Universidade Estadual de Santa Cruz

Indexes e Explains

Eduardo Reis Nobre

15 de maio de 2018

Sumário

1 Introdução

Um banco de dados é capaz de realizar consultas complexas em grande quantidade de dados ainda mantendo velocidades aceitáveis. Sofisticados sistemas e algoritmos são utilizados quando uma consulta é realizada no banco de dados. Uma das estratégias que podem ser utilizadas é a utilização de índices; tais índices permitem que o banco não realize uma pesquisa sequencial, para algo mais eficiente. Através do comando explain o banco lhe informará o plano de pesquisa que o banco planejou para essa consulta. Esse relatório mostra as diferenças entre implementações com índices e sem índices, assim como o que ocorre quando um índice não é utilizado.

Pesquisas sequenciais são a forma mais simples de se encontrar um item em um conjunto de dados. O seu problema se encontra quando o elemento desejado se encontra na última posição do seu conjunto de dados, temos um tempo geral da ordem de N. Tendo uma base de dados organizada em uma estrutura de dados mais sofisticada, podemos então, reduzir drasticamente o tempo de busca de um dado. Uma dessas opções é o uso de uma estrutura em árvore, esse algoritmo pode ser aplicado sobre um conjunto de dados ordenados e ele se baseia na idéia da bissecção, onde a partir de um nó inicial, se decide se o valor procurado se encontra a direita ou à esquerda, escolhendo um caminho a ser seguindo, se decide um novo centro e se repete até encontrar o item, algoritmos desse gênero tem um tempo na ordem de nLogn. Outra possibilidade é utilização de uma tabela hash que busca segmentar os dados e criar atalhos para cada um desses segmentos, sendo que dentro dessa lista de atalhos a busca é sequencial.

1.1 Indices

Uma dessas opções é o uso de uma estrutura em árvore, esse algoritmo pode ser aplicado sobre um conjunto de dados ordenados e ele se baseia na idéia da bissecção, onde a partir de um nó inicial, se decide se o valor procurado se encontra a direita ou à esquerda, escolhendo um caminho a ser seguindo, se decide um novo centro e se repete até encontrar o item, algoritmos desse gênero tem um tempo na ordem de nLogn. Outra possibilidade é utilização de uma tabela hash que busca segmentar os dados e criar atalhos para cada um desses segmentos, sendo que dentro dessa lista de atalhos a busca é sequencial. O uso de índices é aplicado quando desejamos aplicar tais opções. Um índice é uma estrutura no seu banco que pode ser utilizada para a realização de buscas. É necessário levar em consideração que o uso de índices não é indicado para todos os casos e que quando utilizado de forma inadequada, um índice

pode causar o efeito inverso desejado, onde ele passará a afetar negativamente a performance do banco. Seu uso é utilizado normalmente em queries que utilizam o where.

1.2 Explains

Para observar o uso dos índices assim como outros comportamentos do seu banco de dados, precisamos observar como o banco de dados decide realizar uma pesquisa. O seu sofisticado sistema de busca é capaz de analisar e decidir qual a melhor estratégia para uma dada consulta, utilizando a palavra reservada explain antes de uma query, podemos observar o plano de consulta que o banco decidiu ser o melhor para a consulta em questão. É possível que para uma mesma consulta sobre a mesma base dados o banco tenha um plano de consulta diferente, isso pode ocorrer por diversos motivos, um deles pode ser o tamanho do seu banco em um dado momento.
Levando em consideração, podemos observar algumas informações no retorno de uma chamada com a palavra explain. Sua estrutura é dada pelo tipo de pesquisa sendo realizada, em qual tabela, o seu custo, número de linhas e números de colunas, assim como como argumento utilizado no where, se houver um.

2 Ambiente

Os testes foram realizados utilizando um notebook com a seguinte configuração:

Processador: Quad-Core Intel® Core™ i5-8250U CPU @ 1.60GHz
Memória Ram: 8Gb @ 2400MHz
HD: Western Digital 1TB, Leitura 176 MB/s, Escrita 167MB/s
SO: Elementary OS 0.4 Loki

Os testes foram realizados utilizando-se o sgbd postgres versão 9.5.

3 Metodologia

Foram realizados testes sobre a base de dados nos correios. Os testes consistem em buscas simples e com o uso de join sem o uso de índices, utilizando índices de hash e índices utilizando btree.

3.1 Sem indice

```
-- Removal of existing index, if any
drop index cidade_codigo;
explain select * from bairro where cidade_codigo = 16;
```

O código acima gera a seguinte saída:

```
                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on bairro (cost=0.00..560.89 rows=43 width=24)
   Filter: (cidade_codigo = 16)
(2 rows)
```

Como podemos ver, temos uma busca sequencial, como descrito pelo **seq scan** sendo realizado na tabela **bairro** com um custo de inicial de **0.00** e um tempo total de **488.71** unidades de tempo.
Temos então a próxima querie conforme código abaixo.

```
explain select * from bairro left join cidade on
    (cidade.cidade_codigo = bairro.cidade_codigo);
```

