Índices

- Um índice é uma organização de dados que permite acelerar o tempo de acesso as linhas de uma tabela.

- Uma abordagem semelhante ao conceito de índices em banco de dados é o índice remissivo, utilizado em livros. O leitor pode percorrer o índice rapidamente e ir para a página desejada.

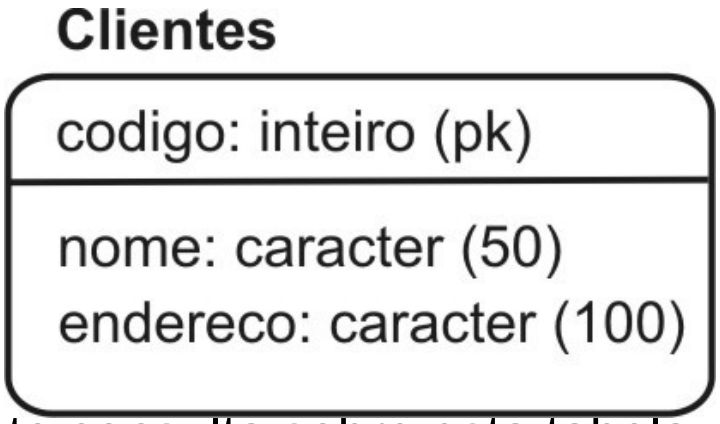
Índices

Em Bancos de dados, existem basicamente três razões para a definição de índices:

- permitir que as linhas sejam acessadas rapidamente através do valor do atributo indexado.
- facilitar a ordenação das linhas por aquele atributo.
- concernente à unicidade, quando é necessário que uma coluna seja única, um índice é criado pelo SGBD com a função de assegurar que não sejam aceitos valores duplicados.

Índices - exemplo

- Exemplo:
  - Considere a tabela:



- Considere a seguinte consulta sobre esta tabela:

```
SELECT nome FROM clientes WHERE codigo = 2;
```

Índices – exemplo (cont.)

Cenário sem índices

- o sistema precisaria varrer toda a tabela `clientes`, linha por linha, para encontrar todas as entradas correspondentes. Havendo muitas linhas em `clientes`, e poucas ou nenhuma linha retornadas pela consulta. Esse método é claramente ineficiente.

Cenário com índices:

- índice para a coluna `codigo`
- O SGBD pode utilizar um método mais eficiente para localizar as linhas correspondentes.

Tipos de Índices

Em SGBDs Relacionais, existem dois tipos básicos de índices, índices ordenados e índices *hash*.

Indices Ordenados

- cria uma estrutura de índice para cada chave de pesquisa.
- Os valores das chaves são armazenados de forma ordenada, possibilitando assim, acesso rápido aos registros associados a cada chave de pesquisa.

Tipos de Índices (cont.)

Índices Hash

- baseiam-se em distribuir uniformemente os valores de chaves para uma determinada faixa no disco de armazenamento, denominada bucket.
- Para se obter um registro, é aplicada uma função, denominada função hash sobre a chave de pesquisa, e então é obtido o bucket que contém aquele registro

Índices Ordenados

Índices ordenados Primários ou *clustering*

- o arquivo que contém os registros, está ordenado sequencialmente.
- geralmente, a chave de pesquisa de um índice primário é a própria chave primária
- consultas por períodos que acessam informações da tabela, produzem melhores resultados quando realizadas utilizando um índice primário.

Índices Ordenados (cont.)

