Trabalhando com Índices no PostgreSQL

Neste artigo veremos uma abordagem quanto a utilização de índices no banco de dados PostgreSQL 9.4. apresentando estruturas básicas de sua utilização e casos onde é necessário a sua utilização.

Ao trabalharmos com bancos de dados, temos a necessidade de apresentar resultados com tamanha eficiência e rapidez, no entanto, chega um determinado momento em que o desempenho da base de dados cai, não sendo mais satisfatório dessa forma. Eis que quando isto acontece, um recurso é bastante utilizado para a resolução desse problema, que é a utilização da indexação no banco de dados. Neste artigo, temos a intenção de apresentar uma visão geral sobre os índices, de forma a termos a apresentação de exemplos para uma melhor assimilação de seus resultados.

Índices de banco de dados

Antes de mais nada, precisamos entender o que são os índices para a base de dados, onde em geral, ele é uma estrutura de dados utilizada para melhorar o tempo de execução das consultas, ou seja, os índices são estruturas que organizam referências a localização dos dados reais das tabelas.

Quando estamos lidando com SGBD's (Sistemas Gerenciadores de Bancos de Dados), como é o caso do PostgreSQL, temos que o índice é uma "cópia" do item que desejamos combinar com uma referência à localização real dos dados. Quando realizamos buscas nas tabelas sem a utilização de índices, dependendo da quantidade de registros, podemos perceber que a busca é um pouco lenta, pois dessa forma, a pesquisa é realizada de forma sequencial. Quando dizemos que existe uma pesquisa sequencial, estamos nos referindo a uma busca linha a linha em toda a tabela (ou conjunto de tabelas) da base de dados com o intuito de obter a informação necessária. Para entendermos melhor os índices, vejamos primeiramente como os dados armazenados nas tabelas são organizados, onde existem duas formas, que são as tabelas heaps e as tabelas organizadas por índices.

Tabelas heap

Quando tratamos de tabelas heap, que é a forma padrão de armazenamento, temos que os dados são armazenados sem uma ordem particular, o que quer dizer que ao adicionarmos novos registros, estes são introduzidos sem uma reorganização dos dados existentes na tabela. Dessa forma, temos um melhor desempenho no momento de inserção de dados, mas, o resultado não é tão bom quando tentamos recuperar as informações. Isso ocorre devido ao fato dos dados não estarem estruturados em uma ordem específica. No momento em que precisamos recuperar estas informações, realizamos uma busca por cada registro presente na tabela, o que acarreta maior tempo para a obtenção da informação.

Tabelas organizadas por índice

A segunda forma de organização é conhecida por "Index-organized Tables - (IOT)", ou simplesmente, Tabelas organizadas por índices, onde esta forma de organização dos índices contém dados de todas as colunas das tabelas.

Os índices podem ser utilizados em qualquer uma das estruturas apresentadas, mas precisamos ter cuidado com a quantidade de índices por tabela, isso devido a sobrecarga durante as operações DML (INSERT, UPDATE e DELETE), onde para cada uma das operações realizadas, é necessária a atualização dos índices, o que pode levar um tempo.

Ao trabalharmos com os índices, temos a nossa disposição uma estrutura adicional de dados, que nos possibilita a pesquisar dados, otimização, junções, relacionamentos e agrupamento de informações.

Vantagens e desvantagens

Uma das vantagens quando lidamos com índices é que as pesquisas são realizadas de maneira mais rápidas na base de dados, quando estes são adicionados em campos únicos, como por exemplo, o CPF de um cliente. A desvantagem encontrada é que os dados são adicionados de forma mais lenta com base nos índices criados, principalmente quando desejamos inserir dados em duas tabelas diferentes, devido a reorganização dos índices. Contudo, é necessário tomar cuidado ao criarmos os índices, pois estes não podem ser gerados para qualquer dado, pois isso faria com que a pesquisa se tornasse mais demorada.

Tipos de índices

Com o PostgreSQL, temos a nossa disposição vários tipos de índices, sendo estes o B-tree, hash, GiST, SP-GiST e o GIN. Cada um dos tipos de índice citados usam um algoritmo diferente que são utilizados para diferentes tpos de consultas. Por padrão, o comando CREATE INDEX cria índices do tipo B-tree, sendo este o que se encaixa melhor nas situações mais cotidianas. Vejamos então os tipos de índices e como utilizá-los.

