CAPÍTULO 8

TRANSAÇÕES EM BANCO DE DADOS

Olá!

Lembra quando, no início deste material, argumentamos que a tecnologia de banco de dados veio a substituir os Sistemas de Arquivos Tradicionais, suprindo as deficiências destes últimos?

Uma dessa deficiências era a impossibilidade de assegurar a integridade dos dados armazenados.

Neste capítulo, veremos uma das mais interessantes estratégias dos sistemas gerenciadores de bancos de dados para assegurar a integridade dos dados. Essa estratégia atende pelo nome de transação.

Prof. Vanderson José Ildefonso Silva

Fala Professor

8.1 O conceito de transação

Como comentado anteriormente – em um dos capítulos iniciais – os sistemas de informação estão sujeitos a paradas e falhas de variadas causas: defeito de hardware, interrupção do fornecimento de energia, erros de software, falha de comunicação entre dispositivos distribuídos, incêndio nas instalações, sabotagem, etc.

Em termos da tecnologia de banco de dados é fundamental assegurar a consistência dos dados; mesmo que algumas dessas falhas ocorram. Suponha que o registro de uma VENDA tenha sido efetuado em nosso banco de dados. A empresa servida por essa tecnologia vendeu uma unidade de notebook. Como existiam 20 unidades antes da VENDA, após a mesma deverão existir no ESTOQUE apenas 19 unidades de notebooks.

Se após a VENDA acima constar nos registros qualquer outra quantidade de notebooks no ESTOQUE, podemos afirmar que o banco de dados não mais se encontra em situação consistente. A integridade dos dados não foi assegurada. Como isso poderia ter acontecido?

Imagine que logo após o registro da VENDA, mas imediatamente antes da atualização do ESTOQUE, ocorra uma interrupção no suprimento de

energia. Com isso, os dados que ainda estavam na memória principal ou nos registradores foram perdidos. A VENDA já havia sido devidamente armazenada na memória secundária, mas a mudança no ESTOQUE ainda não. Em consequência temos uma situação de inconsistência dos dados. O banco de dados não mais reflete a realidade.

Conceitos

Uma **transação** pode ser definida como um conjunto de comandos de programação que, após sua execução, ainda preserva a consistência do banco de dados.

Caso o banco de dados estivesse consistente antes da execução da **transação**, podemos estar certos de que permanecerá consistente após sua execução.

Conforme KORTH, SILBERSCHATZ e SUDARSHAN (2006), toda **transação** é **atômica**. Significa que ou todas as instruções de uma **transação** são executadas ou então nenhuma delas. A possibilidade de apenas parte das instruções associadas à **transação** ser executada é simplesmente inexistente. É uma questão de "**tudo ou nada**".

Sempre que um programa aplicativo acessar e atualizar diversos registros de uma ou mais tabelas, seu programador deverá definilo como uma **transação**. Essa é uma responsabilidade do **programador de aplicativos**. O simples esquecimento nesse caso pode implicar um grave risco para a integridade de um sistema.

Quando uma transação completa sua execução com sucesso, dizemos que a mesma foi **compromissada**. Por outro lado, quando não consegue concluir sua execução com sucesso, a **transação** assume a condição de **abortada**. A regra da atomicidade exige que todas as ações perpetradas por uma **transação abortada** devem ser **desfeitas**, de maneira que essa transação não tenha efeito sobre o estado do **banco de dados**.

Para assegurar a consistência dos dados, transações costumam implementar um histórico incremental com atualizações adiadas. Considere nosso exemplo no início do capítulo, onde ocorria a venda de um *notebook*. Suponha que antes da venda ser efetuada, a situação no banco de dados fosse a seguinte:

PRODUTO	
DESCRIÇÃO	**ESTOQUE**
Notebook	20
TV Plasma	13

LOG DO SISTEMA

VENDA		
NR VENDA	**DESCRICAO**	**QUANTIDADE**

Figura 44: Situação Anterior à Transação.

O cliente chega ao caixa para pagar o Notebook e a transação é iniciada: um registro é inserido no arquivo de LOG_DO_SISTEMA para sinalizar que a transação **T1** foi inicializada ("**<T1, INÍCIO>**").

PRODUTO	
DESCRIÇÃO	**ESTOQUE**
Notebook	20
TV Plasma	13

LOG DO SISTEMA
<T1, INICIO>

VENDA		
NR VENDA	**DESCRICAO**	**QUANTIDADE**

Figura 45: Início da Transação T1.

A transação **T1** foi iniciada, mas ainda não foi concluída. Então o caixa registra a venda do ***Notebook*** (um registro deve ser gravado na tabela VENDA). Porém, antes que isto aconteça, um novo registro é inserido no arquivo de LOG_DO_SISTEMA, informando a inserção na tabela de VENDA.

Note o formato do registro no arquivo de LOG_DO_SISTEMA ("**<T1, Venda, Notebook, 1>**"). Ele indica que a transação T1 inseriu um registro de venda de 1 unidade de Notebook.

