Transação

O objetivo do tutorial é demonstrar como funciona a transação no MySQL.

Carlos Henrique Reis - 30415
Mateus Henrique Toledo - 34849
Victor Rodrigues da Silva - 31054

QUESTÕES DE REVISÃO – CAPÍTULO 10

1) **Explique a seguinte afirmação: a transação é uma unidade lógica de trabalho.**

A transação é uma unidade lógica de trabalho que deve ser concluída ou abortada inteiramente. Não são aceitos estágios intermediários. Uma transação com vários componentes não pode ser parcialmente concluída. Todos os comandos de SQL da transação devem ser concluídos com sucesso. Se um desses comandos falhar, toda a transação é desfeita até o estado original do banco de dados que existia antes de seu início. Uma transação bem-sucedida altera o banco de um estado consistente para outro.

Um banco de dados em estado consistente é aquele em que são satisfeitas as restrições de integridade de todos os dados.

2) **O SGBD não garante que o significado semântico da transação represente efetivamente o evento real. Quais são as possíveis consequências dessa limitação? Dê um exemplo.**

Para assegurar a consistência do Banco de Dados, toda transação deve começar com o banco de dados em um estado considerado consistente. Caso ele não esteja nesse estado, a transação resultará em um banco de dados inconsistente que violará suas regras de integridade e negócio.

A maioria das transações reais é formada por duas ou mais solicitações. A solicitação de banco de dados é o equivalente a um único comando de SQL em um aplicativo ou transação.

Nem todas as transações atualizam o banco de dados, por exemplo, ao fazermos somente uma consulta em alguma tabela do sistema.

A garantia que temos quanto ao código SQL é de sua sintaxe, pois se o comando estiver escrito de forma errada, um erro ocorrerá. Entretanto estando o comando correto e ação executada não representar de fato o que ocorre no mundo real, erros e inconsistências serão acrescentados de forma compulsória à sua execução, por exemplo, ao invés de subtrairmos um determinado valor que

representa uma transferência de uma conta à outra conta, o adicionarmos, estaremos introduzindo uma inconsistência nos dados que não representa de fato o que ocorre no mundo real. E mesmo implementando o mecanismo de transações, o banco de dados estará errado, pois a sintaxe programada pelo programador de Banco de Dados não corresponderá a realidade da transação.

3) **Liste e discuta as quatro propriedades de transações**

Uma unidade lógica de trabalho deve mostrar quatro propriedades, designadas pelas iniciais ACID (atomicidade, consistência, isolamento e durabilidade), para que seja qualificada como uma transação.

***Atomicidade*:** uma transação deve ser uma unidade atômica de trabalho; ou todas as suas modificações de dados são executadas ou nenhuma delas é executada.

***Consistência*:** quando concluída, uma transação deve deixar todos os dados em um estado consistente. Em um banco de dados relacional, todas as regras devem ser aplicadas às modificações da transação para manter toda a integridade dos dados. Todas as estruturas de dados internas, tais como índices em árvore B ou listas duplamente vinculadas, devem estar corretas ao término da transação.

***Isolamento*:** modificações feitas por transações simultâneas devem ser isoladas das modificações feitas por qualquer outra transação simultânea. Uma transação reconhece os dados no estado em que estavam antes de outra transação simultânea tê-los modificado ou reconhece os dados depois que a segunda transação tiver sido concluída, mas não reconhece um estado intermediário. Isso é chamado serializabilidade porque resulta na capacidade de recarregar os dados iniciais e reexecutar uma série de transações de modo que os dados obtidos estejam no mesmo estado em que estavam depois que as transações originais foram executadas.

***Durabilidade*:** depois que uma transação tiver sido concluída, seus efeitos ficam permanentemente no sistema. As modificações persistem até mesmo no caso de uma queda do sistema.

SESSÃO DE PROBLEMAS – CAPÍTULO 10

1) **Suponha que você seja o fabricante do produto ABC, composto pelas peças A, B e C. Cada vez que um novo produto ABC é criado, é necessário adicioná-lo ao estoque utilizando o PROD_QOH de uma tabela chamada PRODUTO. E a cada vez que o produto é criado, as peças A, B e C de estoque devem ser produzidas de uma unidade cada, utilizando PEÇA_QOH da tabela chamada PEÇA.**

a) **Quantas solicitações de bancos de dados você pode identificar para atualização de estoque de PRODUTO e PEÇA?**

SOLICITAÇÃO 1 -> *Update* do produto (PROD_QOH)
SOLICITAÇÃO 2 -> *Update* da peça A (PEÇA_QOH)
SOLICITAÇÃO 3 -> *Update* da peça B (PEÇA_QOH)
SOLICITAÇÃO 4 -> *Update* da peça C (PEÇA_QOH)

