

MAC0439 – Laboratório de Bancos de Dados

Aula “Extra”

**Índices**

Acelerando o acesso  
aos dados

Profa. Kelly Rosa Braghetto

# O que são índices?

- ◆ Estruturas de armazenamento são compostas por *arquivos*.
- ◆ Por exemplo, um **arquivo de dados** pode ser usado para manter uma relação do BD.
- ◆ Um arquivo de dados pode ter um ou mais **arquivos de índice** associados a ele.
- ◆ Cada arquivo de índice associa valores da chave de busca a ponteiros para registros nos arquivos de dados que contém esses valores para o(s) atributo(s) da chave de busca.

# Índices

- ◆ *Índice* = estrutura de dados usada para acelerar o acesso às tuplas de uma relação, dados valores para um ou mais atributos
- ◆ Índices evitam “table scans”
  - ▶ Tuplas são localizadas imediatamente
- ◆ Índices são mantidos pelos SGBDs e ficam armazenados nos seus respectivos bancos de dados

# Tipos de Índices

## ◆ Primários X Secundários

- ▶ Índice primário: é especificado sobre o(s) atributo(s) da chave de ordenação do arquivo de dados
  - ▶ Índice secundário: não determina a localização dos registros.
- ◆ Uma relação pode ter no máximo um índice primário (geralmente criado sobre a chave primária), mas pode ter vários índices secundários sobre outros atributos.
- ◆ Importante não confundir:
- chave primária
  - chave de busca
  - chave de ordenação

# Tipos de Índices

## ◆ **Densos X Esparsos**

- ▶ Índice denso: possui uma entrada no arquivo para cada registro no arquivo de dados
- ▶ Índice esparsos: possui entradas somente para alguns dos registros no arquivo de dados (geralmente, uma entrada no índice para cada bloco do arquivo de dados)
  - Só pode ser usado se o arquivo de dados estiver ordenado pela chave de busca