Índices B-tree

Os índices do tipo B-Tree são o padrão utilizado no momento que criamos nossos índices com a instrução CREATE INDEX. O "B" significa equilibrada (Balanced), e a ideia é que a quantidade de dados em ambos os lados da árvore seja mais ou menos o mesmo. Eles podem operar todos os tipos de dados, e também podem ser utilizados para recuperar valores nulos. Este tipo de índice é projetado para trabalhar muito bem com cache. Um exemplo de sua utilização pode ser visto da seguinte forma:

```
CREATE INDEX ON idx_aluno Alunos (codAluno);
```

Índices Hash

Com relação aos índices hash, estes são úteis apenas para comparações de igualdade. No entanto, este não é um tipo que ofereça transações seguras, sendo assim, é melhor que elas sejam evitadas, além disso, elas precisam ser reconstruídas de forma manual após acidentes. Vejamos um simples exemplo de sua utilização, como o apresentado a seguir:

```
CREATE INDEX idx_aluno ON Alunos USING hash (codAluno);
```

Índices GIN

Os índices do tipo GIN (Generalized Inverted Indexes) são bastante úteis no momento em que um índice deve mapear vários valores para uma linha, o que difere dos índices B-Tree que são otimizados para quando uma linha possui um único valor de chave. Os GIN's são bons para os valores de indexação de matrizes, bem como para a aplicação de pesquisa de textos completos. Vejamos como seria a sua utilização a seguir:

```
CREATE INDEX busca_aluno_idx ON Alunos USING gin (nome gin_trgm_ops, email gin_trgm_ops);
```

Percebam que neste caso temos a utilização do gin_trgm_ops, que é utilizado para dizer ao Postgres usar trigramas utilizando as colunas selecionadas. Uma trigrama é uma estrutura de dados que armazena 3 letras de uma palavra. Com base nisso, o Postgres irá "quebrar" cada coluna de texto em trigramas e em seguida, irá usar isso nos índices quando realizarmos as pesquisas.

Índices GIST

Os índices do tipo GIST (Generalized Search Tree), nos permitem construir estruturas de árvores equilibradas, e podem ser utilizadas para operações mais avançadas que as comparações de igualdade. Eles são utilizados para indexar os tipos de dados geométricos, bem como pesquisas por textos completos. A sua criação seria de acordo com a seguinte expressão:

```
CREATE INDEX busca_aluno_cep_idx ON Alunos USING gist (cep gin_trgm_ops);
```

Índices concorrentes

No momento da construção dos índices, a tabela é bloqueada automaticamente para instruções de inserção na tabela, até que o índice seja construído. No entanto, temos que ter em mente que a criação de índices para as tabelas é uma operação cara, e em caso de ser criado um índice em uma tabela de tamanho relativamente grande, os índices podem levar muito tempo para serem criados. Isso pode causar alguma dificuldade no que diz respeito à realização de quaisquer operações de gravação, o que para ser resolvido, temos a nossa disposição no Postgres, a opção de criarmos índices concorrentes, os quais são úteis no momento em que precisamos criar índices nos bancos de dados em produção. Para este tipo de índice, temos apresentada a sintaxe a seguir:

```
CREATE INDEX CONCURRENTLY index_name ON table_name using btree (column);
```

Single-column index

O índice de coluna única basicamente é utilizado quando uma tabela representa principalmente uma única categoria de dados, ou mesmo consultas que abrangem apenas uma única categoria na tabela. Normalmente, em um projeto de banco de dados, as tabelas representam uma única categoria de dados, devido a isso, normalmente é utilizada um índice de única coluna. A sintaxe para este tipo de índice é o seguinte:

```
CREATE INDEX index_name ON table_name (column);
```

Como simples exemplo, vejamos a seguir como seria a criação de um índice para a nossa tabela de produtos:

```
SELECT COUNT(*) FROM produtos WHERE codigo_produto = 320;
```

Como podemos observar a seguir na **Figura 1**, o registro que nos importa é o que contenha o código de produto igual a 320, o qual devido a não termos um índice definido, levou um tempo de 32 ms para ser encontrado.

