipulação dos dados do banco de dados, seja através da inclusão, exclusão ou alteração dos registros existentes.

Um conceito que será revisto neste caderno será o da atomicidade, uma das características básicas de um banco de dados. Este conceito define que determinadas operações devem ser executadas por completo e, caso a operação seja interrompida no meio do caminho, os dados que já foram alterados devem voltar ao que eram antes do início da execução.

Será abordado também o conceito de permissões de acessos. Você poderá entender como deve ser feita a concessão de permissão ou então a negação da permissão para um usuário em específico.

Com este conjunto de comandos e conceitos, você estará apto a programar os mais diversos bancos de dados.

PORDENTRODOTEMA

Alteração, Exclusão e o Gerenciamento de Transações

Uma linguagem de programação de banco de dados pode ser dividida em três grandes grupos: a linguagem de definição de dados (DDL – *Data Definition Language*), a linguagem de controle de dados (DCL – *Data Control Language*) e a linguagem de manipulação de dados (DML – *Data Manipulation Language*).

A DML contém uma série de comandos responsáveis por inserir, excluir, alterar e consultar os dados que estão/serão armazenados no banco de dados e, neste caderno, daremos continuidade aos comandos de manipulação de dados.

Para este caderno, continuaremos utilizando como base as tabelas exibidas anteriormente nos outros temas. O quadro 3.1 mostrará os dados que populam a tabela Município.

Quadro 3.1– Dados populados na tabela Município

ID_Municipo	Nome_Municipio	UF_Municipio
1	Sao Paulo	SP
2	Rio de Janeiro	RJ
3	Belo Horizonte	MG
4	Curitiba	PR
5	Campinas	SP
6	Angra dos Reis	RJ
7	Niteroi	RJ
8	Salvador	BA
9	Feira de Santana	BA
10	Porto Seguro	BA
11	Guarulhos	SP
12	Cuiaba	MT
13	Barra do Garça	MT
14	Porto Alegre	RS
15	Uruguaiana	RS
16	Caxias do Sul	RS

Para as consultas realizadas anteriormente, foi considerada esta tabela já preenchida com dados. No entanto, existe um comando para que os dados sejam inseridos em uma tabela e este comando é o INSERT.

Para inserir um dado na tabela Municipio, será utilizado o comando mostrado no quadro 3.2, bem como a tabela após a inclusão.

```
INSERT INTO [dbo].[Municipio]([Nome_Municipio], [UF_Municipio])
   VALUES('Palmas', 'RO')
//Erro proposital, para exemplo futuro
```

Quadro 3.2 – Comando INSERT

ID_Municipo	Nome_Municipio	UF_Municipio
1	Sao Paulo	SP
2	Rio de Janeiro	RJ
3	Belo Horizonte	MG
4	Curitiba	PR
5	Campinas	SP
6	Angra dos Reis	RJ
7	Niteroi	RJ
8	Salvador	BA
9	Feira de Santana	BA
10	Porto Seguro	BA
11	Guarulhos	SP
12	Cuiaba	MT
13	Barra do Garça	MT
14	Porto Alegre	RS
15	Uruguaiana	RS
16	Caxias do Sul	RS
17	Palmas	RO

O comando INSERT também pode popular uma tabela baseado em um comando de consulta SELECT. Será criada uma estrutura para armazenar as siglas dos estados e ela será populada com os estados cujos clientes já receberam pelo menos uma nota fiscal, como pode ser visto no quadro 3.3. O resultado da operação também será mostrado no quadro.

```
CREATE TABLE [dbo].[UF_Com_NF] (
   [id_UF] int IDENTITY,
   [UF]    varchar(2) NULL
   )

INSERT INTO UF_com_NF
   SELECT DISTINCT UF_Municipio FROM Municipio m
      INNER JOIN Clientes c
         ON c.ID_Municipio = m.ID_Municipo
      INNER JOIN NF n
         ON n.ID_Cliente = c.ID_Cliente
```

Quadro 3.3 – INSERT baseado em SELECT

id_UF	UF
9	MG
10	RJ
11	SP

A informação criada com o INSERT do quadro 3.2, como identificado no comentário do comando, foi gerada com erro. Para corrigir o erro, será utilizado o comando UPDATE, conforme mostrado no quadro 3.4.

```
UPDATE Municipio SET UF_Municipio = 'TO'
WHERE Nome_Municipio = 'Palmas'
```