# Índice Primário Denso

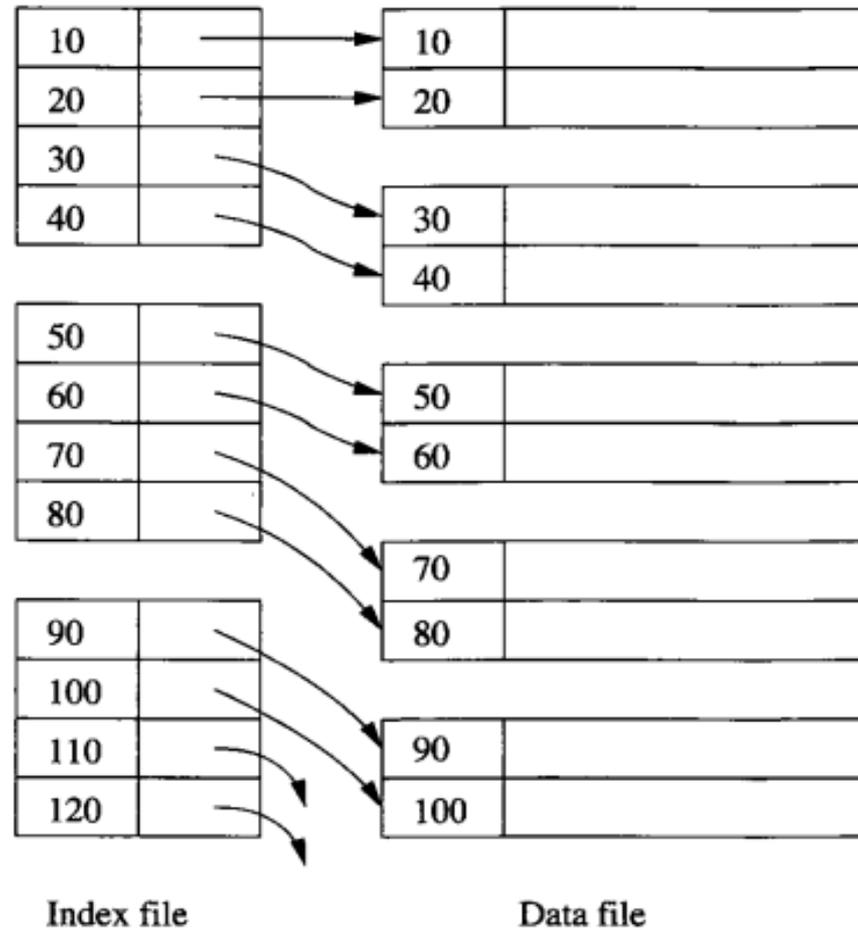
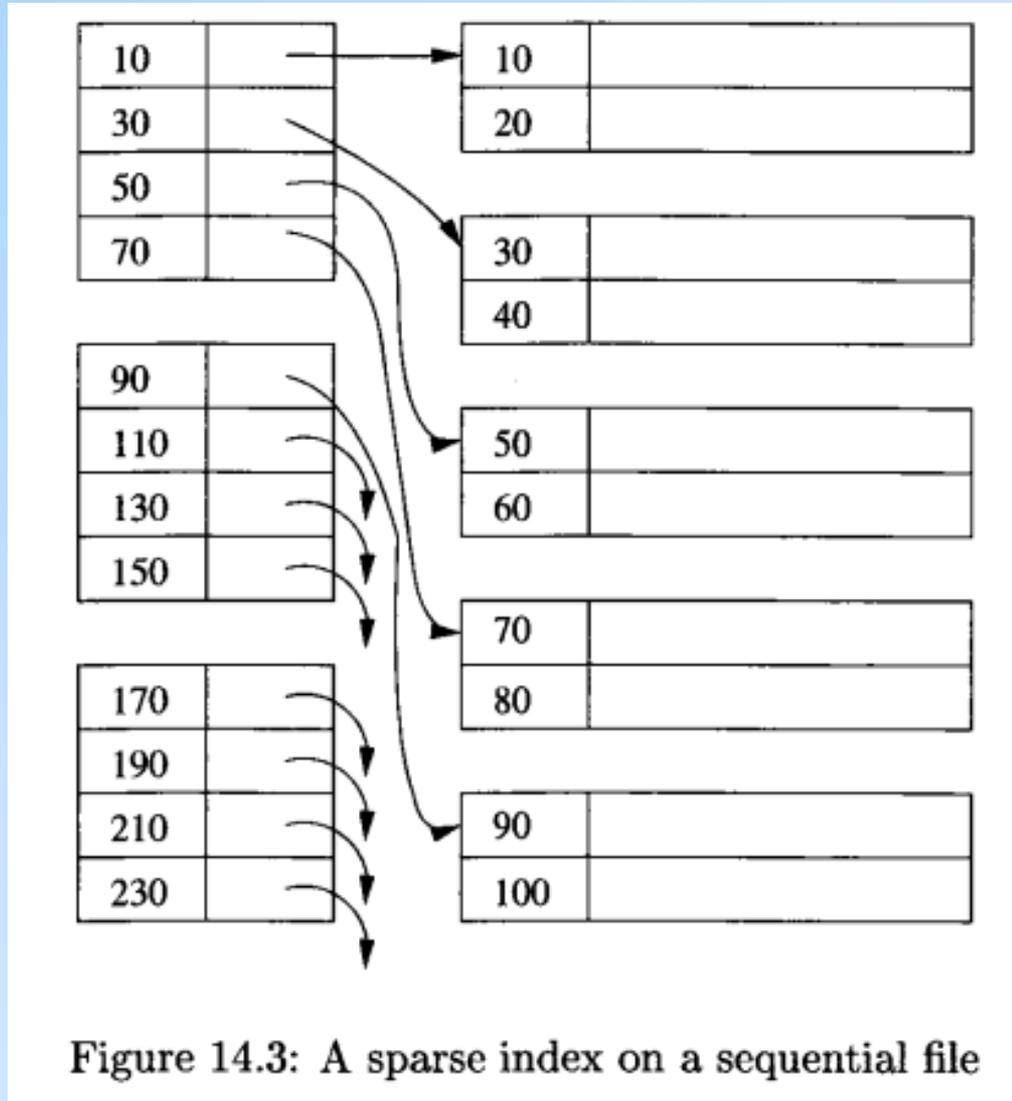


Figure 14.2: A dense index (left) on a sequential data file (right)

