Projeto de Banco de Dados

Tuning

PROF. DR. THIAGO ELIAS

Tuning (Otimização)

- Trataremos de algumas técnicas de otimização de recursos para tirar maior proveito do PostgreSQL.
- Estas técnicas vão desde simples manutenção de um BD, passando por recursos avançados de limpeza e indexação de informações, melhor utilização da linguagem SQL, entre outras técnicas.

Tuning (Otimização)

- O que é otimização de consultas?
  - Uma situação ideal seria se tanto o software quanto o hardware trabalhassem em conjunto para obter o resultado de uma ação de forma mais rápida e precisa possível.
  - O conjunto de técnicas que pode ser utilizado para chegar o mais próximo dessa situação chama-se **Otimização**.
  - Otimizar significa utilizar os recursos que se tem para que sejam utilizadas da melhor forma possível.
  - Estudaremos algumas técnicas de otimização. São elas:
    - Indexação de colunas
    - Melhores práticas do uso da linguagem SQL nas consultas.
    - Configurações Administrativas

Indexação de Colunas

- Talvez seja o método de otimização mais importante, pois é a técnica que consegue obter bons resultados no menor espaço de tempo.
- Os bancos de dados utilizam índices, para que não só consultas, mas inserções, exclusões e atualizações sejam feitas com mais agilidade.

Indexação de Colunas

- Basicamente significa manter armazenado externamente a uma tabela, uma de suas colunas ordenada. Além do valor em questão, armazena-se também uma referência ao registro ao qual seu valor diz respeito.
- Porém os índices também produzem trabalho adicional para o sistema de banco de dados como um todo. Portanto, deve-se adquirir bons conhecimentos sobre o assunto para o seu devido uso.

Indexação de Colunas

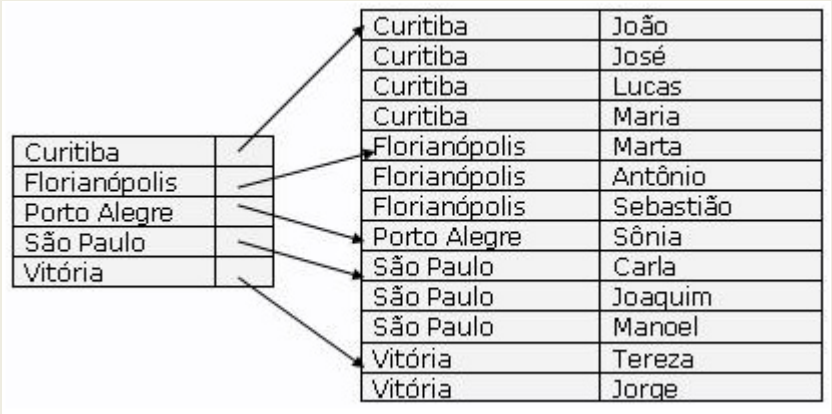
- O que torna a indexação diferente da simples ordenação de uma tabela?
  - A possibilidade de poder utilizar dois ou mais índices para uma mesma tabela.

Indexação de Colunas

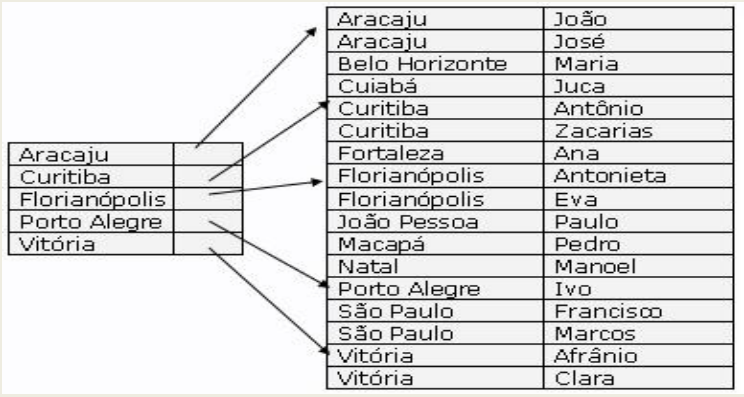
- Cada vez que um registro é inserido ou atualizado, a tabela de índices também é atualizada.
- É difícil criar regras genéricas para determinar que índices devem ser definidos.
- A experiência por parte do administrador e muita verificação experimental é necessária na maioria dos casos.