Quadro 3.4 – Comando UPDATE

ID_Municipo	Nome_Municipio	UF_Municipio
1	Sao Paulo	SP
2	Rio de Janeiro	RJ
3	Belo Horizonte	MG
4	Curitiba	PR
5	Campinas	SP
6	Angra dos Reis	RJ
7	Niteroi	RJ
8	Salvador	BA
9	Feira de Santana	BA
10	Porto Seguro	BA
11	Guarulhos	SP
12	Cuiaba	MT
13	Barra do Garça	MT
14	Porto Alegre	RS
15	Uruguaiana	RS
16	Caxias do Sul	RS
17	Palmas	TO

PORDENTRODOTEMA

O comando UPDATE irá atualizar todos os registros que forem satisfeitos pela cláusula WHERE. Caso não haja a cláusula WHERE no comando, todos os registros da tabela serão afetados.

Através do comando UPDATE pode ser feita uma atualização no preço de venda dos produtos, utilizando **fórmulas** para definir o valor a ser atualizado, como pode ser visto no quadro 3.5.

Quadro 3.5 – UPDATE com fórmula

Consulta antes do UPDATE

ID_Produto	Nome_Produto	ValorVenda_Produto
1	Lapis	1,49
2	Caneta azul	3,5
3	Caderno 100fl	15
4	Grampo	12,9
5	Borracha	4,2
6	Folha Sulfite	18,9

```
UPDATE Produto
SET ValorVenda_Produto = ValorVenda_Produto * 1.1
```

Consulta após UPDATE

ID_Produto	Nome_Produto	ValorVenda_Produto
1	Lapis	1,64
2	Caneta azul	3,85
3	Caderno 100fl	16,5
4	Grampo	14,19
5	Borracha	4,62
6	Folha Sulfite	20,79

A estrutura CASE também pode ser utilizada para que a atualização possa ser variável. O quadro 3.6 mostra a diminuição de 25% do valor de venda caso seja comercializado em caixas (CX) ou a diminuição de 5% caso seja comercializado em pacotes (PCT). Para as demais, os preços serão aumentados em R$ 0,10.

Quadro 3.6 – UPDATE com estrutura CASE.

Consulta antes do UPDATE

ID_Produto	Nome_Produto	ValorVenda_Produto	UnidadeMedida_Produto
1	Lapis	1,64	UN
2	Caneta azul	3,85	UN
3	Caderno 100fl	16,5	UN
4	Grampo	14,19	CX
5	Borracha	4,62	UN
6	Folha Sulfite	20,79	PCT

```
UPDATE Produto
   SET ValorVenda_Produto = CASE
         WHEN UnidadeMedida_Produto = 'CX'
            THEN ROUND(ValorVenda_Produto*0.75,2)
         WHEN UnidadeMedida_Produto = 'PCT'
            THEN ROUND(ValorVenda_Produto*0.95,2)
         ELSE ValorVenda_Produto + 0.1
         END
```

Consulta após UPDATE

ID_Produto	Nome_Produto	ValorVenda_Produto	UnidadeMedida_Produto
1	Lapis	1,74	UN
2	Caneta azul	3,95	UN
3	Caderno 100fl	16,6	UN
4	Grampo	10,64	CX
5	Borracha	4,72	UN
6	Folha Sulfite	19,75	PCT

Outro comando utilizado para a manipulação de dados é o DELETE, responsável pela exclusão de informações de uma tabela. Assim como o UPDATE, ele deve sempre ser utilizado com uma cláusula WHERE para definir quais registros serão afetados. Caso contrário, todos os registros de uma tabela serão apagados. O exemplo pode ser visto no quadro 3.7.

Quadro 3. 7 – Comando DELETE.

Consulta antes do DELETE

id_UF	UF
9	MG
10	RJ
11	SP

DELETE FROM UF_com_NF

Consulta após DELETE

id_UF	UF

É importante reparar que o comando DELETE somente apaga os dados, mas a estrutura da tabela é mantida. Caso queira afetar também a estrutura da tabela, deve ser utilizado o comando DROP TABLE, já visto anteriormente.

