Otimização

A imensa facilidade de executar comandos SELECT eventualmente associados a JOINs pode dar a idéia errada de que tudo tem o mesmo preço sob MySQL. Esta afirmação pode ser verdadeira para aplicativos *one way* ou para aplicações com baixo nível de demanda. Outra dificuldade aqui é a diferença de tempos entre homem e computador. Um segundo de diferença em uma transação não será notado a menos que essa transação seja executada 1.000.000 de vezes por dia, quando então serão 277 horas por dia (?) de atraso.

Indexação de colunas

É o primeiro e principal critério de otimização. Há 50 anos, quando as técnicas de indexação ainda não estavam disponíveis, usava-se o critério da ordem física. Sistemas de informação eram construídos baseados neste princípio. Hoje, com o advento da teoria relacional que prega a irrelevância da posição física do dado no arquivo, toda a ênfase se volta para o índice. Fazendo-se uma analogia do que se disse, imagine um dicionário onde as palavras estão em ordem aleatória (como aliás é o caso dos dicionários analógicos no qual não se procura uma palavra mas sim uma idéia). Nesse caso, apenas um índice permite um mínimo de sucesso na consulta. Por outro lado, suponha querer achar a referência a uma pessoa num livro de história, sem existir um índice, a tarefa fica complicada...

Eis alguns números para provar o que se diz: Seja um arquivo no banco de dados com 1.000.000 de linhas. Uma consulta qualquer exigirá em média 500.000 acessos em caso de sucesso e 1.000.000 de acessos em caso de fracasso. Havendo índice, (supondo uma B-árvore com ramificação de 50 chaves por nodo), a quantidade de acessos baixa para 4 em ambos os casos. Noves fora o trabalho de criar e manter o índice, o ganho é indesmentível. O índice é uma estrutura auxiliar que permite chegar a uma linha específica dentro da tabela pelo conteúdo de um atributo (cuja coluna tenha sido usada para criar o índice) em poucos acessos. Note que resultado parecido seria obtido ao ordenar essa tabela por essa coluna, mas o índice dá um ganho adicional que é a possibilidade de múltiplos índices (colunas) para a mesma tabela. A ordenação física só poderia resolver UMA ordenação.

A velocidade tende a aumentar mais ainda quando uma consulta envolve duas tabelas. Para ver como suponha as seguintes duas tabelas

```
coluna 1        coluna 2
--------        --------
    1               1
    2               2
   ...              3
  3000             ...
                   5000
```

Imagine a seguinte operação

```
SELECT * FROM tabela1 INNER JOIN tabela2
ON tabela1.coluna1 = tabela2.coluna2
WHERE tabela1.coluna1=23
```

Sem indexação a busca demandaria 15.000.000 de acessos (os 3000 da tabela1 vezes os 5000 da tabela2) independentemente do valor procurado ser 23 ou qualquer outro. Se a coluna1 de tabela1 estivesse indexada o número seria bem reduzido e se também a coluna 2 estivesse indexada o resultado seria melhor ainda. O uso de indexação precisa ser bem balanceado, já que ele apresenta os seguintes inconvenientes:

- Maior tempo de execução em comandos INSERT, UPDATE e DELETE. Ao fazer estas operações em tabelas indexadas, o MySQL precisa também atualizar os índices. Alternativamente, em grandes movimentações há a possibilidade de fazer tudo e só depois atualizar os índices (como por exemplo usando LOAD DATA).

- Espaço em disco é grandemente consumido pelos índices. Até porque uma b-árvore sempre é criada (e mantida) meio vazia. Note-se que índices são redundantes em relação à tabela original, pois sempre duplicam dados.

A indexação deve ser usada quando a coluna em questão for utilizada como parâmetro em algum dos seguintes argumentos:

- WHERE, JOIN: Além do acesso direto, neste caso a consulta se encerra assim que um valor inválido for encontrado.

- GROUP BY, ORDER BY: Permite agrupar ou já responder na ordem solicitada

- DISTINCT: de maneira similar ao GROUP BY permite avançar aos saltos.

- MIN e MAX: recupera o conteúdo pedido no início ou fim da coluna

Não se recomenda o uso de índices em colunas com baixa cardinalidade, por exemplo uma coluna de estado civil ou similar.