# Índice Primário Esparso



# Índice Esparso Multinível

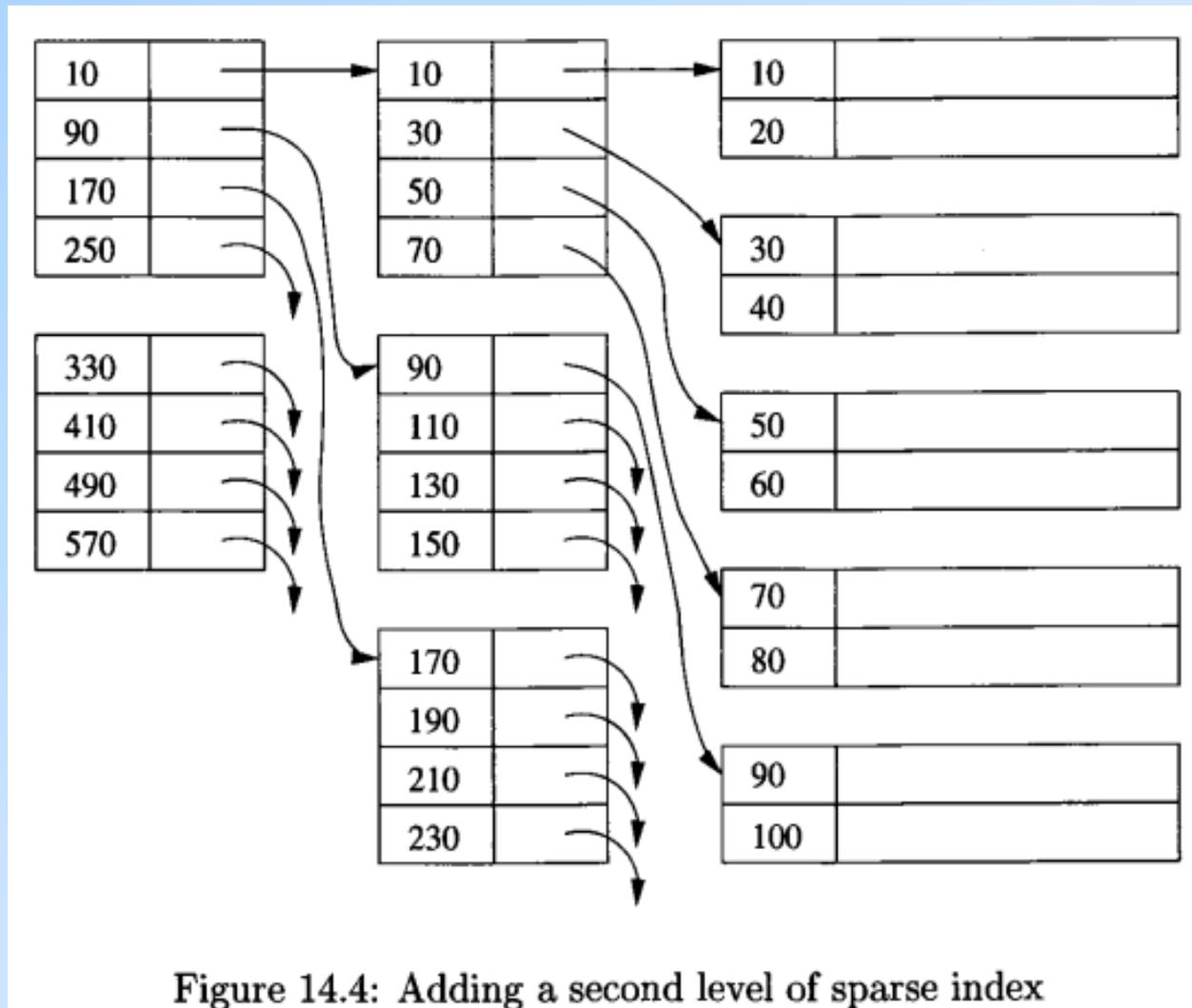


Figure 14.4: Adding a second level of sparse index

# Índice Secundário

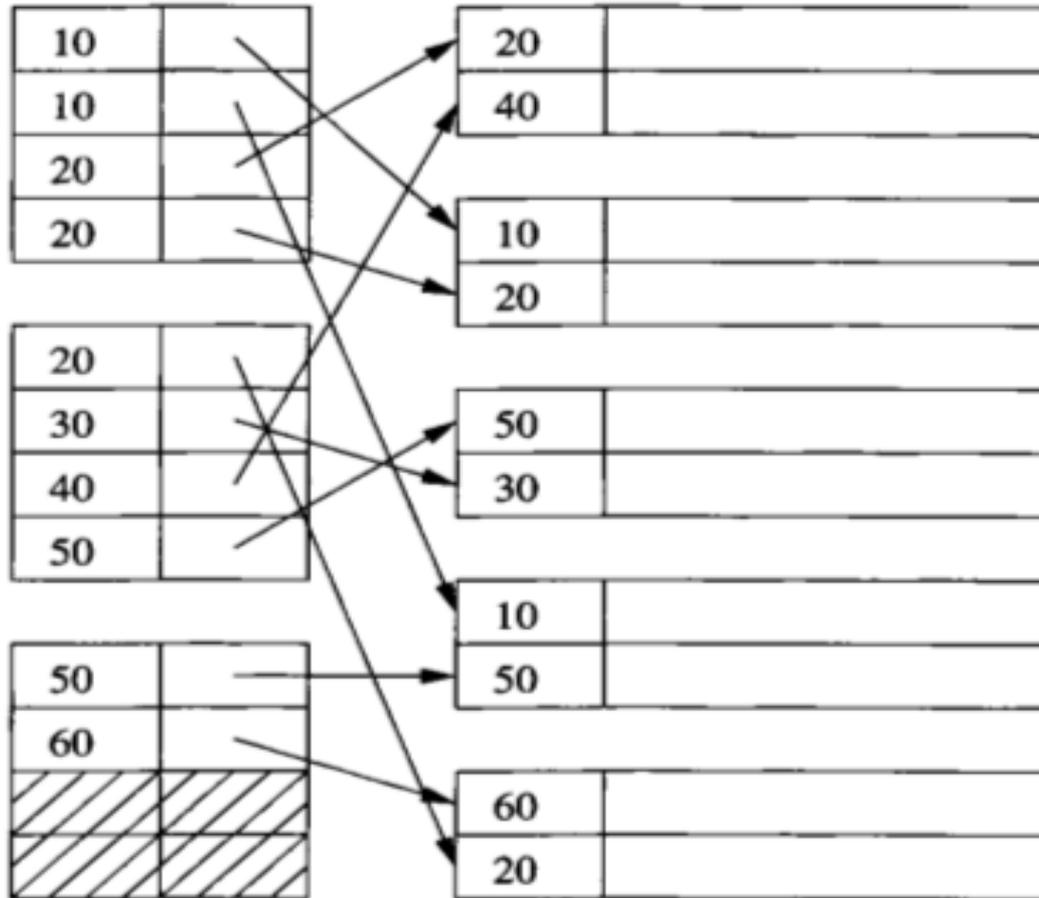


Figure 14.5: A secondary index

**Índices secundários são sempre densos! Por quê?**

# Índice Secundário com Indireção

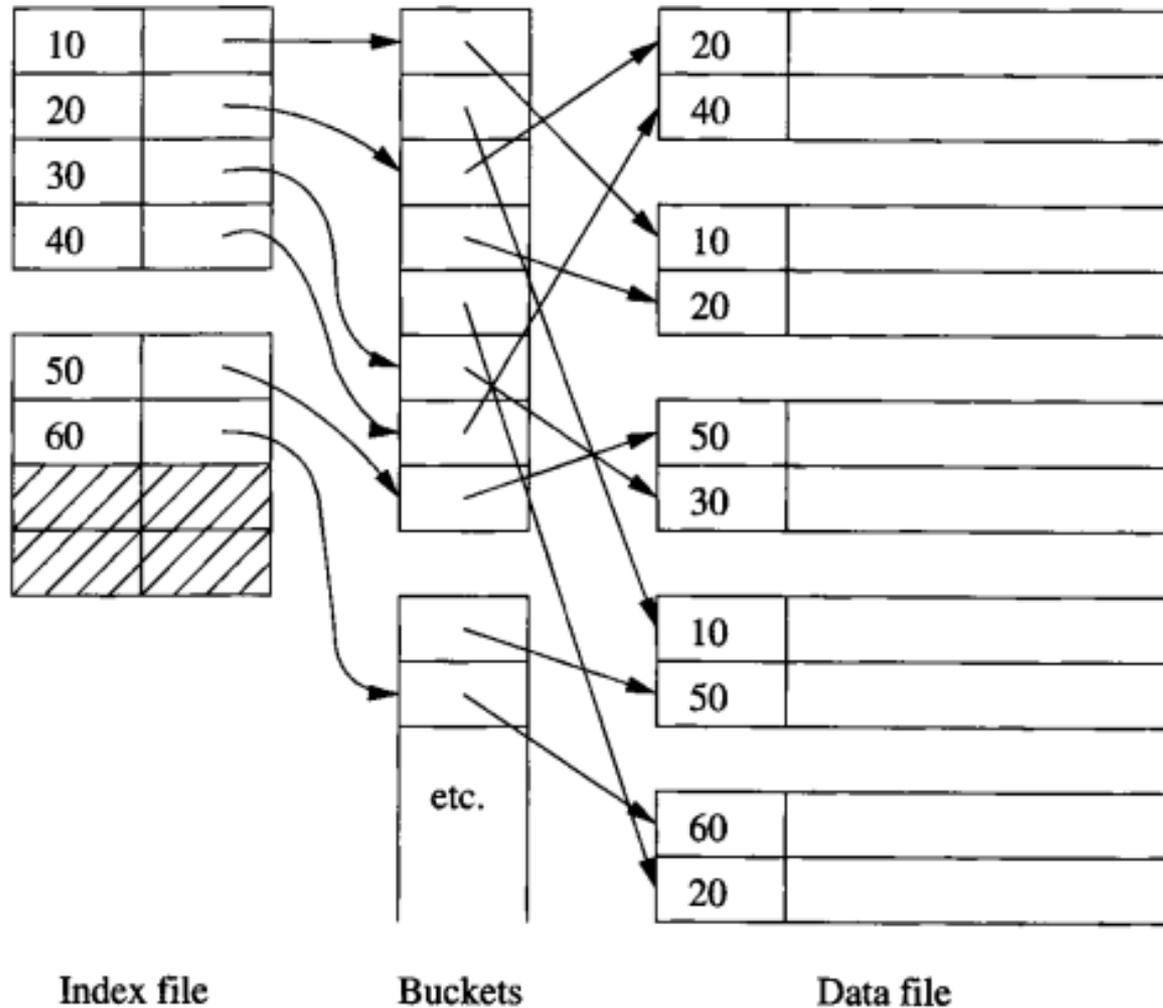