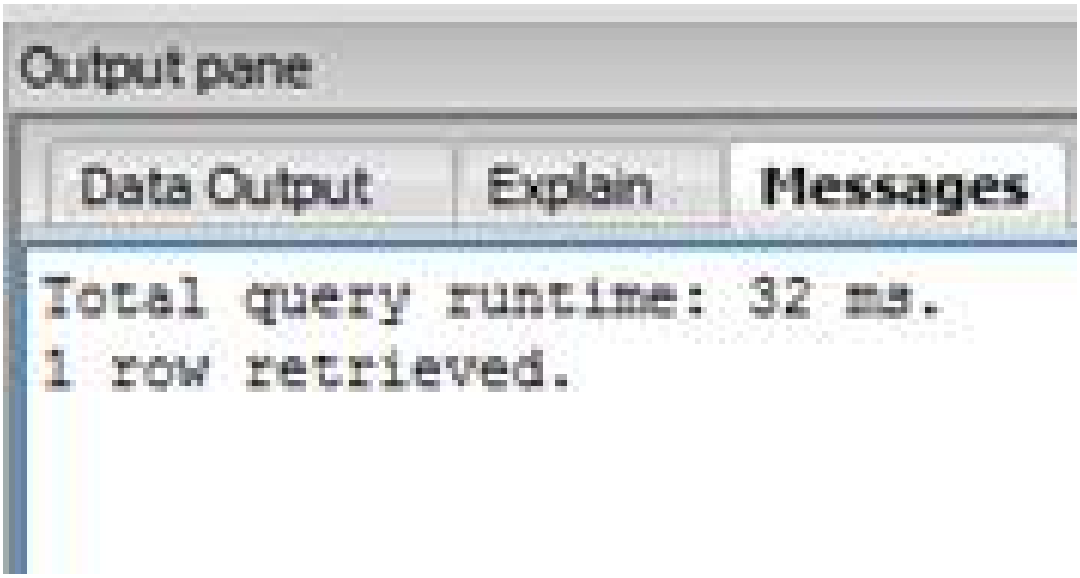

Figura 1. Consulta sem index.

Caso não tenhamos um índice definido, então teremos uma varredura completa na tabela, onde esta será uma operação dispendiosa devido a busca dos registros em questão. Como podemos perceber, apenas uma coluna foi utilizada junto a cláusula WHERE, de forma a termos um índice em uma única coluna, que é o codigo_produto, no caso de nossa consulta. Com o índice, temos que a nossa consulta passa a ser mais otimizada, como podemos ver a seguinte instrução para a criação do índice da nossa consulta:

```
CREATE INDEX produtos_index ON produtos (codigo_produto);
```

Neste momento, criamos um índice Btree, que chamamos de produtos_index, presente na coluna codigo_produto da tabela "produtos". Com essa modificação realizada, tentaremos realizar a consulta novamente para que possamos ver o tempo que será gasto com essa operação, como podemos ver na **Figura 2**.

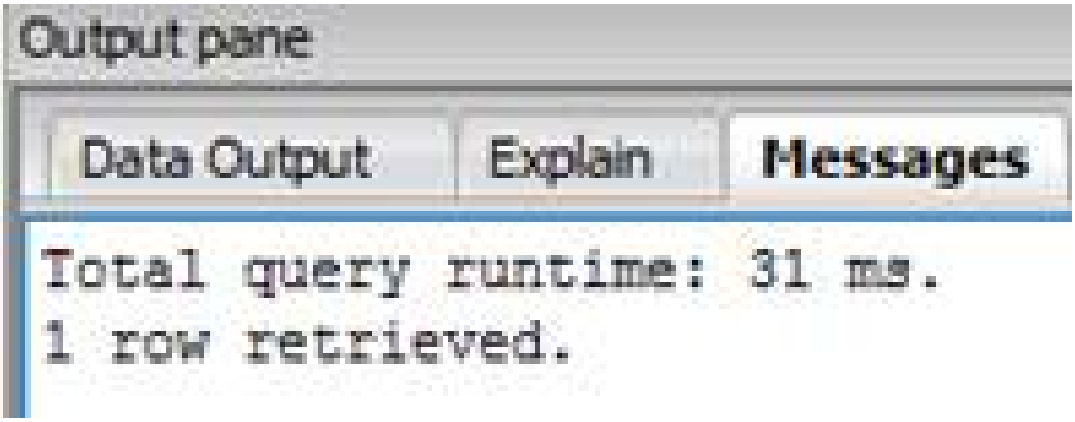

Figura 2. Resultado da consulta com índice.

