Indexação

O objetivo deste tutorial é demonstrar como funciona a indexação no MySQL. Serão abordadas as estruturas de índices no SGBD, a sintaxe de criação e remoção destes índices, como listá-los e ferramentas de análise de consulta que envolve a utilização do mesmo.

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

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	BTREE
MyISAM	BTREE
MEMORY/HEAP	HASH, BTREE
NDB	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.

Indexação

mysql>

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:

```
mysql> EXPLAIN SELECT * FROM contact WHERE LastName = 'Stewart';
+----+-------------+---------+------------+------+---------------+------------+---------+-------+------+----------+-------+
| id | select_type | table   | partitions | type | possible_keys | key        | key_len | ref   | rows | filtered | Extra |
+----+-------------+---------+------------+------+---------------+------------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | contact | NULL       | ref  | index_hash    | index_hash | 52      | const |   93 |   100.00 | NULL  |
+----+-------------+---------+------------+------+---------------+------------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0,00 sec)
```

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:

```
mysql> EXPLAIN SELECT * FROM contact WHERE LastName = 'Stewart' AND  FirstName = 'Isabella';
+----+-------------+---------+------------+------+---------------+---------------+---------+-------------+------+----------+-------+
| id | select_type | table   | partitions | type | possible_keys | key           | key_len | ref         | rows | filtered | Extra |
+----+-------------+---------+------------+------+---------------+---------------+---------+-------------+------+----------+-------+
|  1 | SIMPLE      | contact | NULL       | ref  | index_composto | index_composto | 104     | const,const |    1 |   100.00 | NULL  |
+----+-------------+---------+------------+------+---------------+---------------+---------+-------------+------+----------+-------+
1 row in set, 1 warning (0,10 sec)
```

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:

```
ERROR 1170 (42000): BLOB/TEXT column 'data' used in key specification without a key length
```

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.