INF / UFG

Disciplina

Banco de Dados

Conteúdo

Estruturas de indexação para arquivos.

Preâmbulo

A ideia por trás de um índice ordenado é semelhante à que está por trás do índice usado em um livro, que lista termos importantes ao final, em ordem alfabética, junto com uma lista dos números de página onde o termo aparece no livro.

Podemos pesquisar o índice do livro em busca de certo termo em seu interior e encontrar uma lista de endereços — números de página, nesse caso — e usar esses endereços para localizar as páginas especificadas primeiro e depois procurar o termo em cada página citada.

Preâmbulo

Vamos assumir que existe um arquivo com alguma organização: (i) não-ordenada, (ii) ordenada, ou (iii) *hashing*.

Vamos descrever estruturas de acesso auxiliares chamados **índices**:

>> são usados para acelerar a recuperação de registros;
>> em geral, são arquivos adicionais com caminhos de acesso secundários;
>> proporcionam formas alternativas de acessar os registros;
>> não afetam o lugar físico de registros no arquivo de dados primário;
>> permitem acesso eficiente a registos com base nos campos de indexação;
>> qualquer campo do arquivo pode ser usado para criar um índice;
>> pode-se construir vários índices para um mesmo arquivo de dados.

Inicialmente, estudaremos índices implementados em um único nível ...

Índices Primários

É um arquivo ordenado cujos registros são de tamanho fixo.

Cada registro possui dois campos: **< K(i) , P(i) >**

>> **K(i)** é o do mesmo tipo da chave de ordenação do arquivo (chave primária do arquivo de dados), e
>> **P(i)** é um ponteiro para um bloco do arquivo de dados em disco (um endereço de bloco).

Há um registro no arquivo de índice para cada bloco do arquivo de dados.

Portanto, **K(i) refere-se ao valor do campo-chave do primeiro registro em um bloco de dados**.

Índices Primários

Como criar um índice primário para o arquivo ?

	Nome	Cpf	Data_nascimento	Cargo	Salario	Sexo
Bloco 1	Aaron, Eduardo					
	Abílio, Diana					
	⋮					
	Acosta, Marcos					
Bloco 2	Adams, João					
	Adams, Roberto					
	⋮					
	Akers, Janete					
Bloco 3	Alexandre, Eduardo					
	Alfredo, Roberto					
	⋮					
	Allen, Samuel					
Bloco 4	Allen, Tiago					
	Anderson, Kely					
	⋮					
	Anderson, Joel					
Bloco 5	Anderson, Isaac					
	Angeli, José					
	⋮					
	Anita, Sueli					
Bloco 6	Arnoldo, Marcelo					
	Arnoldo, Estevan					
	⋮					
	Atílio, Timóteo					
	⋮					
Bloco *n*–1	Wanderley, Jaime					
	Wesley, Ronaldo					
	⋮					
	Wong, Manuel					
Bloco *n*	Wong, Pâmela					
	Wuang, Charles					
	⋮					
	Zimmer, André					

Figura 17.7
Alguns blocos de um arquivo ordenado (sequencial) de registros de FUNCIONARIO com Nome como campo-chave de ordenação.

Índices Primários

As três primeiras entradas de índice são as seguintes:

<K(1) = (Aaron, Eduardo), P(1) = endereço de bloco 1>

<K(2) = (Adams, João), P(2) = endereço de bloco 2>

<K(3) = (Alexandre, Eduardo), P(3) = endereço de bloco 3>

O número total de entradas no índice é igual ao número de blocos de disco no arquivo de dados ordenado.

O primeiro registro em cada bloco do arquivo de dados é chamado de **registro de âncora do bloco** ou, simplesmente, **âncora de bloco**.