Suponha que uma falha grave ocorra neste exato momento e o banco de dados tenha de ser reiniciado. A transação ainda não foi compromissada e o banco de dados tem "consciência" de sua condição, pois há um registro **<T1, INICIO>** no arquivo LOG_DO_SISTEMA, mas não um registro **<T1, FIM>**. Portanto, a transação não foi concluída – não foi compromissada.

Então, o banco de dados aborta a transação T1. Contudo, nenhuma transação deve ser apenas parcialmente executada – como determina a regra do "tudo ou nada". Assim, o banco de dados trata de cancelar todas as operações executadas por T1. No arquivo LOG_DO_SISTEMA encontra uma referência à inclusão de um registro na tabela VENDA. Contudo, como não encontra esse registro na tabela, não precisa excluí-lo.

PRODUTO	
DESCRIÇÃO	**ESTOQUE**
Notebook	20
TV Plasma	13

LOG DO SISTEMA
<T1, INICIO>
<T1, Venda, Notebook, 1>

VENDA		
NR VENDA	**DESCRICAO**	**QUANTIDADE**

Figura 46: Transação em andamento: Log do Sistema Atualizado.

Porém, vamos imaginar que nenhuma falha ocorreu até aqui. O próximo passo da transação T1 é incluir o registro na tabela VENDA.

PRODUTO	
DESCRIÇÃO	**ESTOQUE**
Notebook	20
TV Plasma	13

LOG DO SISTEMA
<T1, INICIO>
<T1, Venda, Notebook, 1>

VENDA		
NR VENDA	**DESCRICAO**	**QUANTIDADE**
1	Notebook	1

Figura 47: Transação em andamento: Atualização da Tabela Venda.

Se alguma falha ocorrer nesse exato momento, o banco de dados perceberá que a transação T1 não foi concluída e tratará de excluir o registro inserido em VENDA.

Por outro lado, se nenhuma falha ocorrer até aqui, o arquivo LOG_DO_SISTEMA receberá um novo registro ("**<T1, Produto, Notebook, 20, 19>**"). Este sinaliza que o registro da tabela PRODUTO relativo ao Notebook será alterado de tal forma que a coluna ESTOQUE – que tinha valor 20 – passará a ter valor 19.

PRODUTO	
DESCRIÇÃO	**ESTOQUE**
Notebook	20
TV Plasma	13

LOG DO SISTEMA
<T1, INICIO>
<T1, Venda, Notebook, 1>
<T1, Produto, Notebook, 20, 19>

VENDA		
NR VENDA	**DESCRICAO**	**QUANTIDADE**
1	Notebook	1

Figura 48: Transação em Andamento: Nova Atualização do Log do Sistema.

Em seguida, a tabela PRODUTO é de fato alterada. E o ESTOQUE de Notebook passa a apresentar valor 19. Embora todas as operações da transação tenham sido efetivamente executadas, a transação ainda não foi compromissada, pois o registro **<T1, FIM>** ainda não foi inserido no arquivo LOG_DO_SISTEMA.

Por essa razão, caso uma falha ocorra nesse momento, o banco de dados ainda irá desfazer todas as operações associadas a **T1**. Ao ler o registro **<T1, Venda, Notebook, 1>** o banco de dados entenderá que deve excluir o registro de VENDA relativo ao Notebook. Por sua vez, ao ler o registro **<T1, Produto, Notebook, 20, 19>** o banco resolve alterar o registro de PRODUTO relativo ao Notebook, substituindo o novo valor de ESTOQUE (=19) pelo velho valor da mesma coluna (=20).

Dessa forma, todas as operações levadas a cabo pela transação T1 serão desfeitas. Para todos os efeitos, é como se a transação jamais tivesse existido. O banco de dados permanece consistente, pois o ESTOQUE terá um valor condizente com a realidade.

PRODUTO	
DESCRIÇÃO	**ESTOQUE**
Notebook	19
TV Plasma	13

LOG DO SISTEMA
<T1, INICIO>
<T1, Venda, Notebook, 1>
<T1, Produto, Notebook, 20, 19>

VENDA		
NR VENDA	**DESCRICAO**	**QUANTIDADE**
1	Notebook	1

Figura 49: Transação em vias de ser Compromissada.

No fim da transação, um registro **<T1, FIM>** é inserido no arquivo LOG_DO_SISTEMA. De agora em diante, qualquer falha ocorrida não levará o banco de dados a desfazer as operações de **T1**.

PRODUTO	
DESCRIÇÃO	**ESTOQUE**
Notebook	19
TV Plasma	13

LOG DO SISTEMA
<T1, INICIO>
<T1, Venda, Notebook, 1>
<T1, Produto, Notebook, 20, 19>
<T1, FIM>

VENDA		
NR VENDA	**DESCRICAO**	**QUANTIDADE**
1	Notebook	1

Figura 50: Transação Finalmente Compromissada (COMMIT).

A simples existência dos registros <T1,INICIO> e <T1,FIM> caracteriza a transação T1 como sendo compromissada.