Um das características de um banco de dados é a atomicidade. Este princípio indica que uma operação deve ser realizada por completo e, caso seja interrompida durante a execução, as alterações realizadas anteriormente devem ser descartadas.

O uso do controle de transação está **implícito** nos comandos INSERT, UPDATE ou DELETE quando executados sozinhos, e os registros são alterados somente se o comando for executado com sucesso.

No entanto, existem comandos que possibilitam o controle de transação de forma explícita. São eles: BEGIN TRANSACTION, COMIT TRANSACTION e ROLLBACK TRANSACTION.

O exemplo clássico para este tipo de operação é a transferência eletrônica de dinheiro, na qual é realizada uma operação de saque e outra de depósito, e as duas devem ser realizadas atomicamente, ou seja, uma não pode ser realizada caso a outra não seja. Para exemplificar este contexto, será utilizada a tabela mostrada no quadro 3.8.

Quadro 3.8 – Tabela Conta Bancária e UPDATE da transferência.

Banco	agencia	conta	saldo
5	6658	118779-5	R$ 25.441,00
633	2115	5496085-9	R$ 78.600,00

```
UPDATE ContaBancaria
   SET saldo = saldo - 900
   WHERE banco = '005'
      AND agencia = '6658'
      AND conta = '118779-5'

UPDATE ContaBancaria
   SET saldo = saldo + 900
   WHERE banco = '633'
      AND agencia = '2115'
      AND conta = '5496085-9'
```

No exemplo acima, caso haja alguma falha após a execução do primeiro UPDATE, a primeira conta será diminuída de R$ 900,00, e este valor não será acrescido na outra. O inverso também é valido, caso somente a segunda atualização seja efetuada.

Para evitar este tipo de falha, o comando de transação é utilizado conforme mostra o quadro 3.9

Quadro 3.9 – BEGIN TRANSACTION e COMMIT TRANSACTION

```
BEGIN TRANSACTION Trans01
  UPDATE ContaBancaria
     SET saldo = saldo - 900
     WHERE banco = '005'
        AND agencia = '6658'
        AND conta = '118779-5'

  UPDATE ContaBancaria
     SET saldo = saldo + 900
     WHERE banco = '633'
        AND agencia = '2115'
        AND conta = '5496085-9'
COMMIT TRANSACTION Trans01
```

As alterações solicitadas pelos comandos executados abaixo do BEGIN TRANSACTION são realizadas somente após o comando COMMIT TRANSACTION. Caso algum erro ocorra, o comando de COMMIT não é realizado.

O quadro 3.10 mostra como o banco de dados reage ao ROLLBACK.

Quadro 3.10 – Resultado de um ROLLBACK

```
BEGIN TRANSACTION Trans02
   UPDATE ContaBancaria
   SET saldo = 28000
   WHERE agencia = '6658';

SELECT * from ContaBancaria;
```

banco	agencia	conta	saldo
5	6658	118779-5	R$ 28.000,00
633	2115	5496085-9	R$ 78.600,00

```
ROLLBACK TRANSACTION Trans02;

SELECT * FROM ContaBancaria;
```

banco	agencia	conta	saldo
5	6658	118779-5	R$ 25.441,00
633	2115	5496085-9	R$ 78.600,00

Embora, após o comando UPDATE, o resultado do comando SELECT mostrasse um saldo atualizado, somente um comando de COMMIT oficializaria este dado na **base de dados**. Após a execução do ROLLBACK, o valor anterior é recuperado e a atualização é descartada. A mesma transação pode ser vista, agora com o comando de COMMIT, no quadro 3.11.

Quadro 3.11 – Resultado de um COMMIT.

```
BEGIN TRANSACTION Trans03
   UPDATE ContaBancaria
   SET saldo = 28000
   WHERE agencia = '6658';

SELECT * from ContaBancaria;
```

banco	agencia	conta	saldo
5	6658	118779-5	R$ 28.000,00
633	2115	5496085-9	R$ 78.600,00

```
COMMIT TRANSACTION Trans03;

SELECT * FROM ContaBancaria;
```

banco	agencia	conta	saldo
5	6658	118779-5	R$ 28.000,00
633	2115	5496085-9	R$ 78.600,00

Para automatizar a execução de COMMIT e ROLLBACK, dependendo da execução dos comandos da query, um recurso que pode ser utilizado é a variável global @@ERROR, que mantém seu valor em zero (0) até que um erro aconteça. Com isso, você pode ver no quadro 3.12 uma forma de disparar os comandos de COMMIT ou ROLLBACK automaticamente.