Índices Primários

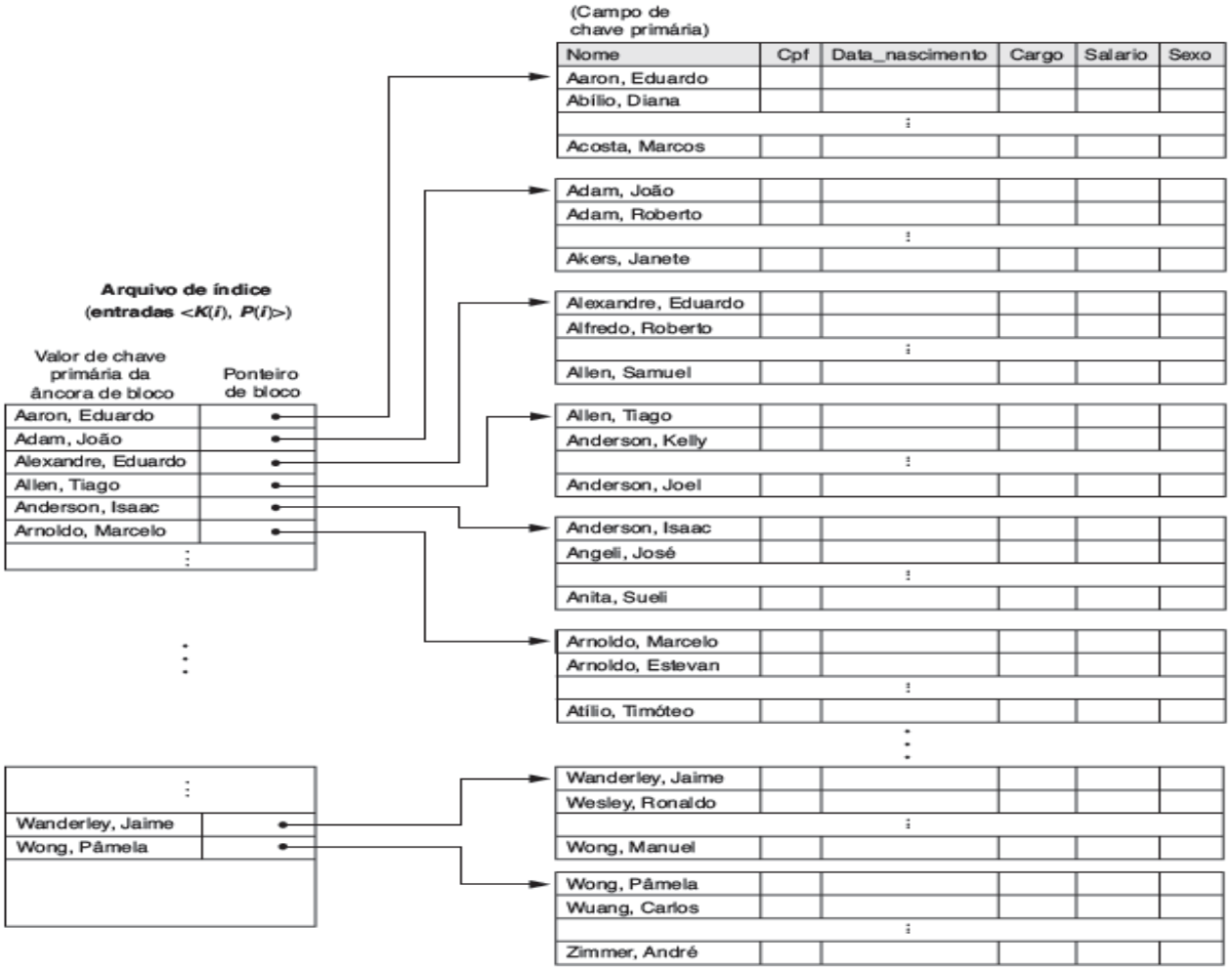

Figura 18.1
Índice primário no campo de chave de ordenação do arquivo mostrado na Figura 17.7.

Índices Primários

O arquivo de índice primário ocupa um espaço muito menor do que o arquivo de dados, por duas razões:

1) há menos entradas de índice do que há registros no arquivo de dados;
2) cada entrada de índice é, tipicamente, menor em tamanho do que um registo de dados, porque possui apenas dois campos.

Uma busca binária no arquivo de índice requer menos blocos acessados do que uma busca binária no arquivo de dados:

b é o número de blocos no arquivo de dados
bi é o número de blocos no arquivo de índice primário
como **bi < b** , então

$$((\log_2 bi) + 1) < (\log_2 b)$$

Índices Primários

Exemplo 1: Suponha

- um arquivo ordenado possui **r = 30.000** registros;
- tamanho de bloco **B = 1024** bytes;
- registros não espalhados;
- registros de tamanho fixo **R = 100** bytes.

Fator de bloco é **bfr = ⌊(B/R)⌋ = ⌊(1024/100)⌋ = 10** registros por bloco.

O número de blocos do arquivo de dados é **b = ⌈(r/bfr)⌉ = ⌈(30000/10)⌉ = 3000** blocos.