Como podemos ver, o resultado da consulta com o índice levou 1 ms a menos, sendo assim um pouco mais rápida em relação a utilização das consultas sem índices. A pouca diferença ocorre devido ao fato de termos poucos registros na base de dados, mas considerem uma base em que tenhamos milhares de registros e percebam que o ganho em tempo de consulta se torna muito mais eficiente.

Índices com várias colunas (multicolumn index)

Muitas vezes, uma consulta nas tabelas de um banco de dados envolve múltiplas colunas de dados para que seja apresentada a informação. Nestes casos, índices single-columns não oferecem um bom desempenho. Devido a esse impasse, torna-se necessário que tenhamos a nosso favor índices de múltiplas colunas, o qual é suportado pelo PostgreSQL, sendo representado pela seguinte sintaxe:

```
CREATE INDEX index_name ON table_name (column1, column2);
```

Para que possamos ver melhor a utilização do índice de múltiplas colunas para otimizarmos as consultas, iremos realizar a consulta onde obteremos o número total de registros cujo codigo_produto será inferior a 400 e o preço do produto será abaixo de 300. Mas antes de realizarmos esta operação, façamos uma consulta simples sem a utilização do índice na tabela para vermos o tempo necessário para a realização da consulta, como podemos ver a seguir:

```
array11
```

Como podemos observar na **Figura 3**, obtivemos um tempo de resposta de 13 ms.

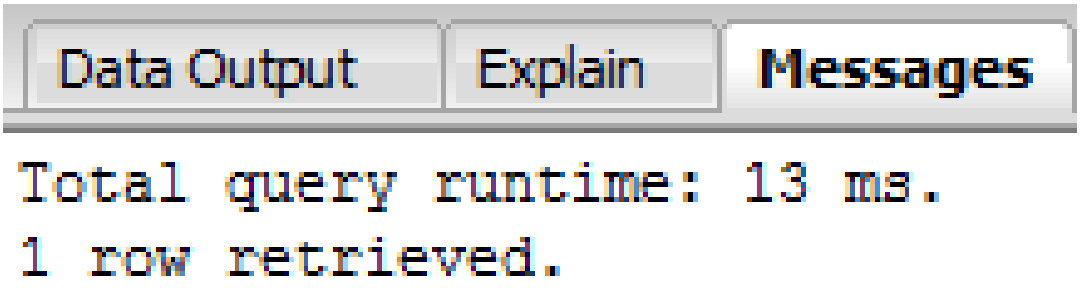

Figura 3. Consulta com múltiplas colunas sem index.

Vejamos agora a criação do índice composto pelas colunas de código_produto e preco para a realização da consulta. O índice será definido da seguinte forma:

```
CREATE INDEX produtos_multicolumns_index ON produtos (codigo_produto, preco);
```

E agora, reexecutemos a consulta, como podemos ver a seguir a seguinte declaração:

```
array11
```

Novamente, obtivemos um resultado mais rápido, onde foi contabilizado um tempo de execução de 12 ms com a utilização do índice, como podemos ver na **Figura 4**.

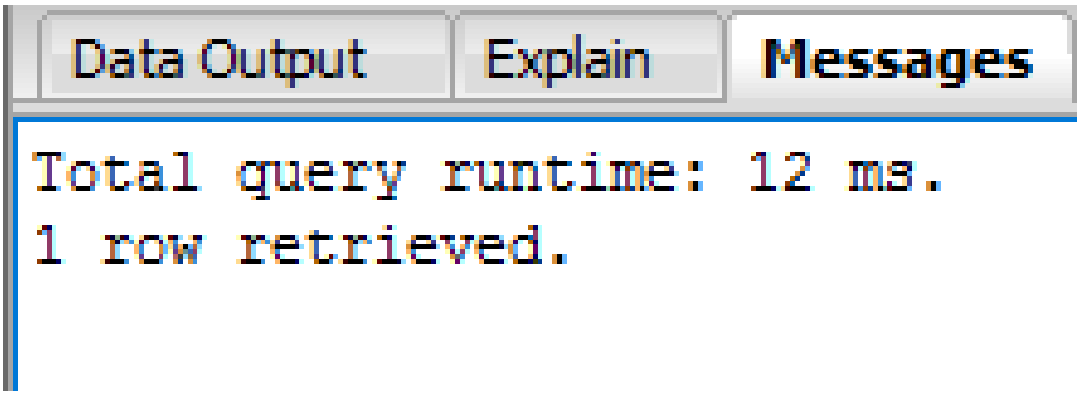

Figura 4. Index utilizando múltiplas colunas.