Índices Primários

- Índices ordenados sequencialmente:
  - Denso:

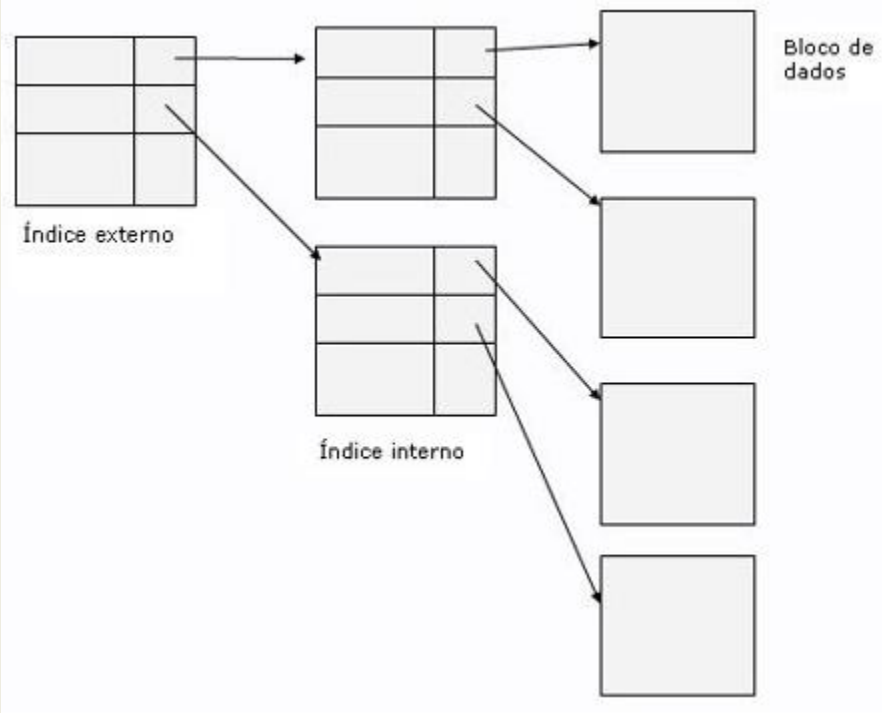


  - Esparso: encontra-se o maior valor, menor ou igual ao valor da chave de busca.