Uma busca binária no arquivo de dados precisa acessar, aproximadamente, $\lceil \log_2 b \rceil = \lceil (\log_2 3000) \rceil = 12$ blocos.

Índices Primários

Exemplo 1: Suponha, ainda:
- o campo chave de ordenação possui **V = 9** bytes;
- um ponteiro de bloco ocupa **P = 6** bytes.

Cada registro (entrada) de índice possui **Ri = (9 + 6) = 15** bytes.

O fator de bloco do arquivo de índice é **bfri = $\lfloor$(B/Ri)$\rfloor$ = $\lfloor$(1024/15)$\rfloor$ = 68** registros (entradas) por bloco.

O número total de registros (entradas) de índice é igual ao número de blocos do arquivo de dados, portanto é **3000**.

O número de blocos de índices é **bi = $\lceil$(ri/bfri)$\rceil$ = $\lceil$(3000/68)$\rceil$ = 45**.

Uma busca binária no arquivo de índice precisa acessar, aproximadamente, $\lceil(\log_2 bi)\rceil = \lceil(\log_2 45)\rceil = 6$ blocos.

Para encontrar um registro usando o índice, é necessário acessar 6 + 1 = 7 blocos.

Índices Primários

A inserção e a exclusão de registros é um problema para qualquer arquivo ordenado.

Por que, em um índice primário, tal problema é agravado ?

R – a movimentação de registros no arquivo de dados, visando a criar espaço para um novo registro sendo inserido, modifica os registros de alguns blocos de ancoragem.

Algumas alternativas (não isentas de reorganização periódica de arquivos):

- usar blocos de *overflow* com registros não ordenados;
- lista ligada de registros de *overflow* para cada bloco de dados;
- marcação lógica de registros excluídos.

Índice Denso X Índice Esparso

Um **índice denso** possui uma entrada para cada valor da chave de pesquisa (e, portanto, todos os registros) no arquivo de dados.

A **índice esparso** (espalhado, disperso) possui entradas de índice para apenas alguns dos valores de pesquisa.

Um índice primário é um índice esparso, uma vez que inclui uma entrada para cada bloco de disco do arquivo de dados (as chaves do seu registro de âncora) e não para cada valor de pesquisa (ou a cada registro).

Índices de Agrupamento (*clustering indexes*)

Se os registros de arquivo estiverem fisicamente ordenados em relação a um campo que permite duplicação de valores (campo de ordenação não tem um valor distinto para cada registro), então:

>> o campo é chamado **campo de agrupamento** (*clustering field*),
>> o arquivo de dados é chamado **arquivo de agrupamento** (*clustering file*).

Um **índice de agrupamento** (*clustering index*) busca a acelerar a recuperação de todos os registros que tenham o mesmo valor para o campo de agrupamento.

>> difere de um índice primário, pois, no índice de agrupamento, o campo de ordenação do arquivo de dados pode possuir valores repetidos.

Índices de Agrupamento (*clustering indexes*)

Um índice de agrupamento também é um arquivo ordenado com dois campos:
>> o primeiro campo é do mesmo tipo do campo de agrupamento do arquivo de dados, e
>> o segundo campo é um ponteiro de bloco de disco.

Há uma entrada (registro) no índice de agrupamento para cada valor distinto do campo de agrupamento.

Cada entrada contém:
>> o valor, e
>> um ponteiro para o primeiro bloco no arquivo de dados que tem um registro com esse valor para seu campo de agrupamento.

Índices de Agrupamento (*clustering indexes*)

