**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.

1. **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.

1. **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.**
2. **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.

1. **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 QUNATIDADE 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.

1. **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*.

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

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

**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?

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

**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

**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

**1. Estudar as estruturas de índices disponíveis no seu SGBD.**

Todos os registros no MySQL são armazenados em **B-Tree**. Os registros do índice ficam armazenados nas folhas da árvore, cada um com tamanho padrão de 16 Kb. A ***engine* InnoDB** tenta sempre deixar 1/16 de páginas livres para realizar inserções e atualizações nos registros de índices de forma menos custosa.

Quando é definida a chave primária da tabela o MySQL cria um índice agrupado (o índice primário do MySQL). Quando é definido um **índice secundário** sobre determinado(s) atributo(s) da tabela, um dos campos salvos no registro de índices é a chave primária da tupla que contém a chave de índice. Tendo a chave primária, após encontrar o valor desejado com uma busca no índice, para obter a tupla que contém esse valor é realizada uma busca no índice agrupado (índice primário MySQL).

Outro tipo de indexação utilizada pelo MySQL são os **índices *hash***. Mesmo sendo diferentes, um índice *hash* é construído sobre um índice de árvore B existente. O analisador de consultas do MySQL quando percebe que um índice *hash* é eficiente para determinada consulta e cabe na memória principal, esse índice é criado automaticamente e a consulta é realizada utilizando-o. Vale ressaltar que a *engine* MyISAM não tem os índices *hash*.

| **Storage Engine** | **Permissible Index Types** |
| --- | --- |
| [InnoDB](https://dev.mysql.com/doc/refman/5.7/en/innodb-storage-engine.html" \o "Chapter 15 The InnoDB Storage Engine) | BTREE |
| [MyISAM](https://dev.mysql.com/doc/refman/5.7/en/myisam-storage-engine.html" \o "16.2 The MyISAM Storage Engine) | BTREE |
| [MEMORY](https://dev.mysql.com/doc/refman/5.7/en/memory-storage-engine.html" \o "16.3 The MEMORY Storage Engine)/HEAP | HASH, BTREE |
| [NDB](https://dev.mysql.com/doc/refman/5.7/en/mysql-cluster.html" \o "Chapter 20 MySQL NDB Cluster 7.5) | HASH, BTREE |

**2. Detalhe a sintaxe de criação e remoção de índices.**

**Criação do índice:**

CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX ***index\_name***

[***index\_type***]

ON ***tbl\_name*** (***index\_col\_name***, ...)

[***index\_option***]

[***algorithm\_option*** | ***lock\_option***] ...

***index\_col\_name****:*

***col\_name*** [(***length***)] [ASC | DESC]

***index\_type****:*

USING {BTREE | HASH | RTREE}

***index\_option****:*

KEY\_BLOCK\_SIZE [=] ***value***

| ***index\_type***

| WITH PARSER ***parser\_name***

| COMMENT '***string***'

***algorithm\_option****:*

ALGORITHM [=] {DEFAULT|INPLACE|COPY}

***lock\_option****:*

LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}

**Exemplo:**

CREATE INDEX part\_of\_name ON customer (name(10));

**Remoção de índices:**

DROP INDEX ***index\_name*** ON ***tbl\_name***

[***algorithm\_option*** | ***lock\_option***]

***algorithm\_option:***

ALGORITHM [=] {DEFAULT|INPLACE|COPY}

***lock\_option:***

LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}

**Exemplo:**

DROP INDEX PRIMARY ON t;

**3. Como listar os índices de uma tabela do banco?**

SHOW {INDEX | INDEXES | KEYS}

{FROM | IN} ***tbl\_name***

[{FROM | IN} ***db\_name***]

[WHERE ***expr***]

**Exemplo:**

SHOW INDEX FROM mytable FROM mydb;

SHOW INDEX FROM mydb.mytable;

**4. Criar uma view no banco. É possível indexar essa view?**

Utilizando o banco de dados *world* disponibilizado na documentação do MySQL, temos:

CREATE VIEW dadoscidade AS SELECT ct.Name, cnt.Name AS estado, cl.Language FROM city ct, country cnt, countrylanguage cl WHERE ct.CountryCode = cnt.Code AND cnt.Code = cl.CountryCode;

Ao tentar indexar essa *view*, o MySQL apontou erro. Na documentação do SGBD não há opções para indexação sobre *views*. Porém, como uma *view* não materializada executa a consulta que a define toda vez que é chamada, o índice deve ser criado na tabela que a *view* utiliza.