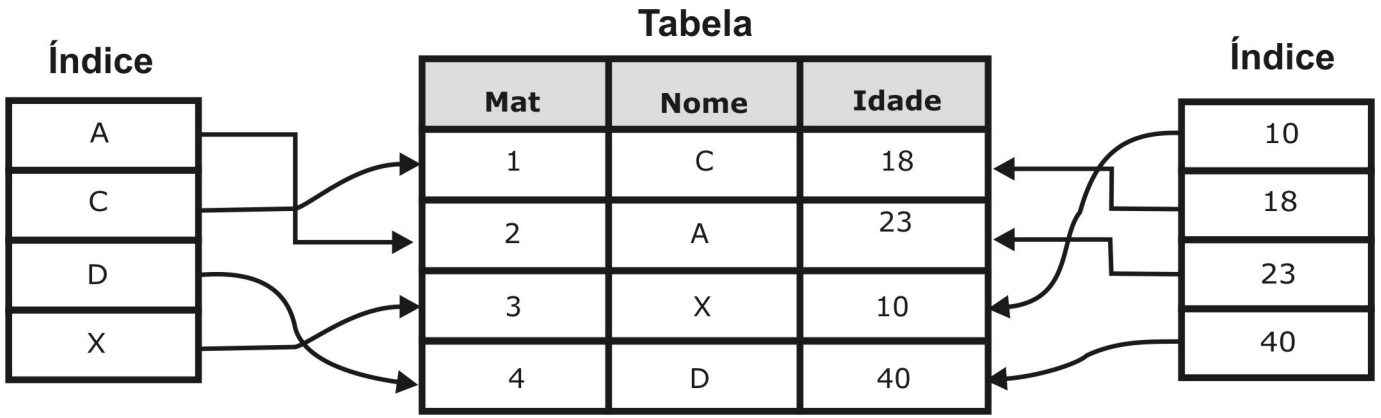
Índices ordenados secundários

- quando as chaves de pesquisa especificam uma ordem diferente da ordem seqüencial do arquivo indexado
- a ordem das chaves nas folhas do índice é diferente das linhas armazenadas na tabela.

Índices Ordenados (cont.)

Índices ordenados Densos

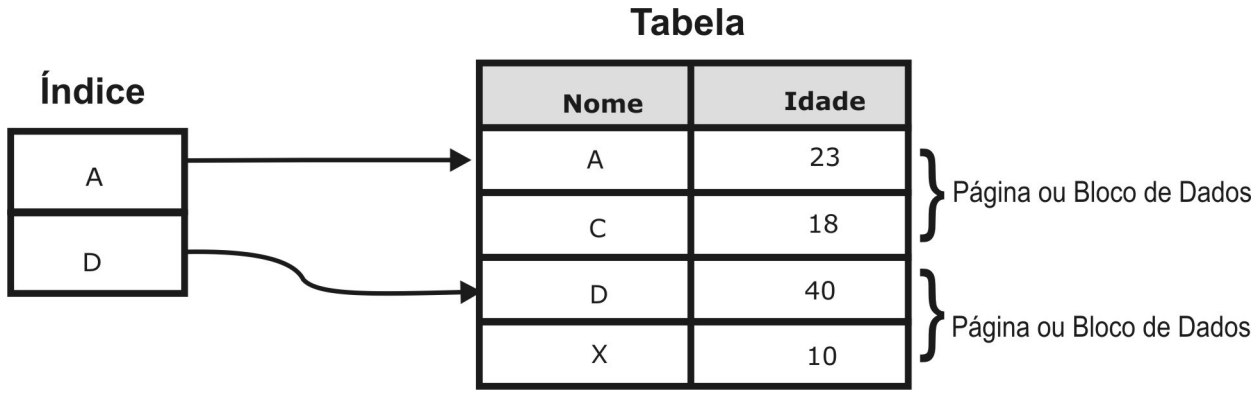
- quando um registro do índice aparece para cada registro do arquivo.
- o registro do índice contém o valor da chave de pesquisa e um ponteiro para o primeiro registro de dados com o valor da chave de pesquisa.



Índices Ordenados (cont.)

Índices ordenados esparços

- O índice esparso corresponde a um registro de índice, criado apenas para alguns dos valores.
- Assim como nos índices densos, cada registro do índice contém um valor de chave de pesquisa e um ponteiro para o primeiro registro de dados com esse valor de chave de pesquisa .



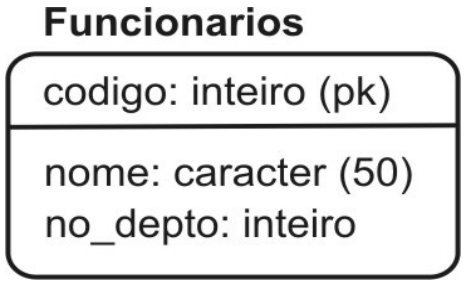
Índices Ordenados (cont.)