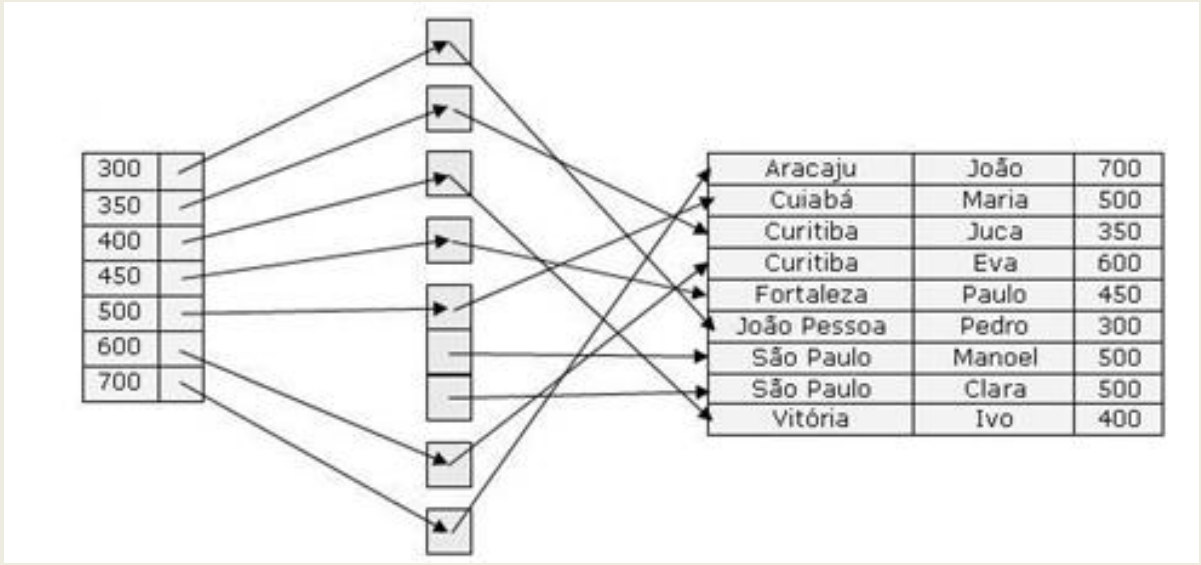
Índices Primários

- No caso de índices muito grandes, cria-se índices esparsos de índices densos



Índices Secundários

- Nos índices secundários, a chave de busca não ordena sequencialmente a tabela.
- Por conta disso, são necessárias estruturas complementares.



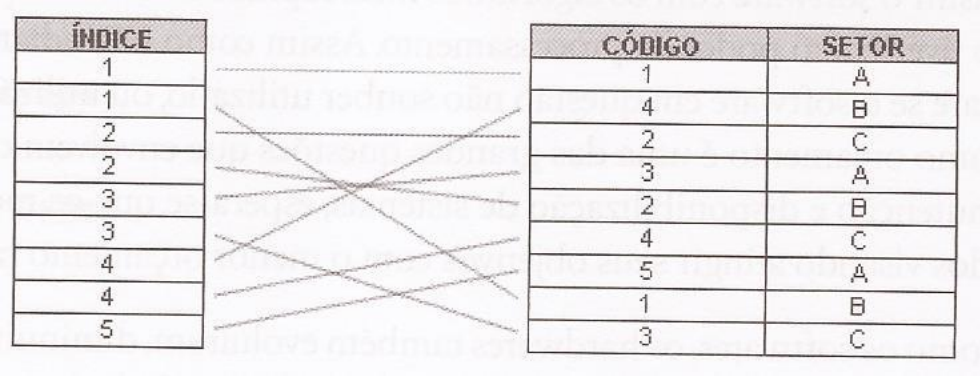
Índices no Postgresql

- B-Tree:
  - São árvores de pesquisa balanceadas. É o tipo *default*.
  - São índices que podem tratar consultas de igualdade e de faixa.
  - Indicado para consultas com os operadores: <, <=, =, >=, >. Também pode ser utilizado com LIKE.
- R-Tree:
  - Também conhecido como árvores R, utiliza o algoritmo de partição quadrática de Guttman, sendo utilizada para indexar estrutura de dados multidimensionais, cuja implementação está limitada a dados com até 8Kbytes, sendo bastante limitada para dados geográficos reais. Utilizado normalmente com dados do tipo box, circle, point e outros.
- Hash:
  - A documentação do Postgresql desencoraja a utilização desta estrutura.
- GiST
  - Árvores de Procura de Índice Generalizadas

Índices no Postgresql

- Sintaxe:
  - CREATE INDEX nome ON tabela USING tipo (campo);
    - tipo: BTREE, RTREE, HASH, GIST
  - Obs: Somente os tipos B-tree e GiST suportam índices com várias colunas.
  - Índices com mais de um campo somente será utilizado se as cláusulas com os campos indexados forem ligados por AND.

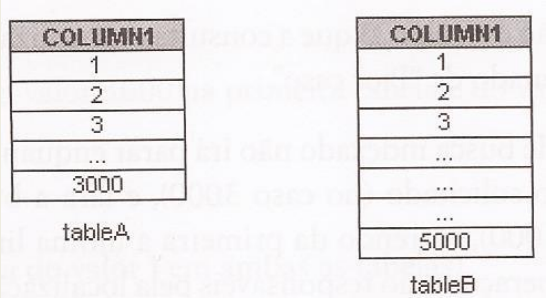
Indexação de Colunas



- Imagine uma consulta na tabela acima cujo parâmetro *where* busque registros cujo código seja igual a 2. Nesse caso o SGBD realizará 9 comparações.
- Agora imaginando a tabela indexada, o SGBD faria apenas 5 comparações.