**5. Quais ferramentas de análise de consulta seu SGBD oferece?**

Para realizar a análise de consulta no MySQL, têm-se os métodos **EXPLAIN** e **ANALYZE TABLE**. EXPLAIN nome\_tabela é um sinônimo para DESCRIBE nome\_tabela ou SHOW COLUMNS FROM nome\_tabela. Quando uma instrução SELECT for precedida da palavra chave EXPLAIN, o MySQL explicará como ele deve processar a SELECT, fornecendo informação sobre como as tabelas estão sendo unidas e em qual ordem. Com a ajuda de EXPLAIN, você pode ver quando devem ser adicionados índices à tabelas para obter uma SELECT mais rápida que utiliza índices para encontrar os registros.

Já o ANALYZE TABLE é utilizado para atualizar estatísticas de tabela tais como a cardinalidade das chaves que podem afetar a escolha que o otimizador faz.

**6. Demonstre os índices criados no momento da criação das tabelas.**

Com o MySQL aberto você deve selecionar o banco de dados que quer utilizar. Usaremos o banco *world* disponibilizado na documentação do MySQL para nosso exemplo.

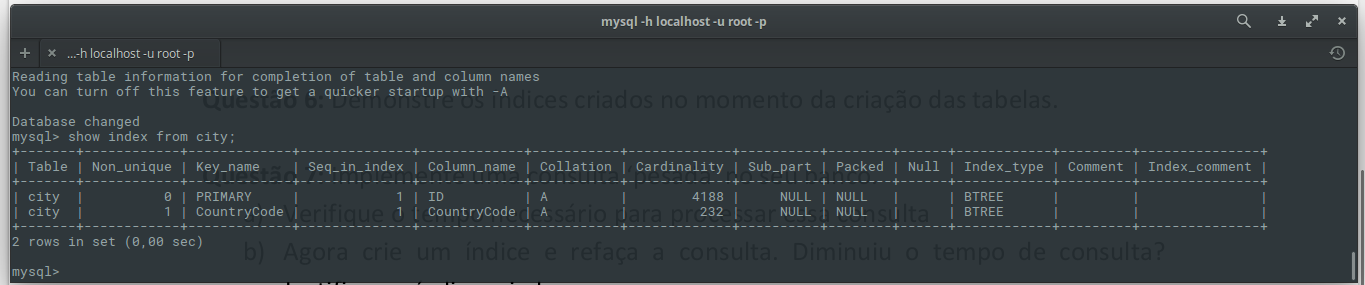
use world;

O próximo passo é usar o seguinte comando para exibir os índices sobre determinada tabela:

SHOW INDEX FROM <nometabela>

Ex.:

SHOW INDEX FROM city;



**Imagem 1:** Resultado da visualização de índices na tabela *city*.

Esse comando lista todos os índices presentes na tabela. Para listar os índices provenientes da criação da tabela, basta executar o comando após a criação.

**7. Implemente uma consulta ‘pesada’ no seu banco.**

**a) Verifique o tempo necessário para processar essa consulta**

A consulta foi feita em um banco de dados que possui uma tabela com 19972 registros. Primeiramente, foi analisado quanto tempo é necessário para a consulta de todos os registros dessa tabela, e o tempo foi de 0,45 segundos.

Para a análise das consultas, o código utilizado foi:

SELECT \* FROM contact WHERE LastName = 'Stewart' AND FirstName = 'Isabella';

Para este caso, o tempo gasto foi de 0,20 segundos, retornando 6542 resgistros.

**b) Agora crie um índice e refaça a consulta. Diminuiu o tempo de consulta? Justifique o índice criado.**

Para a criação do índice, foi utilizado um índice composto de dois atributos, sendo eles ‘LastName’ e ‘FirstName’, ambos do tipo varchar. O código desta vez foi:

CREATE INDEX index\_teste USING btree ON contact (

LastName, FirstName);

O tempo gasto para a criação deste índice foi de 2.74 segundos.

Tendo o tempo da consulta e o índice criado, agora já podemos executar novamente a mesma consulta.