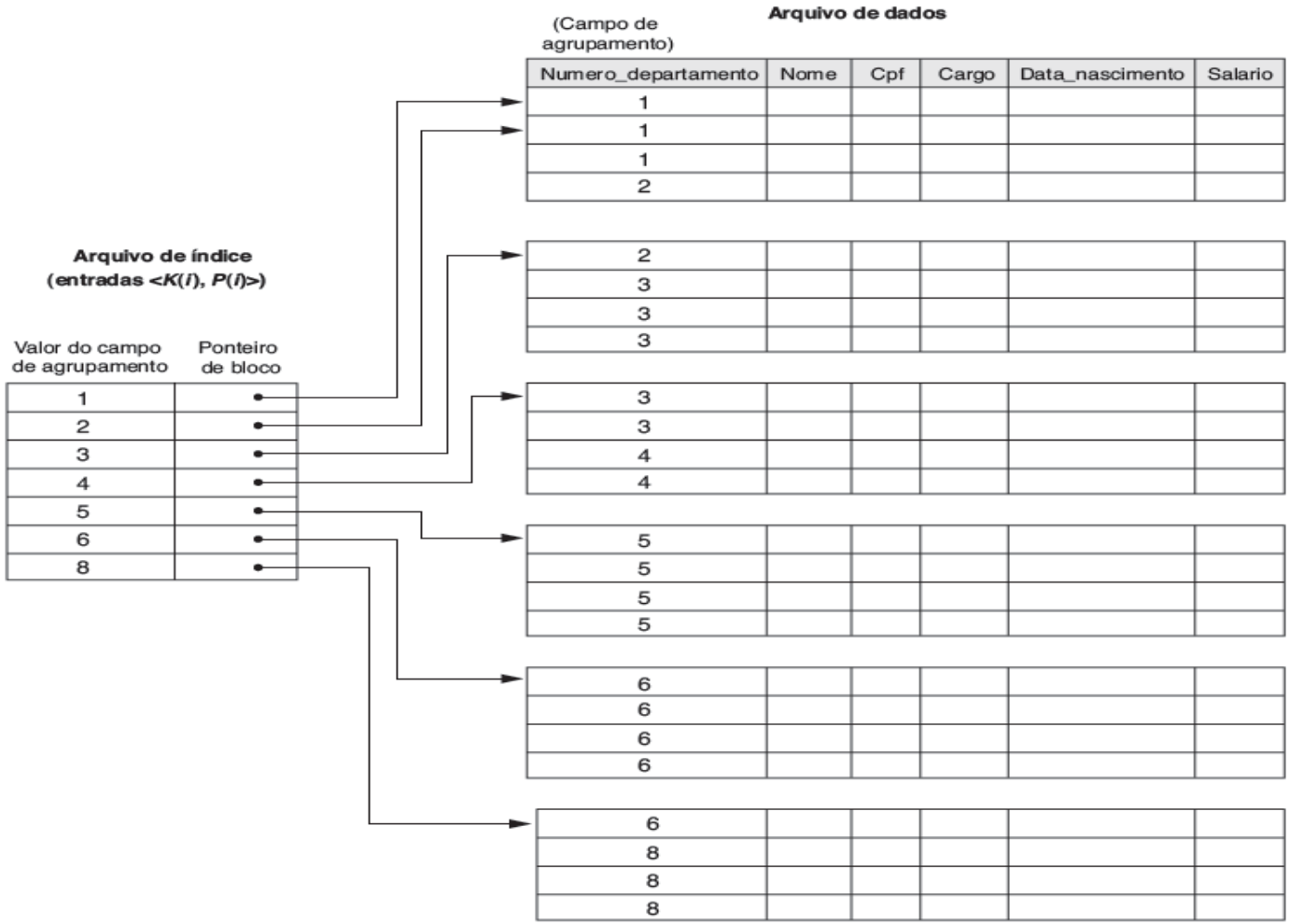

Figura 18.2
Um índice de agrupamento no campo não chave de ordenação Numero_departamento de um arquivo FUNCIONARIO.

Índices de Agrupamento (*clustering indexes*)

A inserção e exclusão de registro pode ainda causar problemas, porque os registros de dados são fisicamente ordenados.

Para aliviar o problema de inserção, pode-se (é comum?) a reserva de um bloco inteiro (ou um conjunto de blocos contíguos) para cada valor do campo de agrupamento:

>> todos os registos com esse valor são colocados no bloco (ou conjunto de blocos);
>> Figura 18.3.

Um índice de agrupamento é outro exemplo de um índice esparso.