Mais truques

Um truque chamado isolamento de índices pede que os índices sejam citados isoladamente nos comandos SQL. Por exemplo, Supondo um índice pela col2 examinemos os dois comandos a seguir

```
SELECT * FROM tab WHERE col1 = col2 - 1;
SELECT * FROM tab WHERE col1 + 1 = col2;
```

Ambos os comandos acima são equivalentes (geram o mesmo resultado), mas o primeiro ignora o índice existente, já que cada conteúdo terá que ser operado antes de entregue ao select. O segundo usa integralmente o índice. Outro truque sugere estudar a ordem de utilização dos índices em JOINS. Os comandos aqui são: USE INDEX, FORCE INDEX, IGNORE INDEX e STRAIGHT_JOIN que sobrepujam a maneira padrão do MySQL atender a comandos. Este tipo de análise deve ser reservado às consultas frequentes e depois da análise do desempenho dos comandos originais.

Atualização automática de índices é outro mecanismo grande consumidor de recursos que pode ser controlado. No dia a dia é bom que este automatismo exista, o que garante índices sempre *up to date*. Entretanto, suponha por exemplo, a adição de 1.000.000 de linhas a uma tabela. Havendo 3 índices, serão 3.000.000 de atualizações de índices. Deixando a atualização ser feita apenas ao final serão 3 atualiações apenas. Os comandos que desabilitam índices em tabelas e reabilitam-nos depois são DISABLE KEYS e ENABLE KEYS. Outra possibilidade é inserir os comandos de inserção, alteração ou exclusão massiva de informações dentro de blocos de transação (com START TRANSACTION e COMMIT), ou então bloquear a tabela por meio de LOCK TABLES e UNLOCK TABLES. Nos dois casos acima, o índice só será atualizado ao final do processo.

Além disso, na inserção de dados deve-se privilegiar a síntaxe multi-registros conhecida como EXTENDED como em

```
INSERT INTO tab VALUES (reg1), (reg2), (...);
```

Pode-se incluir o comando DELAYED após a palavra INSERT. Neste caso a inclusão só será atendida quando não houver conexões sendo atendidas pelo MySQL.

A observação da comparação em expressões SQL também tem potencial de melhoria de desempenho. Lembrando que o MySQL converte automaticamente expressões de tipos diferentes para o mesmo tipo. Por exemplo, no comando

```
SELECT * FROM tab WHERE colstring = 1
```

Esta consulta exige que para cada linha na tabela haja uma conversão da coluna de modo que os conteúdos '1', '1.0' e '0001' sejam corretamente testados. Pior ainda, se esta coluna tiver um índice, ele não será usado. A solução para este problema está em planejar muito bem as colunas do banco de dados e também a execução das consultas. De um modo geral a diversidade de tipos de dado deve ser usada em benefício do projeto ainda que aumente a complexidade do projeto.

Outro foco de perda de desempenho nos arquivos é a fragmentação que ocorre em dados do tipo variável (CHAR e VARCHAR) ou mesmo em tabelas que sofrem muitas atualizações por intervalo de tempo. O comando que defragmenta tabelas é OPTIMIZE TABLE. Um dos seus parâmetros indica que esta tarefa não deve ser "logada" no log binário (já que escravos não necessariamente precisam estar fragmentados). Outra maneira (explícita) de realizar defragmentação é exportar e importar os dados.

As expressões MySQl podem ser analisadas quanto à possível otimização pelo comando EXPLAIN seguido do comando a analisar. A lista de informações é devolvidas é grande, mas uma importante é ROWS que indica uma estimativa de quantas linhas serão acessadas para responder à consulta. Diversos comandos que retornem o mesmo resultados podem ser analisados e comparados para estudar o desempenho do banco de dados. Finalmente, para otimização geral do ambiente MySQL há diversas opções a ser feitas durante a administração do banco de dados. Há muitas variáveis de controle que gerenciam aspectos fundamentais do ambiente. Eis algumas: Prioridade de execução: balanceia operações de leitura (SELECTs) e operações de atualização. Estes dois tipos têm filas de atendimento separadas. Dados podem ser carregados em memória, com maior ou menor espaço de buffers, com uso de cache (espaço para guardar as consultas mais frequentes), com limitação no número de conexões, etc. Um comando interessante é

```
SELECT BENCHMARK(10000000,1+1);
```

Pelo qual se pede ao servidor que meça o tempo gasto para fazer 10.000.000 de adições. A análise deste tempo pode informar sobre o desempenho da CPU do sistema.

☞ Para você fazer