Para darmos continuidade ao nosso propósito de estudarmos com relação a utilização dos índices, iremos criar uma nova base de dados, a qual deve ser chamada de dvdrental. Esta é uma base de dados de exemplo, disponibilizada para testes com o PostgreSQL, onde o link para download está disponível no fim do artigo.

Criando nova base de dados

Para criarmos a base de dados dvdrental, utilizaremos o seguinte comando:

```
CREATE DATABASE dvdrental;
```

Em seguida, quando tivermos finalizado o download e criado a base de dados, precisaremos adicionar os registros às tabelas. Para isso, descompactaremos o arquivo zip, onde teremos um novo arquivo do tipo .Tar. Após descompactar, voltemos ao pgadmin III, e clique com o botão direito na base de dados que acabamos de criar. Dentre as opções apresentadas, selecione a opção "restore", como mostra a **Figura 5**.

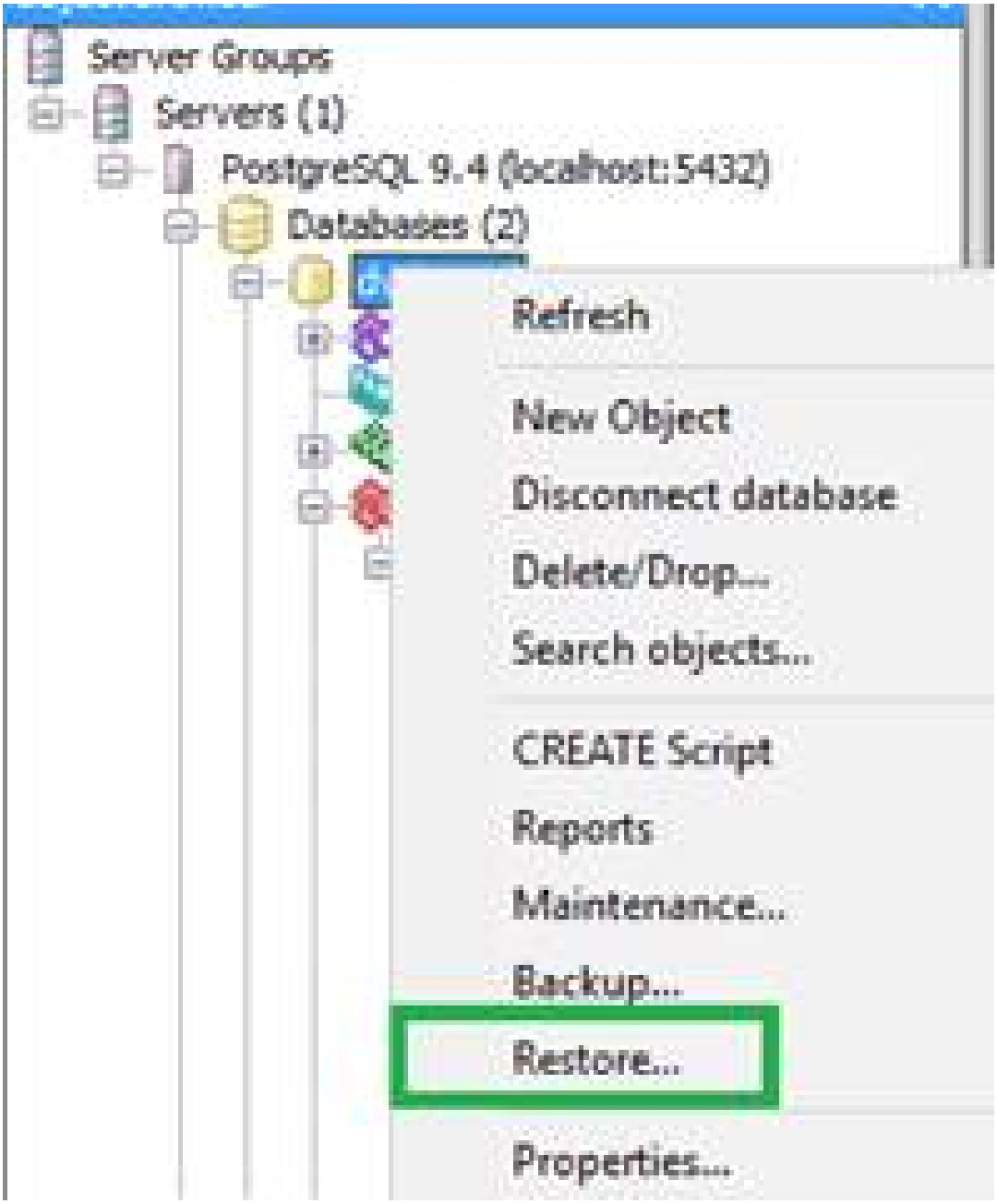

Figura 5. Restaurando uma base de dados.

Após selecionarmos esta opção, um widget será aberto para que especifiquemos o caminho de onde o arquivo .Tar está armazenado, como mostra a **Figura 6**. Por último, clique em "Restore", para que a base de exemplo seja preenchida com os dados das tabelas.

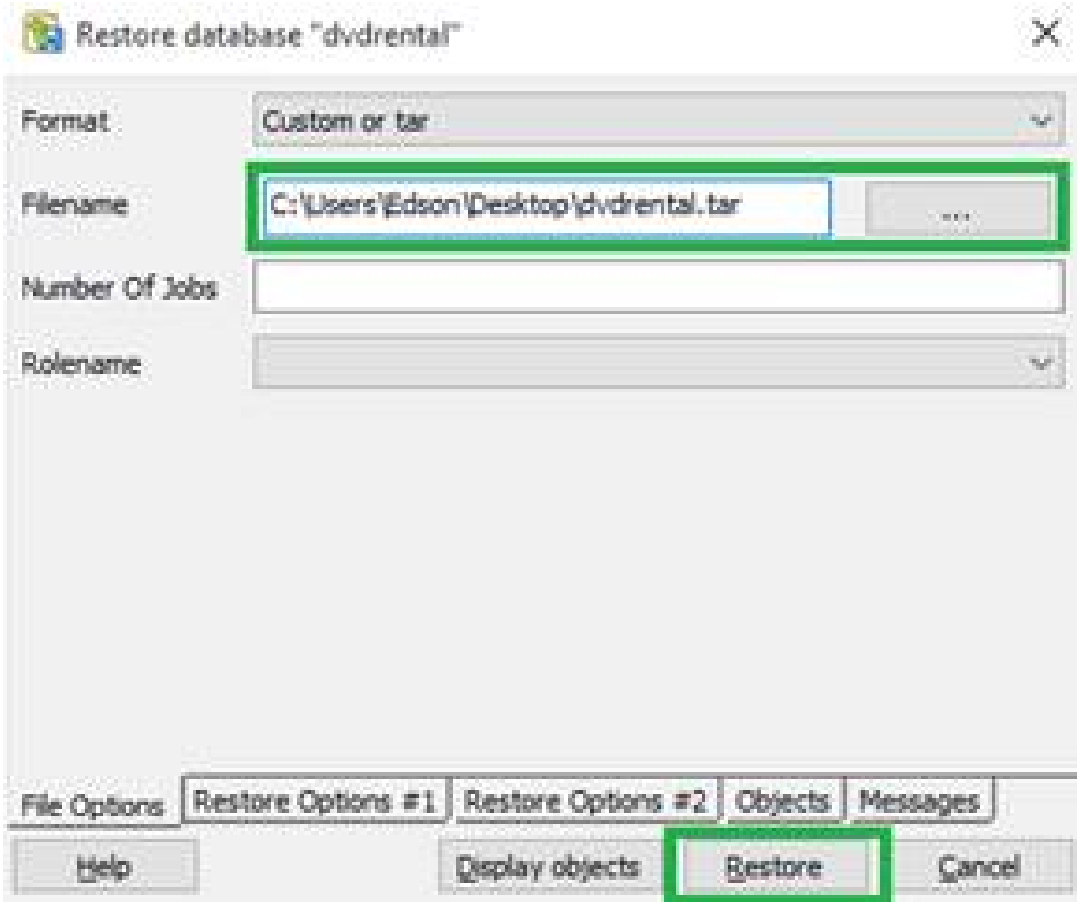

Figura 6. Selecionando arquivo .Tar.