Índices de Agrupamento (*clustering indexes*)

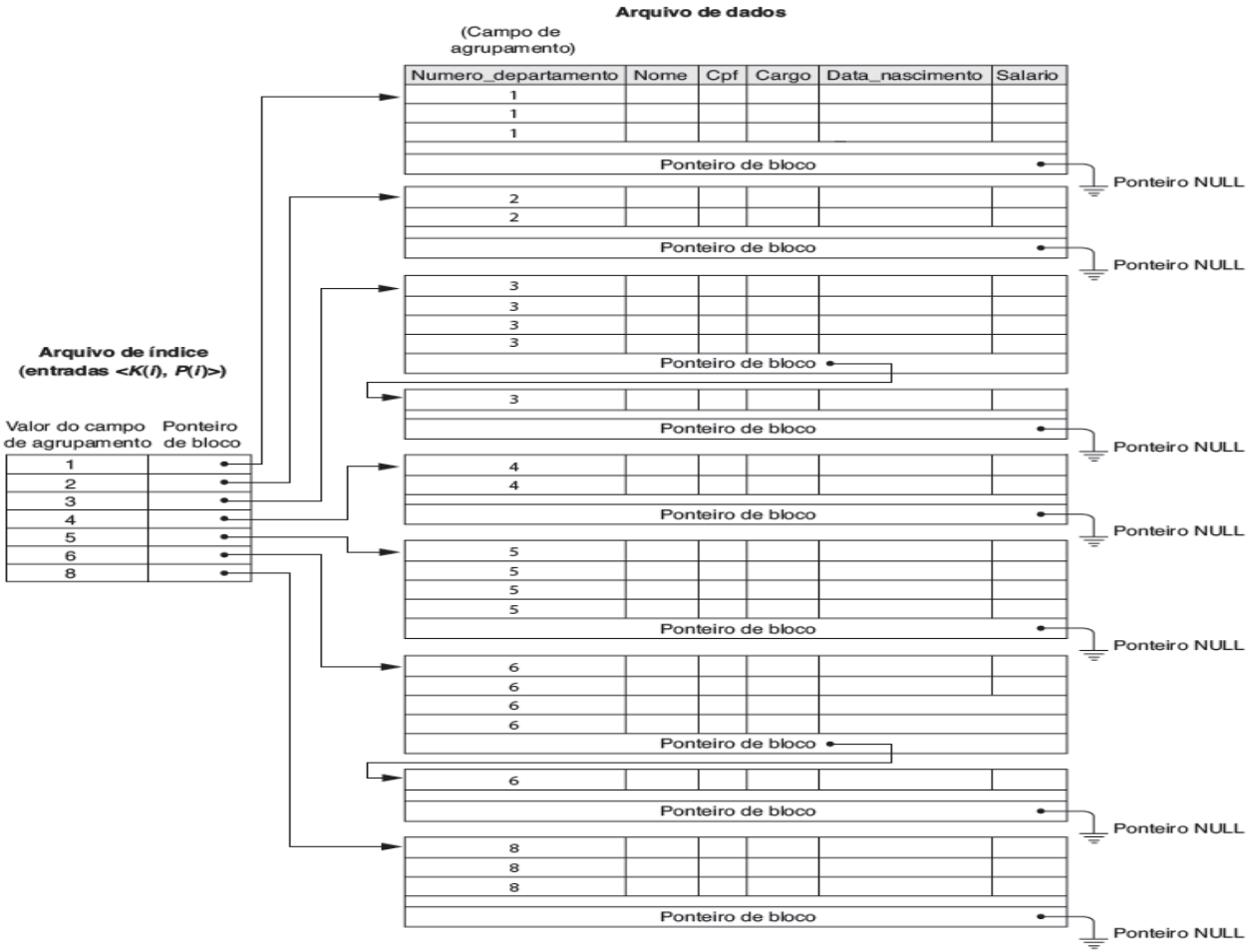

Figura 18.3
O índice de agrupamento com um cluster de bloco separado para cada grupo de registros que compartilham o mesmo valor para o campo de agrupamento.

Índices Secundários

Um índice secundário fornece um meio alternativo de acesso a um arquivo de dados para que algum acesso principal já existe.

O índice secundário pode ser criado em um campo que é uma chave candidata e tem um valor único em cada registro, ou em um campo não-chave com valores duplicados.

O índice secundário é também um arquivo ordenado com dois campos:

>> o primeiro campo é o mesmo tipo de dado do campo de indexação, que um campo não usado para a ordenação do arquivo de dados; e
>> o segundo campo é um ponteiro de bloco ou um ponteiro de registro.

Muitos índices secundários (e, portanto, os campos de indexação) podem ser criados para o mesmo arquivo de dados.

Índices Secundários

Primeiro caso:
O campo de indexação refere-se a um campo, que possui valor distinto para cada registro de dados.

No modelo relacional, esse campo seria qualquer atributo chave com valor único, ou até a chave primária da tabela.