Com a consulta feita pela segunda fez, analisamos que o tempo gasto foi de 0.14 segundos para a execução da mesma consulta. Este tempo menor é justificado pelo caminho utilizado pela consulta com índice. Nestes casos, o espaço de busca é reduzido, consequentemente gastando um tempo menor do que a consulta comum.

**8. Avalie por meio de consultas e descreva a impressão do grupo sobre:**

**a) Índice de Hash**

Criação do índice, o parâmetro utilizado foi o ‘LastName’:

CREATE INDEX index\_hash USING hash ON contact (

LastName);

Resultado:

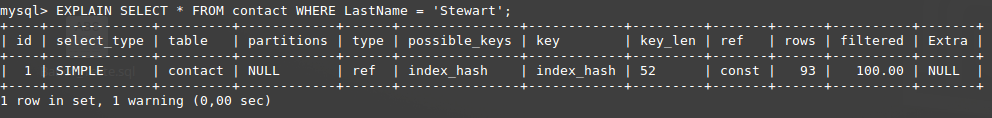
Query OK, 0 rows affected (1,43 sec)

Records: 0 Duplicates: 0 Warnings: 0

Pesquisa:

EXPLAIN SELECT \* FROM contact WHERE LastName = 'Stewart';

Resultado:



**b) Índice composto**

Criação do índice. O parâmetro utilizado foi ‘LastName’ e o ‘FirstName’, ambos utilizados na busca, no caso a indexação é realizada com base em uma B-tree, contudo dois parâmetros são utilizados na criação do mesmo:

CREATE INDEX index\_composto USING btree ON contact (

LastName, FirstName);

Resultado:

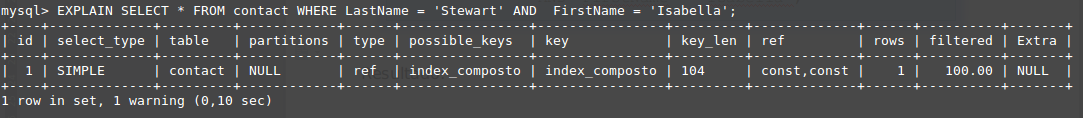
Query OK, 0 rows affected (2,83 sec)

Records: 0 Duplicates: 0 Warnings: 0

Pesquisa:

EXPLAIN SELECT \* FROM contact WHERE LastName = 'Stewart' AND FirstName = 'Isabella';

Resultado:



**c) Índice em um campo BLOB**

Para realização destes testes, uma nova tabela teve que ser criada, para que assim os INSERTs de dados do tipo *BLOB*, nomeados como BLOB no MySQL, pudessem ser realizados da maneira correta. Esta nova tabela é composta apenas por dois atributos, o primeiro é o nome, uma *varchar* de 30 posições e a segunda é um *BLOB*. O comando para criação da tabela foi o seguinte:

CREATE TABLE arquivo (nome varchar(30) PRIMARY KEY, data BLOB);

Após criada a tabela, os dados binários a serem inseridos, precisam ser convertidos previamente para uma cadeia de base64, uma opção rápida é usar um site (http://www.motobit.com/util/base64-decoder-encoder.asp) que tem como propósito fazer exatamente esta conversão, assim basta dar o seguinte comando, substituindo *file\_data* pela *string* criada no site anterior:

INSERT INTO arquivo VALUES ( 'image1, decode('', 'file\_data') );

Tal metodologia aplicada acima não é muito conveniente, é recomendado que se trate as conversões para base64 ocorra diretamente na aplicação, levando em consideração as operações de *encode* (inserção e atualização)e *decode* (recuperação do arquivo).

Criação do índice, utilizando o método *hash* por exemplo:

CREATE INDEX index\_blob USING hash ON arquivo(

data);

Resultado:



Como o próprio SGBD indica na criação do índice, o mesmo não é indicado por ser muito longo, ou seja, demanda um custo de processamento maior, o que não é bom para indexação.

**d) Refaça a consulta utilizando *Prepared Statement* e reavalie a performance.**

PREPARE consulta SELECT \* FROM contact WHERE LastName = 'Stewart' AND FirstName = 'Isabella';

Após a reavaliação das consultas, fica evidente o ganho de performance com o uso do *Prepared Statement*, na tabela em questão haviam cerca de 20.000 registros, no entanto em tabelas com o dobro, ou mesmo dez vezes este tamanho, o ganho pode fazer grande diferença no final.