Unique index (Indice unitário)

Com relação aos índices exclusivos, ou únicos, estes podem ser criados para qualquer coluna pertencente a tabela, pois eles não apenas criam índices, mas sim, reforçam com relação a exclusividade da coluna. A utilização deste tipo de índice se torna vantajoso por questões de integridade dos dados e também por questão de desempenho, o que mostra que pesquisas contendo índices exclusivos sejam muito mais rápidas. Para que possamos criar um índice exclusivo, temos várias maneiras, onde uma delas é utilizando o comando CREATE INDEX UNIQUE, criando uma restrição exclusiva na tabela, ou também podendo ser criado como uma chave primária. A seguir, apresentamos como podemos criar um exemplo de índice exclusivo utilizando o UNIQUE para a tabela customers, presente na base de dados dvdrental:

```
CREATE UNIQUE INDEX customer_unique_index ON customer (customer_id);
```

A forma que apresentamos acima, é uma forma explicita de criarmos um índice exclusivo, onde utilizamos a palavra-chave UNIQUE, mas podendo ser criado também de forma implícita apenas declarando uma chave primária para a tabela. Aqui está um exemplo de uma criação implícita de um índice exclusivo, criando uma chave primária para a tabela. Para demonstrarmos a criação do índice de forma implícita, podemos ver de acordo com a instrução a seguir:

```
ALTER TABLE customer ADD CONSTRAINT primary_key UNIQUE (customer_unique);
```

Com base na instrução anterior, estamos alterando a tabela e adicionando a ela uma restrição exclusiva na coluna customer_id da tabela customer, e esta declaração também cria implicitamente um índice exclusivo. O comando ALTER acrescenta uma restrição única para a coluna customer_id, podendo ser utilizado como uma chave primária.

Expression Index

Os índices de expressão são úteis para consultas que correspondam a alguma função ou modificação dos nossos dados na tabela da base de dados. Dessa forma, temos que o Postgres nos permite indexar os resultados desta função para que as pesquisas sejam mais eficientes, o que por exemplo, pode ocorrer em momentos nos quais temos a intenção de pesquisar pelo nome do cliente, onde a forma padrão de realizarmos esta operação é a seguinte:

```
SELECT * FROM customer WHERE LOWER(first_name) LIKE 'kimberly';
```

Com base na consulta anterior, realizamos uma varredura em cada linha da tabela, realizando a conversão do primeiro nome para minúscula e em seguida, sendo comparado com "kimberly". Vejamos então a criação de uma expression. Index para criar um índice na coluna first_name:

```
CREATE INDEX customer_expression_index ON customer (LOWER(first_name));
```

Com base no índice criado, será realizada uma busca na tabela pelos clientes com base no primeiro nome registrado. Como o nome pode ter sido armazenado com letras maiúsculas e minúsculas, forçamos no índice a busca pelos nomes sempre minúsculos, o que para isso, é necessário a utilização dos índices de expressão. Um índice de expressão é utilizado apenas quando a expressão exata é utilizada em uma consulta.

Índices parciais (Partial Indexes)

Quando tratamos de índices parciais, temos que este tipo de índice abrange apenas um subconjunto de dados pertencentes a tabela, onde está tem a declaração de uma cláusula WHERE. Com isso, temos um aumento na eficiência dos índices, pois reduzimos o tamanho do conjunto de dados a serem pesquisados. Como simples exemplo, podemos ver a criação do índice:

```
CREATE INDEX idx_fk_category_id ON film_category USING btree (film_id) WHERE category_id = 13;
```

Como pode ser visto na instrução acima, temos a criação de um índice que traz um grupo reduzido de informações com base no código da categoria informado, devido a utilização da condição WHERE.

Gerenciando e mantendo índices

Uma das questões que nós, administradores de bancos de dados, precisamos ter em mente é em como devemos lidar com o chamado índice de inchaço, ou Index bloat, nas tabelas do banco de dados PostgreSQL. Neste ponto, temos a nossa disposição que a arquitetura MVCC (Multi-Version Concurrency Control) do PostgreSQL apresenta uma segurança a mais no que diz respeito ao monitoramento e manutenção das bases de dados, especialmente em sistemas com grande quantidade de registros. Mas o que é o MVCC? Quando falamos de MVCC, estamos nos referindo a um método do PostgreSQL utilizado para lidar com a consistência dos dados quando vários processos estão acessando a mesma tabela.