O código acima gera a seguinte saída:

```
                              QUERY PLAN
-----------------------------------------------------------------------
 Hash Left Join (cost=300.24..1185.92 rows=28871 width=53)
   Hash Cond: (bairro.cidade_codigo = cidade.cidade_codigo)
   ->  Seq Scan on bairro (cost=0.00..488.71 rows=28871 width=24)
   ->  Hash  (cost=175.66..175.66 rows=9966 width=29)
         ->  Seq Scan on cidade (cost=0.00..175.66 rows=9966 width=29)
(5 rows)
```

Invertendo a ordem do **join** temos:

```
explain select * from cidade left join bairro on
    (cidade.cidade_codigo = bairro.cidade_codigo);
```

O código acima gera a seguinte saída:

```
                              QUERY PLAN
-----------------------------------------------------------------------
 Hash Right Join (cost=300.24..1185.92 rows=28871 width=53)
   Hash Cond: (bairro.cidade_codigo = cidade.cidade_codigo)
   ->  Seq Scan on bairro (cost=0.00..488.71 rows=28871 width=24)
```

```
   ->  Hash  (cost=175.66..175.66 rows=9966 width=29)
         ->  Seq Scan on cidade (cost=0.00..175.66 rows=9966 width=29)
(5 rows)
```

Comparando os custos, podemos observar que o custo inicial é o mesmo para ambas, pois o banco realizou ambas operações da mesma forma. Ao invés de fazer um **left join** inverso no segundo teste, o banco decidiu por um **right join** obtendo o assim o mesmo resultado.
Esse comportamento deixa claro como os sistemas internos do banco de dados é capaz de tomar decisões e alterar certas coisas, desde que isso não influencie no resultado obtido, outra coisa que pode ser observada é que mesmo sem a existência de um indice, o banco realiza a consulta do join utilizando um indice em hash.

3.2 Indíce Hash

Com a criação de uma forma de indice hash, temos os resultados abaixo.

```
--
create index cidade_codigo on bairro using hash(cidade_codigo);
explain select * from bairro where cidade_codigo = 16;
```

O código acima gera a seguinte saída:

```
                              QUERY PLAN
-----------------------------------------------------------------------------
 Bitmap Heap Scan on bairro (cost=4.33..109.20 rows=43 width=24)
   Recheck Cond: (cidade_codigo = 16)
   ->  Bitmap Index Scan on cidade_codigo (cost=0.00..4.32 rows=43
       width=0)
         Index Cond: (cidade_codigo = 16)
(4 rows)
```

Temos no resultado acima uma demonstração da habilidade dos indices de se adaptar, realizando a pesquisa utilizando um hash via **Bitmap Heap** tendo um speedup de **5.13x**.

```
explain select * from bairro left join cidade on
    (cidade.cidade_codigo = bairro.cidade_codigo);
```

O código acima gera a seguinte saída:

```
                              QUERY PLAN
-----------------------------------------------------------------------
```

```
 Hash Left Join (cost=300.24..1185.92 rows=28871 width=53)
   Hash Cond: (bairro.cidade_codigo = cidade.cidade_codigo)
   ->  Seq Scan on bairro (cost=0.00..488.71 rows=28871 width=24)
   ->  Hash  (cost=175.66..175.66 rows=9966 width=29)
         ->  Seq Scan on cidade (cost=0.00..175.66 rows=9966 width=29)
(5 rows)
```

Fica aparente que o uso de hash, não altera em nada o resultado para a consulta acima, pois o banco, ao trabalhar com um join, fez o uso de um hash mesmo assim.

3.3 Indíce Btree

Apesar do uso de um indice hash ter gerado resultados satisfatórios, o padrão atual para criação de itens como uma regra geral é realizado através de uma btree, uma estrutura de arvore sofisticada.

```
--
create index cidade_codigo on bairro using btree(cidade_codigo);
explain select * from bairro where cidade_codigo = 16;
```

O código acima gera a seguinte saída:

```
                                QUERY PLAN
-------------------------------------------------------------------------------
 Index Scan using cidade_codigo on bairro (cost=0.29..83.03 rows=43
    width=24)
   Index Cond: (cidade_codigo = 16)
(2 rows)
```

Ao compararmos o custo de da consulta acima, com o uso de hash temos uma melhoria passando de **5.13x** para **6.67x** de speedup.
Ao realizarmos uma consulta um pouco mais complexa novamente, temos os seguintes resultados.

```
explain select * from bairro left join cidade on
    (cidade.cidade_codigo = bairro.cidade_codigo);
```

O código acima gera a seguinte saída:

```
                              QUERY PLAN
-----------------------------------------------------------------------
 Hash Left Join (cost=300.24..1185.92 rows=28871 width=53)
   Hash Cond: (bairro.cidade_codigo = cidade.cidade_codigo)
```

```
   ->  Seq Scan on bairro (cost=0.00..488.71 rows=28871 width=24)
   ->  Hash  (cost=175.66..175.66 rows=9966 width=29)
         ->  Seq Scan on cidade (cost=0.00..175.66 rows=9966 width=29)
(5 rows)
```

Novamente, temos um caso onde o uso de um **join** causa um comportamento diferente do esperado. Observando a consulta, vemos que o primeiro passo realiza uma consulta usando hash, que passa para um segundo nó, utilizando utilizando uma busca sequencial e enquato passando para um segundo hash com busca sequencial em cidade.