Índices completos e parciais

- Índices completos são aqueles que indexam todas as linhas de uma determinada tabela.
- Índices parciais indexam somente um subconjunto de linhas definidas por um predicado na criação do índice.
  - Índices parciais são úteis quando há necessidade de indexação de colunas que tenham as distribuições de dados não uniformes, pois exclui a representação de valores com grandes quantidades de repetições.

Índices com múltiplas colunas

- O índice em múltiplas colunas pode ser definido contendo mais de uma coluna.
- Exemplo:



  - Supondo que freqüentemente sejam feitas consultas SQL como:

```
SELECT nome FROM funcionarios WHERE codigo = 2 AND no_depto = 15;
```

  - É apropriado definir um índice contendo as colunas `codigo` e `no_depto`.

Índices com múltiplas colunas (cont.)

- O Otimizador do SGBD, pode utilizar um índice com várias colunas, para comandos envolvendo a coluna mais a esquerda na definição do índice, e quaisquer colunas listadas a sua direita, sem omissões.
- Exemplo:
  - um índice contendo as colunas `codigo, nome e no_depto,` pode ser utilizado em comandos, envolvendo `codigo, nome e no_depto`, ou em comandos com `codigo e nome`, ou em comandos com apenas `codigo,` mas não em outras combinações.
  - Em um comando envolvendo `codigo e no_depto`, o Otimizador pode utilizar o índice apenas para `codigo`, tratando `no_depto` como uma coluna comum não indexada.

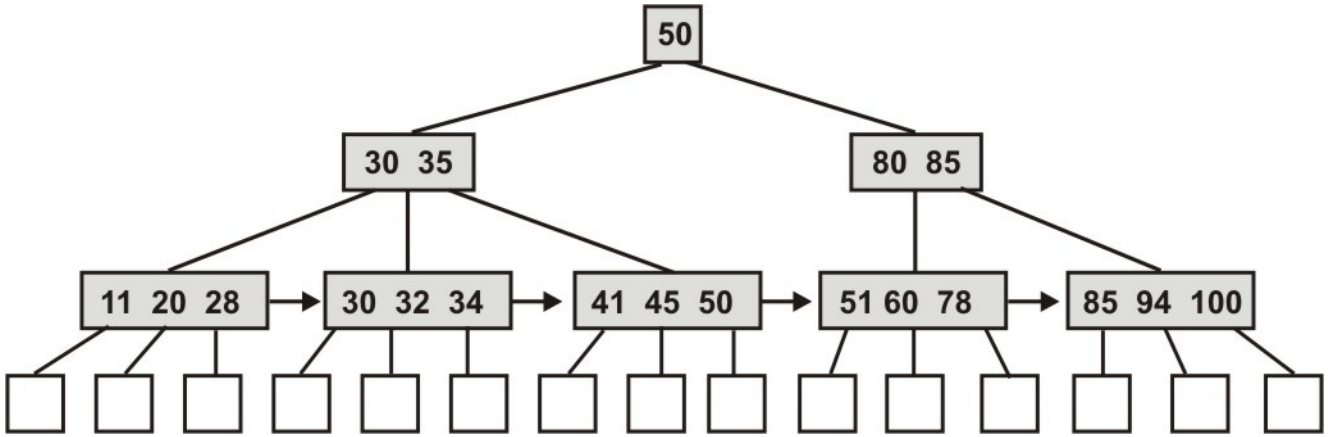
Estrutura de dados para Índices

- Uma árvore é uma estrutura de dados cujos elementos têm relacionamentos um para muitos entre si. Cada elemento tem no máximo um pai. De acordo com a terminologia padrão, cada elemento da árvore é chamado de nó, e os relacionamentos entre elementos são chamados ramos. A profundidade de um nó é a distância desse até a raiz. Um conjunto de nós, com mesma profundidade, é denominado nível da árvore e o número máximo de descendentes para cada um dos nós é denominado grau da árvore. A maior profundidade de um nó é a altura da árvore