8.2 Transações no MYSQL

Por padrão, o SGBD MySQL apresenta configuração no modo AUTOCOMMIT. Isso significa que qualquer atualização de tabela – INSERT, DELETE e UPDATE – é acompanhada da execução automática do comando COMMIT (Compromissar). Portanto, cada vez que um

comando INSERT, DELETE ou UPDATE é executado, o MySQL trata de confirmar as alterações automaticamente, tornando as transações compromissadas.

```
SET AUTOCOMMIT=0
```

O comando acima altera a configuração do MySQL, desabilitando o AUTOCOMMIT. Por outro lado, o comando abaixo volta a habilitar o AUTOCOMMIT.

```
SET AUTOCOMMIT=1
```

O comando START TRANSACTION inicializa uma transação, enquanto o comando COMMIT leva a transação ao estado de compromissada e o comando ROLLBACK aborta a transação.

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM PRODUTO;
+----+--------------------------+---------+---------+
| ID | NOME                     | ESTOQUE | PRECO   |
|  1 | DVD PLAYER               |      30 |  299.99 |
|  2 | APARELHO DE TELEVISÃO LCD |      20 | 1429.99 |
2 rows in set (0.00 sec)

mysql> INSERT INTO VENDA VALUES (2,1,5,290);
Query OK, 1 row affected (0.41 sec)

mysql> UPDATE PRODUTO SET ESTOQUE = 25
  WHERE ID = 1;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql>
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

A sequência de comandos acima exemplifica a utilização de transações no MySQL. O comando "START TRANSACTION;" inicia a transação. Em seguida fazemos uma consulta (SELECT) para visualizar o ESTOQUE dos produtos. Então, inserimos um novo registro na tabela VENDA (INSERT) e atualizamos (UPDATE) um registro de PRODUTO. Finalmente, confirmamos todas as atualizações com o comando COMMIT ("Compromissar"). Se tivéssemos concluído a transação com o comando ROLLBACK, todas as atualizações teriam sido desfeitas.

Se uma falha qualquer ocorresse entre os comandos INSERT e UPDATE, o próprio MYSQL executaria o comando ROLLBACK. Porém, isso somente seria possível se, antes dos comandos INSERT e UPDATE, o comando START TRANSACTION fosse executado. Se o usuário do banco de dados esquecer esse último comando, a transação simplesmente não existe e falhas poderiam conduzir o banco de dados a uma situação de inconsistência.

Conceitos

Atividade 33

No capítulo 7, foi apresentado um exemplo de procedimento armazenado (***stored procedure***) para a transferência de valores de uma conta corrente para outra.

Esse procedimento armazenado de nome TRANSFERENCIA recebe como parâmetros de entrada (1) o número da conta corrente de onde o dinheiro será retirado, (2) o número da conta onde será depositado e (3) o valor a ser transferido.

Apesar de pronto e testado, este procedimento armazenado não faz uso de **transação**. Sua tarefa é alterá-lo para que passe a trabalhar com uma transação. Ela impedirá a possibilidade de um valor debitado em uma conta não ser creditado em outra.

Solução:

```
DELIMITER //
CREATE PROCEDURE TRANSFERENCIA (
    ->   IN CONTA_ORIGEM INT,
    ->   IN CONTA_DESTINO INT,
    ->   IN VALOR NUMERIC(8,2)

    -> BEGIN

    ->  DECLARE SALDO_ORIGEM NUMERIC(8,2);
    ->  DECLARE SALDO_DESTINO NUMERIC(8,2);

    ->  SELECT SALDO INTO SALDO_ORIGEM
    ->  FROM CONTA_CORRENTE
    ->  WHERE NR_CONTA = CONTA_ORIGEM;

    ->  IF SALDO_ORIGEM < VALOR THEN

    ->    SELECT "FALHA NA TRANSFERENCIA - SALDO
           INSUFICIENTE";

    ->  ELSE

    ->    SELECT "TRANSFERENCIA AUTORIZADA - SALDO
           SUFICIENTE";

          START TRANSACTION;

    ->    UPDATE CONTA_CORRENTE SET SALDO =
                  (SALDO_ORIGEM - VALOR)
    ->    WHERE NR_CONTA = CONTA_ORIGEM;

    ->    SELECT "VALOR R$ ",VALOR," RETIRADO DA CONTA
                  ",CONTA_ORIGEM;

    ->    SELECT SALDO INTO SALDO_DESTINO
    ->    FROM CONTA_CORRENTE
    ->    WHERE NR_CONTA = CONTA_DESTINO;

    ->    UPDATE CONTA_CORRENTE SET SALDO =
                  (SALDO_DESTINO + VALOR)
    ->    WHERE NR_CONTA = CONTA_DESTINO;

          COMMIT;


    ->    SELECT "VALOR R$ ",VALOR, " DEPOSITADO NA
                  CONTA", CONTA_DESTINO;

    ->  END IF;
    -> END//

DELIMITER ;
```