Quadro 3.12 – Automatização de COMMIT e ROLLBACK.

```
BEGIN TRANSACTION Trans01
   UPDATE ContaBancaria
      SET saldo = saldo - 900
      WHERE banco = '005'
         AND agencia = '6658'
         AND conta = '118779-5'

   UPDATE ContaBancaria
      SET saldo = saldo + 900
      WHERE banco = '633'
         AND agencia = '2115'
         AND conta = '5496085-9'
IF @@ERROR <> 0
   ROLLBACK TRANSACTION Trans01
ELSE
   COMMIT TRANSACTION Trans01
```

Para que os comandos vistos neste caderno possam ser executados, assim como SELECT do caderno anterior, os comandos da DCL, linguagem de controle de dados, devem ser utilizados. Estes comandos servem para dar ou negar o privilégio de acesso aos objetos da base de dados.

A autorização deve ser dada por usuário e, para isso, serão utilizados os comandos CREATE LOGIN e CREATE USER, conforme mostra o quadro 3.13.

Quadro 3.13 – CREATE LOGIN e CREATE USER.

```
CREATE LOGIN novo_usuario WITH PASSWORD = 'nova_senha'

USE BD_TADS

CREATE USER novo_usuario FOR LOGIN novo_usuario
```

A permissão é concedida através do comando GRANT e negada pelo comando DENY. Estes comandos devem ser executados levando em consideração o que será permitido/negado e a quem. As opções mais comuns de permissão/negação são os comandos SELECT, INSERT, UPDATE e DELETE. O uso da permissão GRANT pode ser visto no quadro 3.14.

Quadro 3.14 – GRANT – concessão de permissão.

```
GRANT SELECT
   on Clientes, Municipio
  TO novo_usuario

GRANT SELECT, UPDATE
  ON NF,NFItens
  TO novo_usuario

DENY INSERT, DELETE
  ON NF,NFItens, Clientes, Municipio
  TO novo_usuario
```

No exemplo anterior, o novo_usuário recebeu permissão de visualizar o conteúdo das tabelas Clientes e Municipio, bem como visualizar e alterar as tabelas NF e NFItens. Ao mesmo tempo, está sendo negada permissão de executar os comandos INSERT e DELETE para as tabelas citadas anteriormente.

O comando DENY anula qualquer permissão anterior mesmo que haja um comando GRANT dando permissão ao usuário executar alguma operação em determinada tabela.

Os comandos de permissão podem ser especificados para que somente campos específicos sejam afetados pela permissão. Um exemplo pode ser visto no quadro 3.15.

Quadro 3.15 – GRANT com campo específico.

```
GRANT UPDATE(Desconto_NFItens)
  ON NFItens
  TO novo_usuario
```

PORDENTRODOTEMA

Este argumento de utilizar com a permissão UPDATE indica que somente o campo Desconto_NFItens pode ser alterado pelo usuário novo_usuario. Esta mesma estratégia pode ser utilizada pela permissão INSERT.

Uma permissão pode ser removida através do comando REVOKE e sua sintaxe é semelhante à concessão/negação de permissão. Um exemplo é mostrado quadro 3.16.

Quadro 3.16 – REVOKE.

```
REVOKE SELECT
  ON Clientes
  FROM novo_usuario
```

Diferentemente do DENY, o REVOKE não nega uma permissão. Este comando somente retira a permissão concedida/negada, deixando assim as permissões como eram anteriormente à operação de GRANT ou DENY. Caso a tabela Clientes tenha permissão pública negada como opção padrão, o usuário novo_usuario se enquadrará na mesma regra, visto que sua permissão foi retirada.

ACOMPANHENAWEB

Instruções de transação (Transact-SQL)

- Instruções detalhadas sobre transação no MS SQL Server. Este site contém, detalhadamente, todas as possibilidades de utilização do recurso de transações. Vale a pena aprofundar-se neste assunto para que seu uso seja efetivo.