Há uma entrada (registro) de índice para cada registo do arquivo de dados:
>> o valor do campo para o registro, e
>> um ponteiro para: (i) o bloco do registro de dados; ou o registro propriamente dito.

Neste primeiro caso, o **índice é denso**.

Índices Secundários

Primeiro caso.

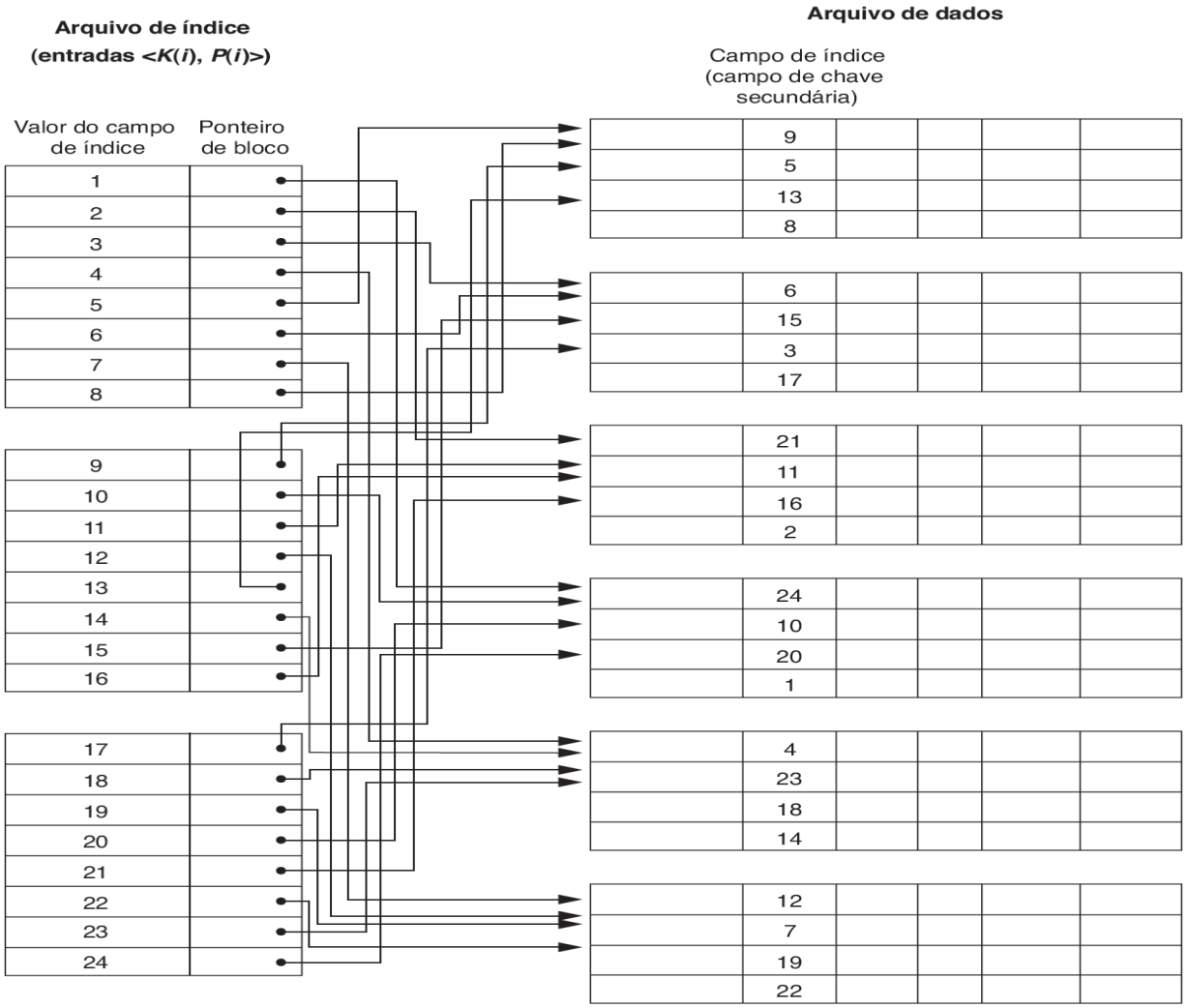

Figura 18.4
Um índice secundário denso (com ponteiros de bloco) em um campo de chave não ordenado de um arquivo.

Índices Secundários

Primeiro caso:
O campo de indexação refere-se a um campo, que possui valor distinto para cada registro de dados.