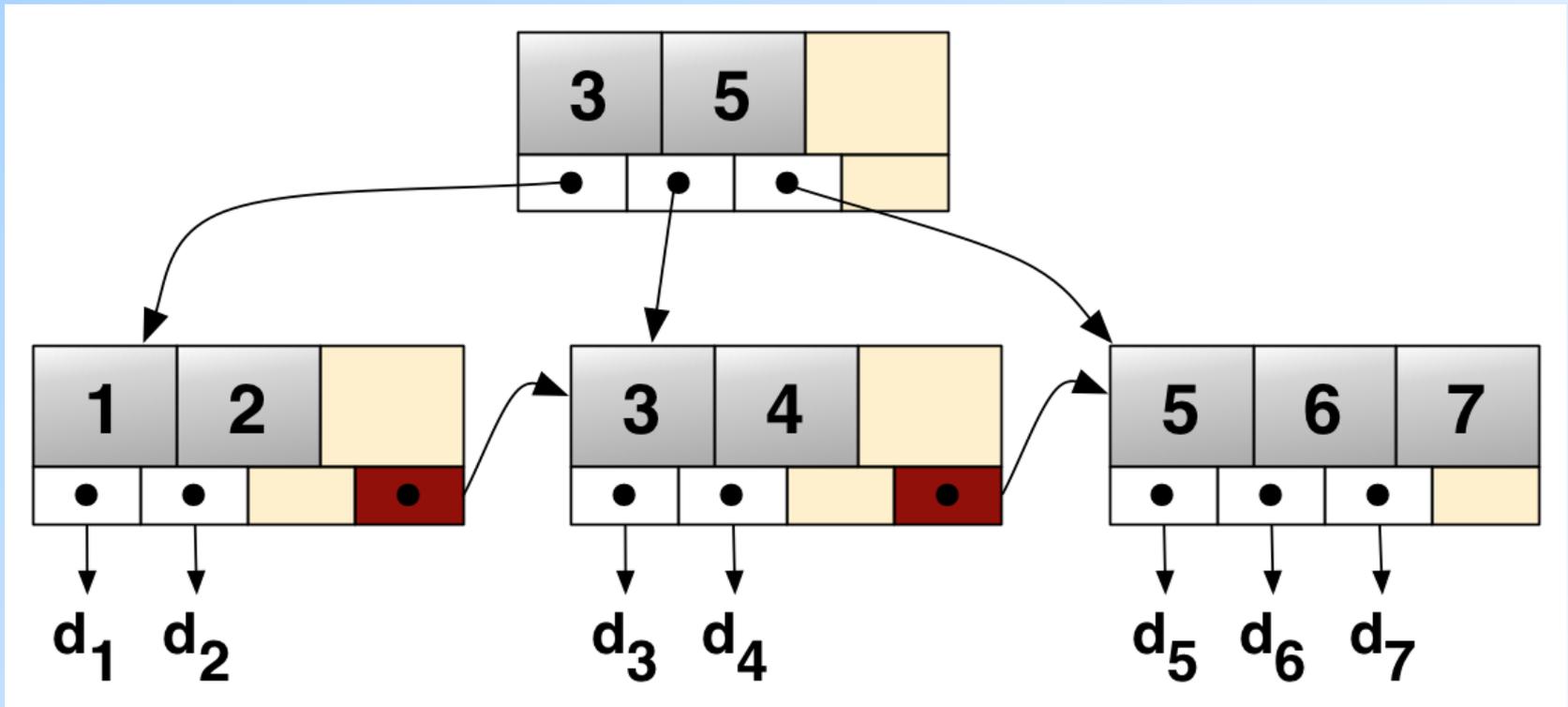


Figure 14.7: Saving space by using indirection in a secondary index

# Estruturas de Dados Usadas para Implementar Índices

- ◆ Tabela *hash*
  - ▶ Tempo de acesso: constante
  - ▶ Úteis para condições envolvendo comparações de igualdade
- ◆ Árvore balanceada de busca, com nós “gigantes” (uma página inteira do disco)
  - ▶ *árvore-B (B-tree)*
  - ▶ Tempo de acesso: logarítmico
  - ▶ Úteis para condições envolvendo comparações feitas com os operadores =, >, >=, <, <=

# Recordar é Viver: Estrutura de Índice B+ Tree



Fonte: [http://en.wikipedia.org/wiki/B+\\_tree](http://en.wikipedia.org/wiki/B+_tree)