Disponível em: <http://msdn.microsoft.com/pt-br/library/ms174377.aspx>. Acesso em: 15 abr. 2014.

AGORAÉASUAVEZ

Instruções:

Agora, chegou a sua vez de exercitar seu aprendizado. A seguir, você encontrará algumas questões de múltipla escolha e dissertativas. Leia cuidadosamente os enunciados e atente-se para o que está sendo pedido.

Questão 1

Defina o conceito de atomicidade. Dê um exemplo que ilustre este conceito.

Questão 2

Os comandos executados após o BEGIN TRANSACTION são confirmados somente após um comando:

a) END TRANSACTION

b) ROLLBACK TRANSACTION

c) CASE TRANSACTION

d) COMMIT TRANSACTION

e) UPDATE

AGORA É A SUA VEZ

Questão 3

O campo saldo possui o valor inicial de R$ 5.000,00, e as operações a seguir serão realizadas após a declaração de início de uma transação. O campo saldo é alterado, sendo acrescido de R$ 800 e, após, é realizada uma operação de ROLLBACK. Qual o valor no campo saldo antes e depois do ROLLBACK?

a) R$ 5.000,00 / R$ 5.000,00

b) R$ 5.800,00 / R$ 5.000,00

c) R$ 5.000,00 / R$ 5.800,00

d) R$ 5.800,00 / R$ 5.800,00

Questão 4

Utilizando como base o enunciado da questão 3, crie um query que satisfaça a descrição.

Questão 5

Utilizando como base a tabela Produtos mostrada neste caderno, crie uma atualização, na qual seja possível reajustar o preço dos produtos vendidos em unidades (UN) em 10%. Os outros produtos devem ser reajustados somente em 5%. Esta operação deverá utilizar os comandos de transação, e o desfecho da transação deverá ser automatizado, utilizando a variável @@ERROR.

FINALIZANDO

Neste caderno você conheceu os demais comandos pertencentes à Linguagem de Manipulação de Dados (DML), além de comando de transação e também os comandos da Linguagem de Controle de Dados (DCL).

Através destes comandos, em conjunto com os vistos nos módulos anteriores, você tem a capacidade de programar grandes e complexos bancos de dados.

REFERÊNCIAS

SILBERSCHATZ, Abrahan; KORTH, Henry; SUDARSHAN, S. *Sistema de Banco de Dados*. Elsevier, 2012.

MICROSOFT, *Instruções de Transação (Transact STL)*. Disponível em: <http://msdn.microsoft.com/pt-br/library/ms174377.aspx>. Acesso em: 15 abr. 2014.

GLOSSÁRIO

Fórmulas: operações que resultam em um novo valor.

Implícito: de forma escondida, que não precisa ser definido pelo programador.

Base de dados: o mesmo que banco de dados.

GABARITO

Questão 1

Resposta: Este conceito define que determinadas operações devem ser executadas por completo e, caso a operação seja interrompida no meio do caminho, os dados que já foram alterados devem voltar ao que eram, antes do início da execução.

Qualquer exemplo que retrate operações que necessitam ocorrer simultaneamente é valido, como por exemplo, a geração de um título no financeiro após a emissão de uma nota fiscal, a baixa no estoque após uma venda, etc.

GABARITO

Questão 2

Resposta: Alternativa D.

Questão 3

Resposta: Alternativa B.

Questão 4

Resposta:

```
BEGIN TRANSACTION Trans03
   UPDATE ContaBancaria
   SET saldo = saldo + 800

SELECT * from ContaBancaria;

ROLLBACK TRANSACTION Trans03;

SELECT * FROM ContaBancaria;
```

Questão 5

Resposta:

```
BEGIN TRANSACTION Trans04
UPDATE Produto
   SET ValorVenda_Produto = CASE
         WHEN UnidadeMedida_Produto = 'UN'
            THEN ROUND(ValorVenda_Produto*1.10,2)
         ELSE ValorVenda_Produto * 1.05
         END
IF @@ERROR <> 0
   ROLLBACK TRANSACTION Trans04
ELSE
   COMMIT TRANSACTION Trans04
```


Anhanguera