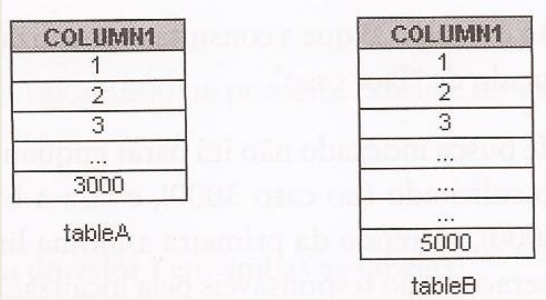
Indexação de Colunas

- A velocidade de processamento tende a apresentar resultados melhores quando as consultas em questão envolvem duas ou mais tabelas.
- Para ilustrar, considere as duas tabelas a seguir:



Indexação de Colunas

- Considere a seguinte consulta:
- Select * from tableA inner join tableB on tableA.column1=tableB.column1 where tableA.column1=23



- Sem a indexação, serão gerados 15.000.000 operações (3000 x 5000), independente do valor procurado ser 23.

Indexação de Colunas

- Considerando a primeira tabela indexada, a consulta encerraria assim que o valor encontrado na primeira tabela fosse superior a 23.
- Assim, o SGBD realizaria 115.001((23 x 5000) +1) operações. Bem menos que os 15.000.000 anteriores.
- No caso do valor buscado ser 3000, ocorreria o *pior caso*.
- No caso do valor buscado ser 1, ocorreria o *melhor caso*, onde apenas 5001 operações são utilizadas.

Indexação de Colunas

- Ademais, a tendência de melhorar o resultado é ainda maior se as consultas em questão filtrassem mais valores em suas cláusulas where. Suponha a seguintes consulta:
  - SELECT * FROM tableA INNER JOIN tableB ON tableA.column1=tableB.column1 WHERE tableA.column1=23 AND tableB.column1=23
- Estando as duas tabelas indexadas, teríamos 531 operações (23 x 23 + 1 + 1)

Considerações Importantes Sobre Indexação

- A utilização de indexação traz igualmente algumas desvantagens ao banco de dados, as quais, porém, se espera que sejam bem menores que as vantagens obtidas. São elas:
  - *Tempos de execução em comandos INSERT, DELETE e UPDATE*: Ao se manipular uma tabela, todos os índices associados àquela tabela também devem ser manipulados. Quanto mais índices possuir uma tabela, mais demorada será a execução de uma operação insert, delete ou update.
  - *Espaço em disco utilizado*: para armazenar as listas de índices externamente às tabelas, há redundância de dados. Recomenda-se também usar tipos de dados adequados.

Quando Utilizar Indexação

- A indexação deve ser utilizada preferencialmente quando a coluna em questão for usada como parâmetro em um dos seguintes argumentos:
  - Where, Join: As consultas encerram suas buscas assim que identificam a impossibilidade de não encontrarem mais o resultado solicitado.
  - Group By, Order By:
  - Distinct
  - Funções MIN e MAX: As funções de buscas já sabem que encontrarão os mínimos valores nas primeiras linhas e os máximos valores nas últimas linhas.

Quando Utilizar Indexação

- Além disso, não é recomendado utilizar índices em colunas cuja distinção de valores seja muito pequena quando comparada ao número total de registro. Por exemplo, a coluna “Estado _civil” em uma tabela com 1000 registros.

Exemplo Prático

- Crie a tabela PESSOA com duas colunas: código (serial) e idade (inteiro).
- Crie uma função que insira 500 mil registros na tabela (use a função *random()* para gerar idades aleatórias).
- Use as expressões “EXPLAIN ANALYSE” (será detalhada posteriormente), para verificar o tempo de execução de uma consulta que busca pessoas com idade igual a 22 anos.
- Crie um índice para a coluna idade.
- Demonstre que, com a utilização do índice, a consulta foi otimizada.

Técnicas de Otimização

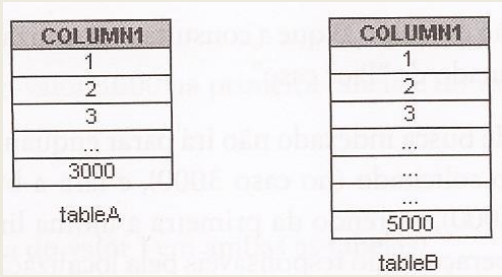
- Além de índices, existem outras práticas recomendadas que tem como objetivo executar expressões SQL mais rapidamente.
- O postgreSQL está preparado com alguns algoritmos de inteligência artificial para executar uma análise de cada expressão antes de executá-la. Por exemplo, a consulta *select * from table1 where 1=0* não será executada pois ele sabe que nenhum registro será encontrado.
- Apesar disso, nossas consultas devem ser bem estruturadas para cooperar com o servidor de BD.