# Recordar é Viver: Estrutura de um Índice Hash

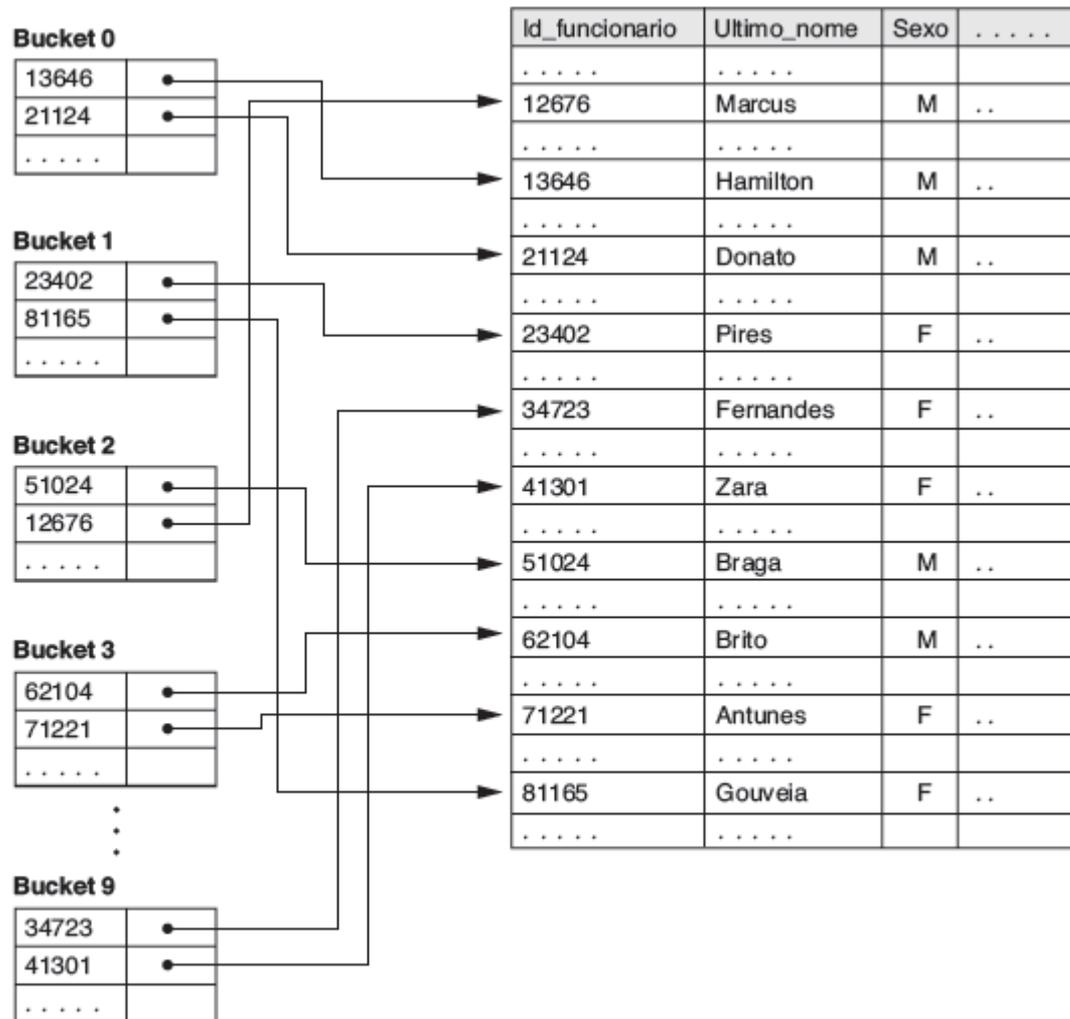


Figura 18.15

Indexação baseada em hash.

# Declaração de Índices

## ◆ Sintaxe típica na SQL

```
CREATE INDEX <nome_indice> ON  
  <nome_tabela>(<lista de atributos>);
```

## ◆ Exemplos:

```
CREATE INDEX IdxFabRefri ON  
  Refrigerantes(fabricante);
```

```
CREATE UNIQUE INDEX IdxRefri ON  
  Refrigerantes(nome);
```

```
CREATE INDEX IdxVenda ON  
  Vendas(nome_lanch, nome_refri);
```

# O Uso de Índices

- ◆ Dado um valor  $v$ , o índice nos leva apenas às tuplas que possuem  $v$  como valor para o(s) atributo(s) do índice
- ◆ **Exemplo:** use `IdxFabRefri` e `IdxVendas` para encontrar o preço dos refrigerantes produzidos pela 'Cola-Coca' e vendidos no 'Sujinhos'. (próximo slide)

# O Uso de Índices (2)

```
SELECT price FROM
           Refrigerantes, Vendas
WHERE fabricante = 'Cola-Coca' AND
       Refrigerante.nome =
       Vendas.nome_refri AND
       nome_lanch = 'Sujinhos';
```