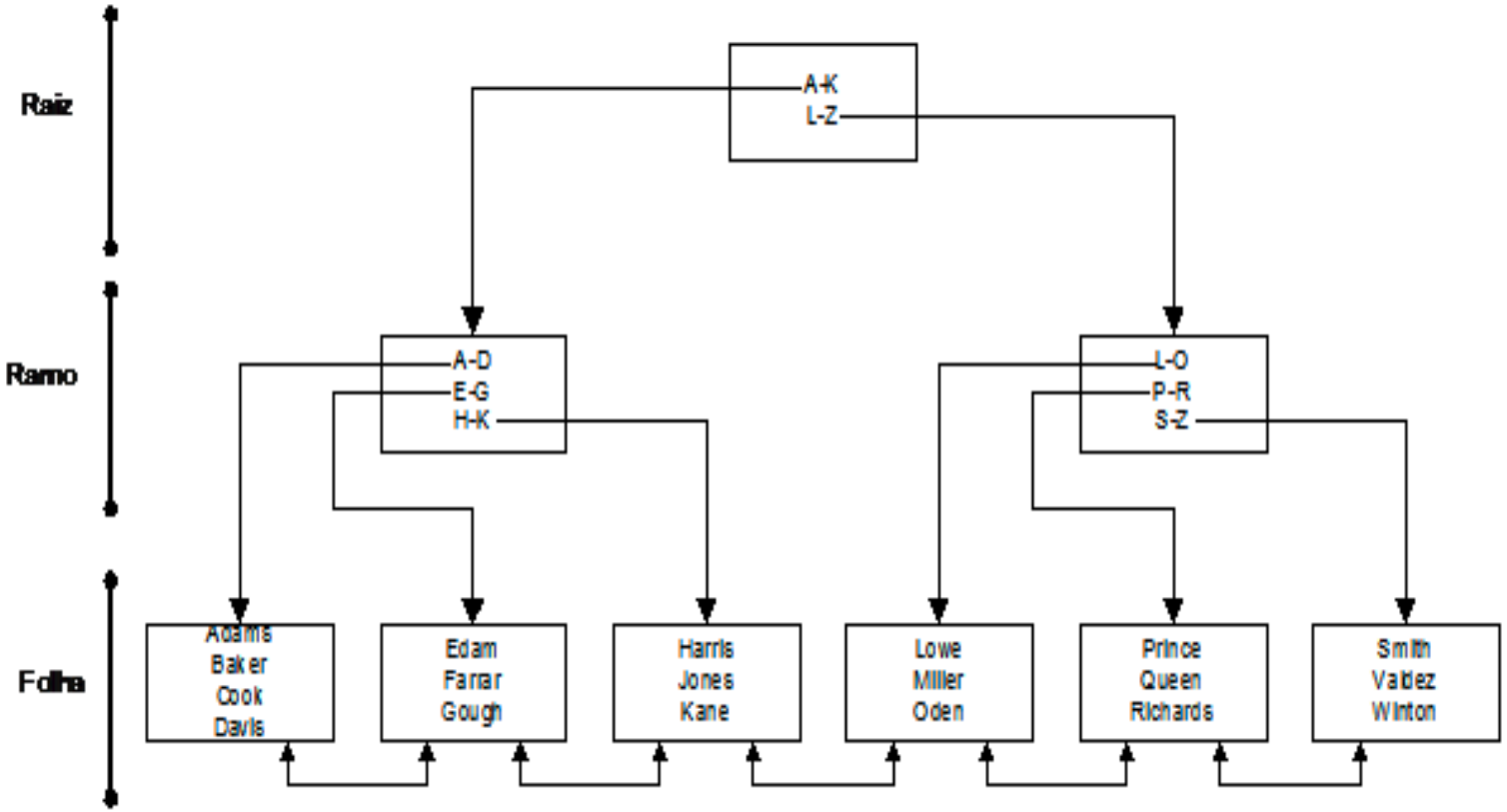
Índices em árvore B+

Árvore B+

- é uma árvore balanceada, cujas folhas contêm uma seqüência de pares chave-ponteiro.
- as chaves são ordenadas pelo seu valor.
- armazenamento de diversas chaves por nó, criando uma árvore de grau alto, ou seja, a altura da árvore tende a ser pequena, mesmo para um conjunto considerável de chaves



Indices em árvore B+



Seleção de Índices

- Apesar de grande esforço dos DBAs, existem sérios problemas de desempenho em SGBDs causados por configurações incorretas de índices.
- Ainda, há muito desconhecimentos sobre a necessidade de criá-los ou não.
- Considere o exemplo a seguir, onde 2 tipos de comandos SQL são submetidos, sobre uma tabela denominada `Funcionarios`:

(1) insert into Funcionarios (codigo, nome, salario, no_depto)
values (4, 'Fulano da silva', 5000, 7);

(2) select codigo, nome, salario `From Funcionarios`
where salario between 2500 and 5000
order by nome;

Seleção de Índices

- Considere um cenário onde os comandos são submetidos ao SGBD de forma concorrente e com uma freqüência de comandos do Tipo 2, muito maior do que os comandos do Tipo 1.
- Nesse cenário, um índice sobre as colunas salario e nome da tabela Funcionarios pode trazer benefícios de desempenho no processamento das transações onde estão inseridos.
- Os comandos do Tipo (1) são menos freqüentes, e assim é esperado um custo menor, decorrente da atualização do índice sobre a tabela Funcionarios, compensando a criação do mesmo.
- De outra forma, se tivermos a freqüência de comandos do Tipo (1) muito superior do que a freqüência de comandos do Tipo (2), a criação de índices não é recomendável, devido ao custo de atualização dos índices presentes sobre a tabela.

Seleção de Índices

- Tradicionalmente, os índices podem aumentar a velocidade de execução de consultas em SGBDs relacionais nos seguintes casos:
  - Limitar dados a ser acessados, quando se aplicam predicados;
  - Ordenações de registros utilizando cláusulas como ORDER BY ou GROUP BY;
  - Fazer junção de uma tabela com outra(s); e
  - Eliminar registros repetidos, utilizando a Cláusula DISTINCT.

Comando Explain

- Permite a visualização do plano de execução para cada comando SQL enviado pelos usuários, salvos comandos utilitários.
- Embora os custos fornecidos pelo Explain sejam apenas estimativas, já é possível fazer melhorias em comandos e a verificação da utilização de índices por parte do Otimizador do SGBD.

Comando Explain (cont.)

Dados apresentados:

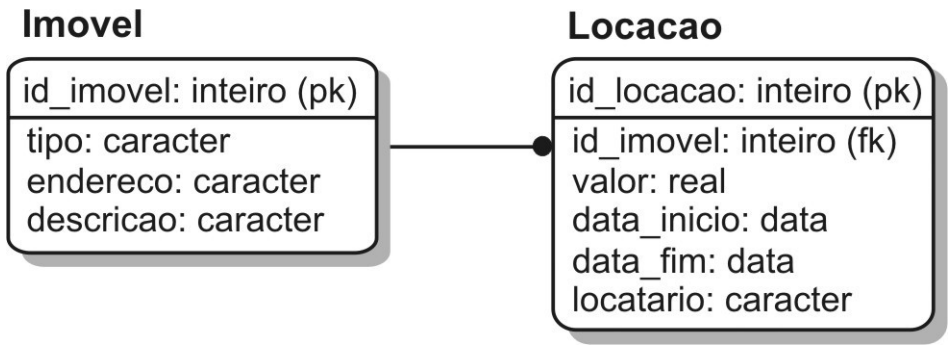
- Custo de partida estimado: estima o esforço gasto antes da varredura de saída iniciar. Como por exemplo, o tempo para fazer a ordenação em um nó.
- Custo total estimado: calcula o custo total gasto, caso todas as linhas forem pesquisadas pelo comando.
- Número de linhas de saída estimado: calcula o número estimado de linhas para o nó envolvido, caso seja executado até o fim.
- Largura média estimada: medida em bytes, referente ao tamanho das linhas de saída para o presente nó do plano.

Índices no PostgreSQL

- Exemplo utilizando Arvore B+:
  - Sintaxe:

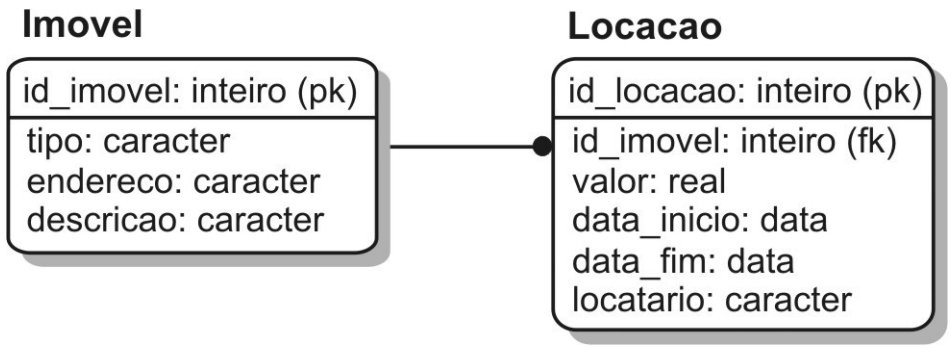
```
create index
      <nome_indice> on <tabela>
      using btree (<coluna1, coluna2, coluna n>);
```

Estudo de caso



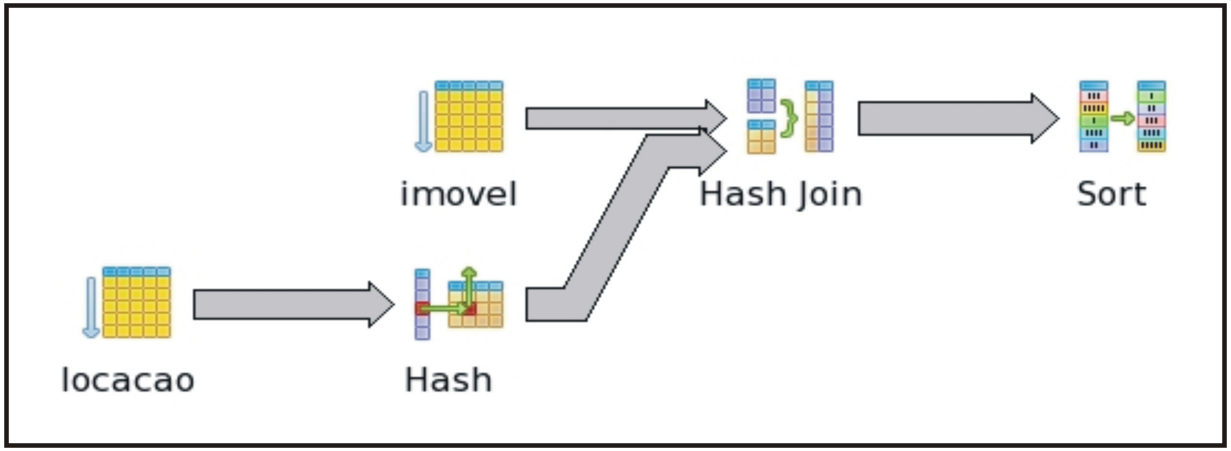

```
bd_imob=# select i.tipo,i.endereco,l.valor,l.locatario
bd_imob=# from imovel i, locacao l
bd_imob=# where valor between 1000 and 3000
bd_imob=# and data_inicio >= '20070101'
bd_imob=# and i.id_imovel = l.id_imovel
bd_imob=# order by l.valor;
```

Estudo de caso




```
bd_imob=# select i.tipo,i.endereco,l.valor,l.locatario
bd_imob=# from imovel i, locacao l
bd_imob=# where valor between 1000 and 3000
bd_imob=# and data_inicio >= '20070101'
bd_imob=# and i.id_imovel = l.id_imovel
bd_imob=# order by l.valor;
```