Ou seja, são realizadas no mínimo quatro solicitações.

b) **Utilizando SQL, apresente todas as solicitações identificadas na etapa 1:**

Supondo que não será criado nem artifício facilitador para realizar esta função, as solicitações serão representadas da maneira simples:

-- SOLICITAÇÃO 1 (ATUALIZA A QUANTIDADE DE PRODUTOS NO ESTOQUE):

```
UPDATE PRODUTO
SET PROD_QOH = PROD_QOH+1
WHERE PROD_CODIG = 'ABC';
```

-- SOLICITAÇÃO 2 (QUE ENGLOBA TODAS AS SOLICITAÇÕES DE ATUALIZAÇÃO

DAS PEÇAS NO ESTOQUE):

```
UPDATE PEÇA
SET PEÇA_QOH = PEÇA_QOH+1
WHERE PEÇA_CODIG = 'A' OR PEÇA_CODIG = 'B' OR PEÇA_CODIG = 'C';
```

Para realização desta tarefa poderiam ser utilizados os conceitos de TRIGGER, o que de certa forma automatizaria o processo, contudo, TRIGGERs serão abordados nas próximas páginas.

c) **Apresente as transações completas:**

-- TRANSAÇÕES:

```
START TRANSACTION;
UPDATE PRODUTO SET PROD_QOH = PROD_QOH+1 WHERE PROD_CODIG = 'ABC';
UPDATE PEÇA SET PEÇA_QOH = PEÇA_QOH+1 WHERE PEÇA_CODIG = 'A' OR
PEÇA_CODIG = 'B' OR PEÇA_CODIG = 'C';
COMMIT;
```

Questão 2: Investigar como se faz para desabilitar o *autocommit*.

O AUTOCOMMIT é ativado por padrão, logo o MySQL confirma implicitamente cada instrução como uma transação.

Uma vez desativado, será possível manusear cada transação, porém fica a critério do DBA/DEV. Para desativar o autocommit:

```
set session autocommit=0;
set autocommit=0;
```

Para verificar a configuração através do select:

```
select @@autocommit;
```

A partir da versão 5.5.8 do MySQL, é possível definir o autocommit=0 de forma global na seção do [mysqld] no seu arquivo de configuração (default my.cnf).

Questão 3: Quais são as formas possíveis de iniciar uma transação no SGBD?

Primeiramente no MySQL apenas o InnoDB suporta transações a nível ACID, logo estas opções não são válidas ou não tem o efeito esperado em outros Engines.

Sintaxe:

```
START TRANSACTION OR BEGIN
instruções SQL
COMMIT OR ROLLBACK
SET autocommit = {0 | 1}(opcional)
```

Exemplo:

```
START TRANSACTION;
SELECT * FROM table1 WHERE type=1;
UPDATE table2 SET summary=123 WHERE type=1;
COMMIT;
```

- Instruções de Controle SQL de uma transação:

`Start Transaction` (ou `BEGIN`): Inicia explicitamente uma nova transação, mantendo ela aberta até que seja fechada (concluída) por um `COMMIT` ou `ROLLBACK`.

`SavePoint`: Um ponto delimitado na transação, que oferece a possibilidade de um `rollback` de qualquer comando executado após este ponto salvo. **Cuidado: Ao efetuar um novo `savepoint` com o mesmo nome, o antigo será sobreescrito pelo novo criado.

`Rollback to savepoint`: Efetua o `rollback` da transação até o ponto criado. Este ponto continuará a existir mesmo após o `rollback`, permitindo que ele seja reutilizado.

`Rollback`: Cancela todas as alterações efetuadas na transação.

`Commit`: Torna permanente as alterações tratadas na transação ao banco de dados.

Questão 4: Quais são as opções para parar a execução de uma trigger (interromper a transação), caso alguma condição não seja validada?

Uma forma possível para se abortar transações no MySQL é utilizando um `SIGNAL`. Após utilizar uma estrutura condicional no `trigger` e for necessário encerrar a transação pode ser utilizado o seguinte comando, por exemplo:

```
SIGNAL SQLSTATE '45000';
```

Esse comando geraria uma exceção do tipo "Exceção não manipulada definida pelo usuário". O valor 45000 no `SIGNAL` é para esta exceção.

Podem ser definidos outros tipos de exceções ou outros valores para definir a exceção conforme a documentação presente em https://dev.mysql.com/doc/refman/5.7/en/signal.html.