Por que ocorrem os inchaços?

O MVCC foi escolhido para lidar com as múltiplas transações e sessões no PostgreSQL, onde estas ocorrem nas mesmas linhas quase que ao mesmo tempo. Devido a isso, temos como resultado de uma parte específica do MVCC, a ocorrência dos "inchaços", que é concentrada nas manipulações de exclusão e atualização.

No momento que realizamos a exclusão de uma linha, ela não é realmente apagada, mas sim, marcada como indisponível para futuras transações que ocorrem após a exclusão, o que também acontece no momento da atualização dos registros, onde a linha "antiga" é mantida ativa até que todas as operações sejam finalizadas, tornando-a indisponível para qualquer outra operação. Em seguida, temos o processo de VACUUM que marca as linhas indisponíveis como sendo um espaço útil para a inserção de novos registros ou para atualizações futuras. No entanto, várias são as razões para a ocorrência dos inchaços, o que precisamos corrigir para termos mais desempenho na base de dados. De certo, o maior causador dos inchaços nas tabelas é o VACUUM, mas ele é um parâmetro que pode ser configurado, sendo possível tanto a inativação quanto uma configuração errada. Tendo

esse ponto em vista, vejamos as maneiras possíveis de corrigirmos os índices de inchaço.

Realização de dumps e restauração

A maneira mais simples de prevenir os inchaços é realizando backups das tabelas utilizando o comando pg_dump, onde ele exclui a tabela e em seguida, recarrega os dados para tabela novamente, porém, esta é uma operação cara.

Utilização do VACUUM

Esta opção de comando "devolve" o espaço em disco para o sistema de arquivos, sendo que isto é feito em casos muito específicos. O espaço utilizado está contido em arquivos de páginas que compõem as tabelas e índices no Postgres. As páginas dos arquivos possuem um mesmo tamanho, mas objetos de tamanhos diferentes. No momento da utilização do VACUUM, acontece a marcação para cada linha em um arquivo de página como sendo indisponível, onde o espaço em disco é devolvido ao sistema de arquivos. A sintaxe do VACUUM é a seguinte:

```
VACUUM table_name
```

Ou

```
VACUUM FULL table_name
```

No momento em que utilizamos o comando VACUUM com a flag FULL, teremos que todo o espaço reutilizável será devolvido ao sistema de arquivos, com a diferença de que dessa forma, ele reescreve completamente a tabela para novas páginas de arquivos. Um exemplo de sua utilização pode ser visto a seguir na tabela rental:

```
VACUUM FULL rental;
```

Utilização de CLUSTERS

Outra maneira de gerenciarmos o inchaço das tabelas é com a utilização do comando CLUSTER, o qual é utilizado para reordenar fisicamente as linhas com base no índice. Quando utilizamos o comando CLUSTER, estamos criando na verdade uma cópia inicial de toda a tabela, onde a anterior é então descartada. Para utilizarmos o comando CLUSTER, é necessário que haja espaço suficiente em disco, para que a cópia inicial dos dados seja mantida enquanto a cópia é criada.a sintaxe básica de sua utilização é a seguinte:

```
CLUSTER table_name USING index_name
```

Utilizando a reindexação

Por fim, temos a reindexação, onde quando um índice torna-se ineficiente devido ao inchaço, a utilização da reindexação passa a ser uma opção favorável para a obtenção de um máximo desempenho dos índices. a sintaxe utilizada para esse caso é a seguinte:

```
REINDEX TABLE payment;
```

Como podemos ver, os índices são uma maneira comum para a melhoria do desempenho de um banco de dados, onde eles permitem que os servidores de banco de dados encontrem e recuperem linhas específicas de forma muito mais rápida do que sem a utilização dos índices. Ainda assim, os índices adicionam ao sistema uma sobrecarga no banco de dados como um todo, o que implica que devemos utilizá-los de forma sensata. Com isso finalizamos mais este artigo, esperamos que tenham gostado. Até a próxima! =)

Links

Site oficial do PostgreSQL
http://www.postgresql.org/docs/9.4/static/sql-createindex.html

Base de dados exemplo
http://www.postgresqltutorial.com/postgresql-sample-database/

por Edson José
Banco de dados & BigData expert