1. Usa IdxFabRefri para obter todos os refrigerantes feitos pela 'Cola-coca'.
2. Depois, usa IdxVendas para obter os preços desses refrigerantes no Sujinhos

# Sintonia Fina (*Tuning*) de BDs

- ◆ Uma das principais dificuldades relacionadas a acelerar o acesso a um BD é decidir quais índices criar
- ◆ **Pró:** Um índice melhora o desempenho de consultas que podem usá-los
- ◆ **Contra:** Um índice piora o desempenho das modificações feitas sobre sua relação (porque uma modificação na relação pode fazer com que o índice precise ser atualizado também)

# Efeitos “Colaterais” do Uso de Índices

- ◆ Mais consumo de espaço de armazenamento (problema pequeno)
- ◆ Custo para a criação do índice (problema médio)
- ◆ Custo para a manutenção do índice (problema grave)
  - ▶ Os benefícios do uso de um índice para melhorar o tempo de execução das consultas podem ser mitigados por esse custo de manutenção

# Exemplo: Sintonia Fina

- ◆ Suponha que as únicas coisas que fazemos sobre o nosso BD de refri são:
  1. Inserir novos fatos em uma relação (10%).
  2. Encontrar o preço de um dado refri em uma dada lanchonete (90%).
- ◆ Nesse caso, **IdxVendas** em `Vendas(nome_lanch, nome_refri)` realmente ajudaria em um melhor desempenho, mas **IdxRefri** sobre `Refrigerantes(fabricante)` somente atrapalharia .

# Benefícios de um Índice

## Dependem de:

- ◆ Tamanho da tabela (e possivelmente de seu layout)
- ◆ Distribuição dos valores das colunas indexadas
- ◆ Frequência de consultas X frequência de modificações

# Softwares “Conselheiros” para Sintonia Fina (*Tuning Advisors*)

- ◆ Subárea de pesquisa muito importante
  - ▶ Fazer a sintonia de um BD de forma manual é uma tarefa muito árdua
- ◆ Um conselheiro obtém um conjunto de consultas que serão usadas como carga de trabalho para avaliar o desempenho do BD. Para constituir essa carga, o conselheiro:
  1. Escolhe consultas de forma aleatória, a partir de um histórico de consultas executadas sobre o BD ou
  2. Usa consultas de exemplo fornecidas por um projetista do BD

# Software “Conselheiros” para Sintonia Fina *(Tuning Advisors)* (2)

- ◆ O conselheiro gera índices candidatos e avalia cada um deles usando a carga de trabalho (= consultas) selecionada:
  - ▶ Cada consulta da carga é submetida ao otimizador de consulta, que assume que somente o índice em avaliação está disponível
  - ▶ A melhora/degradação no tempo médio de execução das consultas é medida

# Criação de Índices no PostgreSQL

- ◆ Estrutura geral (bem simplificada!):

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ]
```

```
<nome do índice> ON < nome da tabela >
```

```
[ USING <método> ] ( { coluna } [ ASC | DESC ] [, ...] );
```

- ◆ **UNIQUE** cria uma restrição de unicidade sobre a(s) coluna(s) do índice.
- ◆ **CONCURRENTLY** permite o PostgreSQL construir o índice sem bloquear a tabela para modificações concorrentes (inserts, updates ou deletes) → **isso pode ser perigoso!**

# Criação de Índices no PostgreSQL

- ◆ O índice também pode ser construído para valores computados a partir de uma expressão envolvendo um ou mais atributos de uma tabela. Ex:

```
CREATE INDEX idx_maiuscula ON  
Lanchonete(upper(nome)) ;
```

- ◆ Além de índices para tabelas, podemos também criar índices para visões materializadas.

# Métodos (Tipos) para Índices no PostgreSQL

Métodos existentes no PostgreSQL:

**Btree (padrão), Hash, GiST, SP-GiST e GIN**

- ◆ Somente o método **Btree** pode ser usado para índices do tipo **UNIQUE**.
- ◆ Somente os métodos **Btree, GiST e GIN** suportam índices com várias colunas.
- ◆ Índices com mais de um campo somente serão utilizados para agilizar consultas se as cláusulas envolvendo os campos indexados forem ligadas por AND.
- ◆ Quando indicamos atributos como chave primária em uma tabela, automaticamente é criado um índice Btree sobre eles.

# Métodos para a Criação de Índices no PostgreSQL

- ◆ **Btree** - Árvores-B podem ser usadas em consultas com condições de seleção baseadas em igualdade de valores ou em intervalos, sobre dados que podem estar armazenados ordenadamente
- ◆ O planejador de consultas do PostgreSQL considerará o uso de um índice do tipo **Btree** sempre que uma coluna indexada estiver envolvida em uma comparação usando um ou mais dos seguintes operadores:
  - ▶ **<, <=, =, >= e >**
  - ▶ **like** (se o padrão for uma constante e estiver ancorado no início da string, como em coluna like 'MAC%')

# Métodos para a Criação de Índices no PostgreSQL

- ◆ **Hash** - só pode ser usado em consultas envolvendo condições de seleção baseadas em simples comparações de igualdade
- ◆ O planejador de consultas do PostgreSQL considerará o uso de um índice do tipo **Hash** sempre que uma coluna indexada estiver envolvida em uma comparação usando o operador “=”
- ◆ Nota: por problemas na forma como ele é implementado no gerenciador, o uso desse tipo de índice no PostgreSQL atualmente é desencorajado

# GiSTs

## *(Generalized Search Trees)*

- ◆ **GiSTs** não são um tipo único de índice, mas sim uma infraestrutura dentro da qual muitas estratégias de indexação diferentes podem ser implementadas para diferentes tipos de dados.
- ◆ São baseados em árvores de busca balanceadas.
- ◆ A distribuição padrão do PostgreSQL inclui classes de operações para vários tipos de dados bidimensionais e geométricos, que suportam consultas indexadas usando os seguintes operadores:
  - ◆ <<, &<, &>, >>, <<|, &<|, |&>, |>>, @>, <@, ~ =, &&
  - ◆ O significado desses operadores está em:  
<http://www.postgresql.org/docs/9.3/static/functions-geometry.html>

# GiSTs

## *(Generalized Search Trees)*

- ◆ Índices do tipo GiST também são capazes de otimizar consultas do tipo "vizinhos-mais próximos", como em:

```
SELECT * FROM lugares
ORDER BY localizacao <-> point '(101,456)'
LIMIT 10;
```

que encontra os 10 lugares mais próximos de um dado ponto alvo.

- ◆ O operador <-> calcula a distância entre pontos.

# SP-GiSTs

## *(Space-partitioned GiSTs)*

- ◆ **Assim como os GiSTs, os SP-GiSTs são** uma infraestrutura dentro da qual diferentes estratégias de indexação podem ser implementadas.
- ◆ Permitem a implementação de estruturas de dados em disco **não balanceadas**, como as **árvores de prefixo** (*tries*) e as **árvores k-d**
- ◆ Suportam consultas indexadas usando os seguintes operadores:
  - ◆  $\ll$ ,  $\gg$ ,  $\sim$ ,  $\leq$ ,  $\leq^@$ ,  $\leq^{\wedge}$ ,  $\geq^{\wedge}$
  - ◆ O significado desses operadores está em:  
<http://www.postgresql.org/docs/9.3/static/functions-geometry.html>

# GIN

## *(Generalized Inverted Index)*

- ◆ **GINs** são índices invertidos que podem lidar com valores que contêm mais de uma chave, como um vetor, por exemplo.
- ◆ Assim como GiST, GIN pode suportar diferentes estratégias de indexação.
- ◆ Os operadores com os quais um índice GIN pode ser usado variam dependendo da estratégia de indexação.
- ◆ A distribuição do PostgreSQL inclui classes de operadores GIN para vetores unidimensionais (tipo *array*), que suportam consultas usando os seguintes operadores:
  - ▶ <@ (está contido em?), @> (contém?), =, && (sobrepõe?)
  - ▶ <http://www.postgresql.org/docs/9.3/static/functions-array.html>

# Plano de Execução de uma Consulta

- ◆ Para ver qual é o plano de execução que um SGBD usa para uma dada consulta, usamos o comando **EXPLAIN**.
- ◆ Exemplo:

```
EXPLAIN SELECT price FROM Refrigerantes, Vendas
WHERE fabricante = 'Cola-Coca' AND
  Refrigerante.nome = Vendas.nome_refri AND
  nome_lanch = 'Sujinhos';
```

# Para mais Detalhes sobre Índices no PostgreSQL

- ◆ Livro: “Sistemas de Bancos de Dados”, 6ª Edição, Elmasri e Navathe – Capítulo 18
- ◆ Livro: “Database Systems – The complete book”, 2nd edition, Garcia Molina, Ullman, Widom – Seções 8.3, 8.4 e Capítulo 14
- ◆ <http://www.postgresql.org/docs/9.3/static/sql-createindex.html>
- ◆ <http://www.postgresql.org/docs/9.3/static/indexes.html>
- ◆ <http://www.postgresql.org/docs/9.3/static/gist.html>
- ◆ <http://www.postgresql.org/docs/9.3/static/spgist-intro.html>
- ◆ <http://www.postgresql.org/docs/9.3/static/gin-intro.html>

# Referências Bibliográficas

- ◆ Capítulo 6 do livro “Database Systems – The Complete Book” (1ª edição), Garcia-Molina, Ullman e Widom