Técnicas de Otimização
-Isolamento de Índices-

- O PostgreSQL utiliza melhor os índices quando os mesmos encontram-se de forma isolada nas expressões SQL.
- Considere a seguinte consulta
  - Select * from table where column1 = column2 - 1
- Supondo que a *column2* é indexada, neste caso o recurso não poderá ser utilizado, pois seria necessário recalcular todos os valores do índice novamente.
- Solução: Isolar a coluna índice
  - Select * from table where column1 + 1 = column2

Técnicas de Otimização
-Ordem de Utilização dos Índices (JOINS)-

- Considerando a figura a seguir, suponha que ambas as colunas sejam indexadas e que a seguinte expressão SQL seja solicitada:
  - Select * from tableB inner join tableA on tableB.column1=tableA.column1



- Nesse caso, o PostgreSQL utilizaria a *tableA* como referência (e não *tableB*), pois assim realizaria uma número menor de operações, uma vez que a *tableA* tem menos registros que *tableB*.

Técnicas de Otimização
-Evitar o Uso da Cláusula *IN*-

- A cláusula IN reduz o desempenho do servidor de BD. Isso porque a cláusula não executa corretamente a utilização de índices. Ex:
  - Select * from table where cod In (1,4,10)
- Solução:
  - Select * from table where cod=1 Or cod=4 Or cod = 10

Técnicas de Otimização
-Evitar o Uso de Sub-Selects-

- Outra técnica SQL que também prejudica o desempenho do servidor é a utilização de sub-selects em comandos SQL.
- Contudo, há situações em que os sub-selects são extremamente práticos. Nesses casos, devemos medir a execução com indicadores de tempo e decidir se é possível reescrever o comando de outra forma.

Técnicas de Otimização
-Estatísticas do Servidor-

- O PostgreSQL baseia-se em um conjunto de estatísticas de cada tabela para gerenciar e otimizar os comandos internos das expressões SQL.
- Assim, é recomendado manter tais estatísticas atualizadas com as novas informações das tabelas.
- Comando:
  - ANALYZE [VERBOSE] [tabela[coluna]]
- VERBOSE mostra as informações na tela.
- Caso omitidos os argumentos TABELA e COLUNA, serão considerados como todas.

Técnicas de Otimização
-Desfragmentação e Limpeza-

- Outra característica gerada tanto por tipos de dados de tamanho variável quanto por tabelas que sofrem constantes atualizações é a fragmentação do espaço em disco.
- Ela é definida como “buracos” na memória que aumenta o tempo de retorno do sistema.
- Para contornar o problema, é recomendado executar o comando VACUUM regularmente em tabelas com alta frequência de atualizações.

Técnicas de Otimização
-Desfragmentação e Limpeza-

- Sintaxe:
  - VACUUM [modo] [VERBOSE] [ANALYZE] [tabela]
- Modo:
  - FULL: além da limpeza de espaços deslocados, é realizada uma fragmentação para que a tabela ocupe um menor número de blocos no disco.
  - FREEZE: utiliza uma forma mais avançada para o tratamento de performance.
  - Não-Informado: São apenas limpos os espaços de informações deslocadas.

Técnicas de Otimização
-Desfragmentação e Limpeza-

- Sintaxe:
  - VACUUM [modo] [VERBOSE] [ANALYZE] [tabela]
- VERBOSE: são apresentadas na tela informações do processo.
- ANALYSE: faz com que cada tabela atualize o arquivo de estatística, possibilitando que o servidor, futuramente, identifique os melhores meios de executar um comando SQL.
- Obs: caso o nome da tabela seja omitido, todo o banco de dados ativo será organizado.

Técnicas de Otimização
-Análise de Expressões-

- O PostgreSQL disponibiliza um comando que possibilita analisar previamente consultas SQL (select, insert, delete, update, execute e declare), para saber como será o comportamento das mesmas.
- Isso é importante para compararmos desempenhos.
- Comando: EXPLAIN
- Sintaxe:
  - Para utilizá-lo, basta acrescentar o termo antes da consulta SQL.