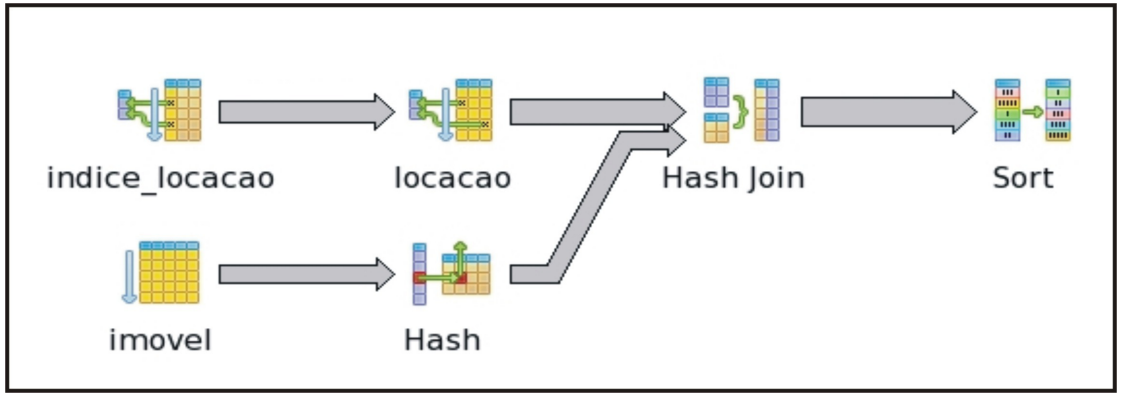
Índices no PostgreSQL



Sem índices

```
QUERY PLAN
------------------------------------------------------------------------
Sort (cost=9159.59..9205.01 rows=18171 width=63)
   Sort Key: l.valor
   -> Hash Join (cost=375.00..7156.55 rows=18171 width=63)
        Hash Cond: (l.id_imovel = i.id_imovel)
      -> Seq Scan on locacao l  (cost=0.00..5037.00 rows=18171 width=8)
           Filter: ((valor >= 1000) AND (valor <= 3000) AND
(data_inicio                  >= '2007-01-01 00:00:00-02'))
      -> Hash (cost=350.00..350.00 rows=10000 width=63)
          -> Seq Scan on imovel i(cost=0.00..350.00 rows=10000
width=63)
(8 rows)
```

Índices no PostgreSQL



Com indices

```
                    QUERY PLAN
------------------------------------------------------------------------
Sort (cost=6453.38..6497.29 rows=17563 width=63)
  Sort Key: l.valor
  -> Hash Join (cost=1502.93..4522.17 rows=17563 width=63)
      Hash Cond: (l.id_imovel = i.id_imovel)
      -> Bitmap Heap Scan on locacao l (cost=1127.93..2972.28
                                  rows=17563 width=8)
            Recheck Cond: ((valor >= 1000) AND (valor <= 3000) AND
                                   (data_inicio >= '2007-01-01 00:00:00'))
            -> Bitmap Index Scan on indice_locacao  (cost=0.00..1127.93


rows=17563 width=0)
                  Index Cond: ((valor >= 1000) AND (valor <= 3000) AND
                                   (data_inicio >= '2007-01-01 00:00:00'))
      -> Hash (cost=350.00..350.00 rows=10000 width=63)
          -> Seq Scan on imovel i(cost=0.00..350.00 rows=10000 width=63)
(10 rows)
```

Índices no Oracle

- Exemplo

```
CREATE [UNIQUE] INDEX nomeIndice ON
nomeTabela(coluna1 [, coluna2...])
[TABLESPACE nomeTablespace];
```

Índices no PostgreSQL

- O nome do índice é obrigatório e distingue este objecto de outros. Alguns comandos em Oracle criam índices de forma implícita, por exemplo a definição de uma restrição Primary Key ou Unique Key. Nestes casos o índice recebe o nome que for atribuído à restrição. Estes índices materializam a regra imposta pela restrição.
- A cláusula UNIQUE é opcional. Na sua ausência o índice suporta valores repetidos, enquanto na sua presença é obrigatório definir valores diferentes. As restrições PRIMARY KEY e UNIQUE KEY forçam a existência de um índice do tipo UNIQUE. No caso da restrição PRIMARY KEY ainda é forçado que todos os valores sejam NOT NULL.

Índices no Oracle

- Um índice está sempre associado a uma tabela e é removido de forma automática quando a respectiva tabela é removida.
- Um índice usa no mínimo uma coluna da tabela. Quando utiliza mais que uma coluna é um índice concatenado ou composto. Quando temos um índice concatenado e UNIQUE significa que as colunas do índice, consideradas de forma isolada, podem ter valores repetidos, mas a concatenação dos valores das colunas, considerada como um grupo, não pode ter valores repetidos. Se além disso o índice fizer parte da PRIMARY KEY, então todas as colunas têm que ter valores NOT NULL.
- A palavra reservada TABLESPACE indica onde o índice será guardado. Quando é omitida o índice é armazenado no TABLESPACE definido por omissão para esse utilizador. Faz parte das boas práticas do SGBD Oracle que as tabelas e os índices sejam guardados em TABLESPACES diferentes.