Questão 5: Normalmente, na sintaxe de criação de trigger aparece o termo 'FOR EACH ROW'. O que esse termo significa? Qual a diferença entre um trigger de linha e um de instrução? Seu SGBD implementa quais tipos de triggers?

O termo `for each row` em um `trigger` indica que as instruções definidas no `trigger` devem ser executadas para cada tupla alterada. Por exemplo, se um `trigger` for disparado após o `UPDATE` em uma tabela, as instruções do `trigger` serão executadas para todas as tuplas alteradas pelo evento `update`.

Um `trigger` de linha é executado para cada linha que for alterada pelo evento que o dispara (como citado acima, `for each row`). Um `trigger` de instrução é executado apenas uma vez para a instrução SQL que o dispara.

Exemplo: Um `trigger` de linha após uma instrução `update` sobre 10 tuplas seria executado 10 vezes, uma para cada tupla. Um `trigger` de instrução realizado após um `update` em 10 tuplas seria executado apenas uma vez.

Segundo a documentação do SGBD na versão mySQL 5.7 (a que estamos usando) a cláusula `FOR EACH ROW` é obrigatória na sintaxe de criação do `trigger`, ou seja, por padrão há apenas `triggers` de linha nesse sistema gerenciador de banco de dados.

Questão 6: Dada a tabela TBCONTA ilustrada abaixo, originalmente vazia:


```
CREATE TABLE TBCONTA
(
   Codigo INT NOT NULL,
   conta INT NOT NULL,
   saldo INT NOT NULL,
   limite INT NOT NULL,
);
```

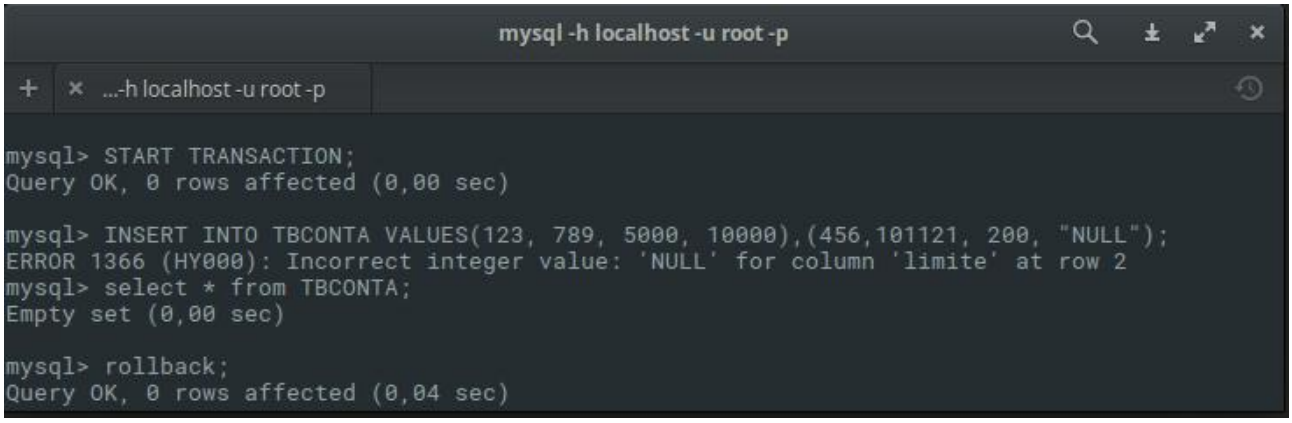
a) Reproduza no seu SGBD as ações ilustradas abaixo, e verifique o que acontece. Discuta e explique seus resultados.

- Inicializa a transação
  - Inserir o registro (123, 5000, 10000)
  - Inserir o registro (456, 200, “NULL”)
- Se houver erros na transação: Rollback
- Senão: Commit

Após a criação da tabela, foi executada a seguinte transação no SGBD:

```
START TRANSACTION;
INSERT INTO TBCONTA VALUES(123, 789, 5000, 10000),(456,101121, 200,
"NULL");
rollback;
```

O resultado foi o seguinte:



Como mostrado na imagem acima, o SGBD ao detectar erro em uma das operações da transação, a encerrou automaticamente. Isso mostra que o MySQL garante a atomicidade, uma das propriedades ACID. Para comprovar isso, tentamos realizar essa transação novamente e realizar o `commit` após a inserção na tabela TBCONTA.

Mesmo após tentar persistir os dados com o comando `commit`, isso não ocorreu, já que a inserção de valor nulo em campo obrigatório foi negada.

b) Considerando agora que a tabela TBCONTA possui os registros identificados pelos códigos 123 e 456, reproduza no seu SGBD as ações ilustradas abaixo, e verifique o que acontece. Discuta e explique seus resultados.