Técnicas de Otimização
-Análise de Expressões-

- Um dos indicadores mais importantes é o retorno da variável ***cost***, a qual informa o custo de esforço do PostgreSQL para atingir aquele objetivo.
- Se preferir visualizar em milissegundos o tempo de execução que o comando levaria, utilize o comando EXPLAIN com a opção ANALYSE.

Técnicas de Otimização
-Análise de Expressões-

QUERY PLAN text
Seq Scan on genero (cost=0.00..1.10 rows=10 width=12) (actual time=0.011..0.013 rows=10 loops=1)
Total runtime: 0.049 ms

- Entre outras informações disponíveis, estão:
  - Rows: indica quantos registros o comando espera encontrar
  - Width: informa o tamanho médio em bytes dos registros retornados pela sintaxe.
  - Seq Scan: busca sequencial. Poderia também apresentar:
    - Index Scan: utiliza algum índice
    - Sort: alguma ordenação (order by) foi utilizada para gerar o resultado
    - Unique: utiliza estrutura de colunas unique ou consultas com DISTINCT
    - Nested Loop: utiliza um JOIN para unir duas ou mais tabelas

Técnicas de Otimização
-Normalização do Banco de Dados-

- Em um banco normalizado, o espaço ocupado no disco normalmente é o menor possível, e seu processamento tende a ser mais rápido.
- Além disso, a escrito dos comandos SQL tornam-se menores e menos complicadas, aumentando a performance do BD.

Configurações Administrativas

- Até o momento, tratamos de técnicas a nível de desenvolvedor e programador de aplicações e SQL.
- Porém, configurações do servidor também podem melhorar a sua performance.
- Há várias variáveis que permitem que o servidor seja customizado para os diferentes situações de uso.
- Todos os ajustes serão feitos nas respectivas variáveis no arquivo ***postgresql.conf*** .

Configurações Administrativas
Número máximo de conexões simultâneas

- É possível gerenciar o número máximo de conexões simultâneas.
- Variável:
  - Max_connections
- O ajuste ideal depende de outros fatores, como capacidade de processamento da CPU, total de memória disponível, taxa de tráfego e capacidade de atendimento da rede...

Configurações Administrativas
Buffers de Memória Compartilhados

- Responsável por definir o tamanho dos buffers utilizados pelo PostgreSQL.
- Variável:
  - Shared_buffers
- Em casos nos quais o servidor é dedicado para o servidor PostgreSQL, é recomendada a utilização de 15% do total da memória RAM.
- O aumento dos buffers é uma das alterações que podem trazer resultados melhores em curto espaço de tempo.
- Obs: testes mostram que valores muito altos podem prejudicar a performance.

Configurações Administrativas
Intervalo e Tempo da Gravação em Disco

- A cada transação realizada, uma função chamada de *fsync* é invocada pelo servidor, forçando a gravação dos dados que estiverem em memória para o disco.
- O grande uso do *fsync* pode afetar um pouco o desempenho do servidor, sendo possível alterar esta funcionalidade para *OFF*.
- Desvantagem:
  - Caso ocorra algum erro após a realização de uma transação e antes da mesma ser gravada em disco.
- Esta não é uma ação recomendada.

Configurações Administrativas
Realização de COMMIT de Múltiplas Transações

- É possível definir o comportamento das gravações de transações no PostgreSQL.
- O PostgreSQL pode aguardar um determinado período de tempo recebendo os COMMITs de mais de uma transação ativa, e, ao atingir determinado número de operações, gravá-las em disco de uma só vez.
- Esse tempo é definido em milissegundos na variável *commit_delay*, cujo valor máximo é 100.000 microssegundos.

Configurações Administrativas
Realização de COMMIT de Múltiplas Transações

- Obs: O aguardo de outras transações realizarem o COMMIT não ocorrerá se a função *fsync* estiver desabilitada.

Configurações Administrativas
Cache

- Assim como os browsers da web, o PostgreSQL também realiza o cache de algumas informações, para evitar dezenas de leituras em tabelas no BD.
- É possível alterar o tamanho do cache através da variável *effective_cache_size*.
- Recomenda-se usar entre 25% e 50% do total da memória RAM.