- Inicializa a transação
  - Update no registro 123, atualizando o saldo para saldo - 1000

- Update no registro 456, atualizando o saldo para saldo + 1000000000000

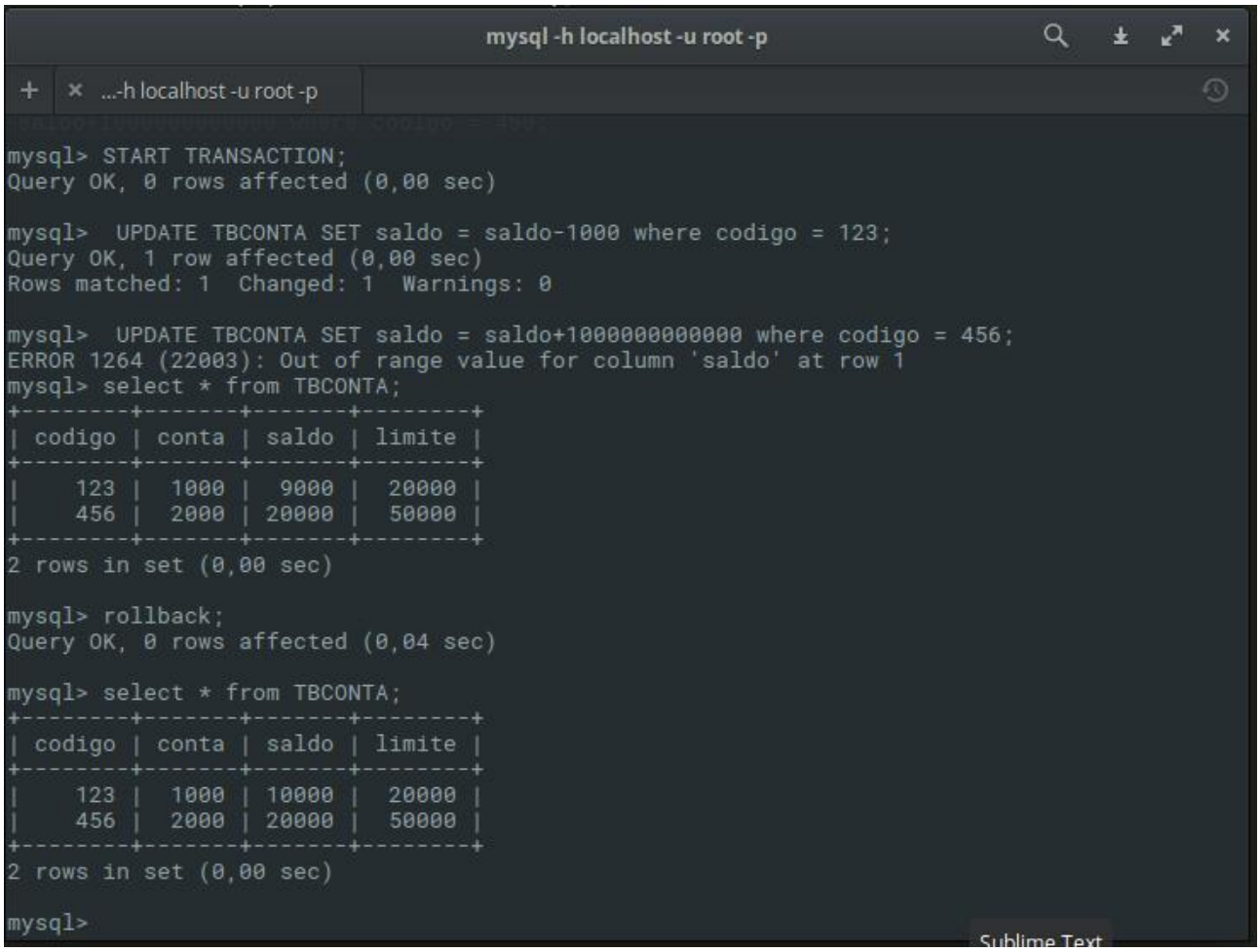
- Se houver erros na transação: Rollback
- Senão: Commit

Para executarmos as operações de `INSERT` na tabela TBCONTA foram inseridas duas tuplas:

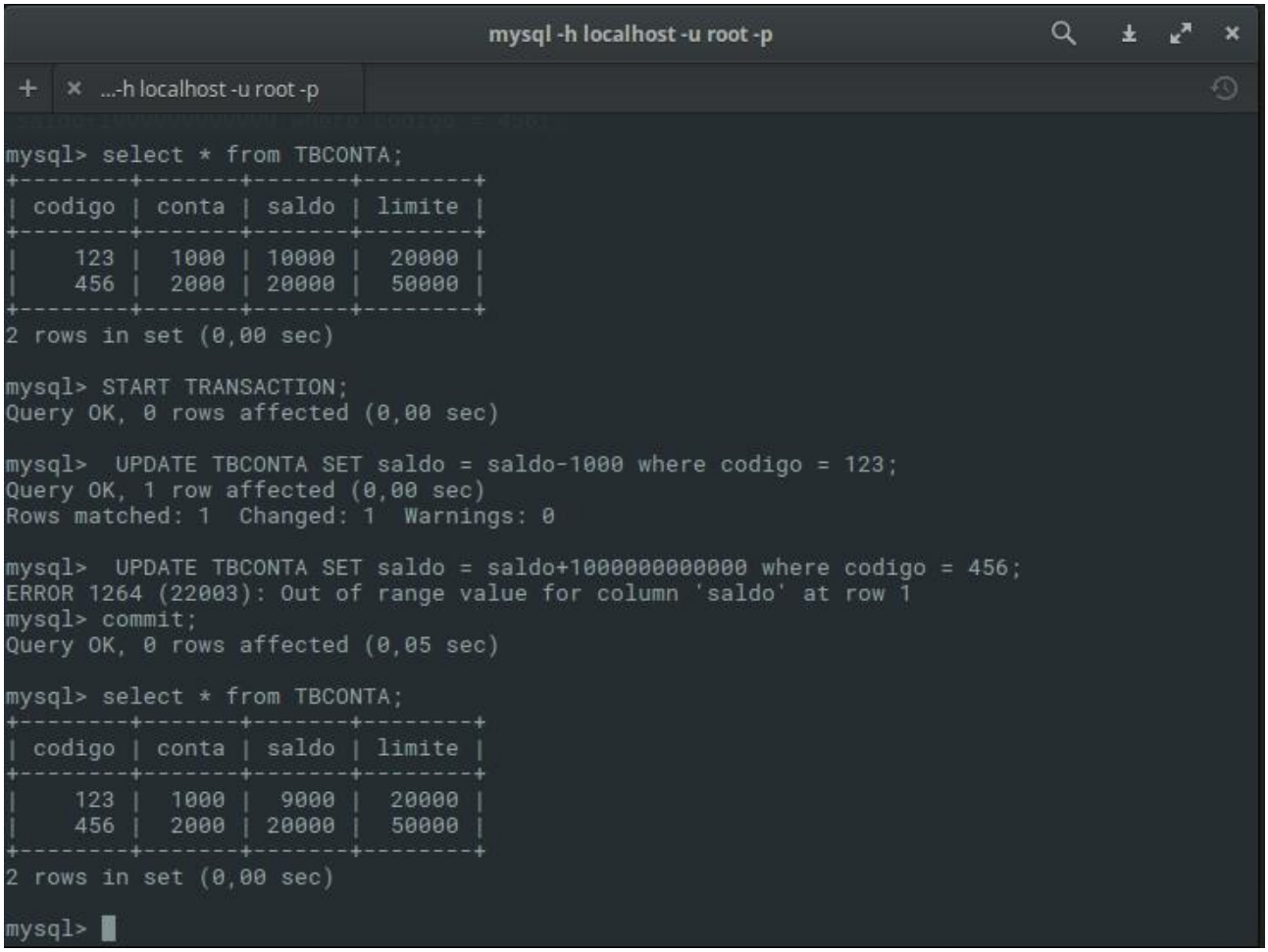
A transação que será realizada sobre as duas tuplas recém adicionadas é a seguinte:

```
START TRANSACTION;
UPDATE TBCONTA SET saldo = saldo-1000 where codigo = 123;
UPDATE TBCONTA SET saldo = saldo+1000000000000 where codigo = 456;
rollback;
```

O resultado da execução no MySQL foi a seguinte:



Como uma das operações ocorreu com erro, resolvemos mostrar o estado da tabela TBCONTA durante a transação com um `select` e ela indicou que houve alteração em uma tupla. Após o `rollback` a tabela estava da mesma forma que antes da transação. Como uma das tuplas foi alterada durante a transação, resolvemos testar a operação de `commit` após o `update` que teve erro durante sua execução:



Mesmo após o erro em uma das operações, o comando *commit* fez com que o MySQL salvasse a operação realizada com sucesso, alterando apenas uma das duas tuplas.