É possível realizar busca binária no arquivo de índice secundário ?
É possível realizar busca binária diretamente no arquivo de dados ?

Um índice secundário geralmente precisa de mais espaço de armazenamento e tempo de busca mais longo do que um índice primário, por causa de seu maior número de entradas.

Se o predicado de uma pesquisa referencia campos que não possuem índices (qualquer tipo de índice), ocorrerá uma busca linear.

Para um índice primário, ainda podemos usar uma pesquisa binária no arquivo principal (arquivo de dados), mesmo que o índice não existia.

Índices Secundários

Exemplo 2:

Considere o arquivo do Exemplo 1, com **r = 30.000** registros de comprimento fixo de tamanho **R = 100** bytes armazenados em um disco com o tamanho do bloco **B = 1024** bytes. O arquivo tem **b = 3000** blocos. Um ponteiro de bloco ocupa **P = 6** bytes.

Suponha que queremos procurar um registro com um valor específico para o campo de um índice secundário, que é **V = 9** bytes de tamanho.

Sem o índice secundário, a busca linear no arquivo exigiria **b / 2 = 3000/2 = 1500** blocos acessados em média.

Suponha que há um índice secundário. Cada entrada (registro) de índice tem **Ri = (9 + 6) = 15** bytes, seu fator de bloco é **bfri = ⌊ (B Ri /) = ⌋ ⌊ (1024/15) ⌋ = 68** entradas por bloco. O número total de entradas (registros) de índice é igual ao número de registos de dados, **ri = 30.000**. O número de blocos necessários para o índice, portanto, é **bi = ⌈ (ri / bfri) ⌉ = ⌈ (3000/68) ⌉ = 442** blocos.

Índices Secundários

Exemplo 2:

Uma pesquisa binária no índice secundário precisa acessar $\lceil (\log_2 bi) \rceil = \lceil (\log_2 442) \rceil = 9$ blocos.

Para buscar um registro usando o índice secundário, precisamos de um bloco adicional de acesso ao arquivo de dados: **9 + 1 = 10** blocos acessados.

Conclusões:

- o índice secundário proporciona uma grande melhoria em relação à pesquisa linear (10 contra 1500 blocos acessados);

- o índice primário é ligeiramente melhor que o índice secundário (7 contra 10 blocos acessados); esta diferença surge porque o índice primário é esparso, com apenas 45 blocos de comprimento.

Índices Secundários

Segundo caso:
O campo de indexação refere-se a um campo não chave, que também não é usado na ordenação do arquivo.

Nesse caso, podem existir vários registros de dados para um mesmo valor do campo de indexação.

Índices Secundários

Segundo caso:

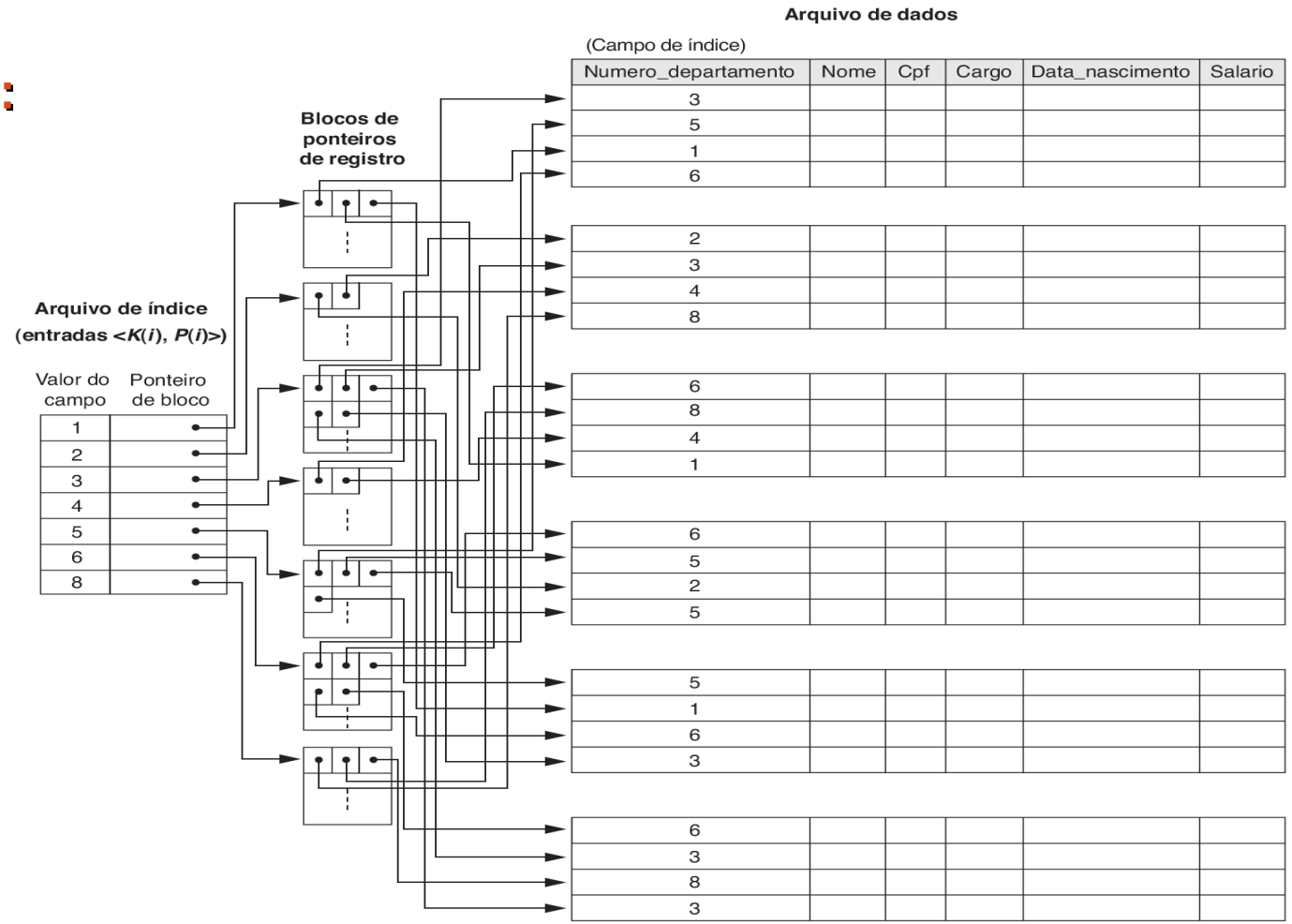

Figura 18.5
Um índice secundário (com ponteiros de registro) em um campo não chave implementado usando um nível de indireção, de modo que as entradas de índice são de tamanho fixo e têm valores de campo únicos.

Índices

Table 18.1 Types of Indexes Based on the Properties of the Indexing Field

	Index Field Used for Physical Ordering of the File	Index Field Not Used for Physical Ordering of the File
Indexing field is key	Primary index	Secondary index (Key)
Indexing field is nonkey	Clustering index	Secondary index (NonKey)

Índices

Table 18.2 Properties of Index Types

Type of Index	Number of (First-level) Index Entries	Dense or Nondense (Sparse)	Block Anchoring on the Data File
Primary	Number of blocks in data file	Nondense	Yes
Clustering	Number of distinct index field values	Nondense	Yes/no[a]
Secondary (key)	Number of records in data file	Dense	No
Secondary (nonkey)	Number of records[b] or number of distinct index field values[c]	Dense or Nondense	No

[a]Yes if every distinct value of the ordering field starts a new block; no otherwise.
[b]For option 1.
[c]For options 2 and 3.

Índices em Múltiplos Níveis

A ideia subjacente a um índice multinível é reduzir o espaço de busca.

Em uma estrutura de índice multinível, o índice de qualquer arquivo de dados é chamado de **primeiro nível** (ou nível base).

O índice que referencia o primeiro nível é chamando **segundo nível**; o índice que referencia o segundo nível é chamado **terceiro nível**, e assim por diante.

O fator de bloco *bfri* para o segundo nível, e para todos os níveis subsequentes, é o mesmo que o descrito para o primeiro nível, porque todas as entradas de índice são do mesmo tamanho, cada uma tem um valor de campo e um endereço de bloco.

Índices em Múltiplos Níveis

Se o primeiro nível tem **r1** entradas (registros), então esse nível possui **b1 = ⌈ (r1/bfri) ⌉** blocos.

O segundo nível possui **b1** entradas (registros), então esse nível possui **b2 = ⌈ (b1/bfri) ⌉** blocos.

O terceiro nível possui **b3 = ⌈ (b2/bfri) ⌉** blocos.

E assim por diante...

Índices em Múltiplos Níveis

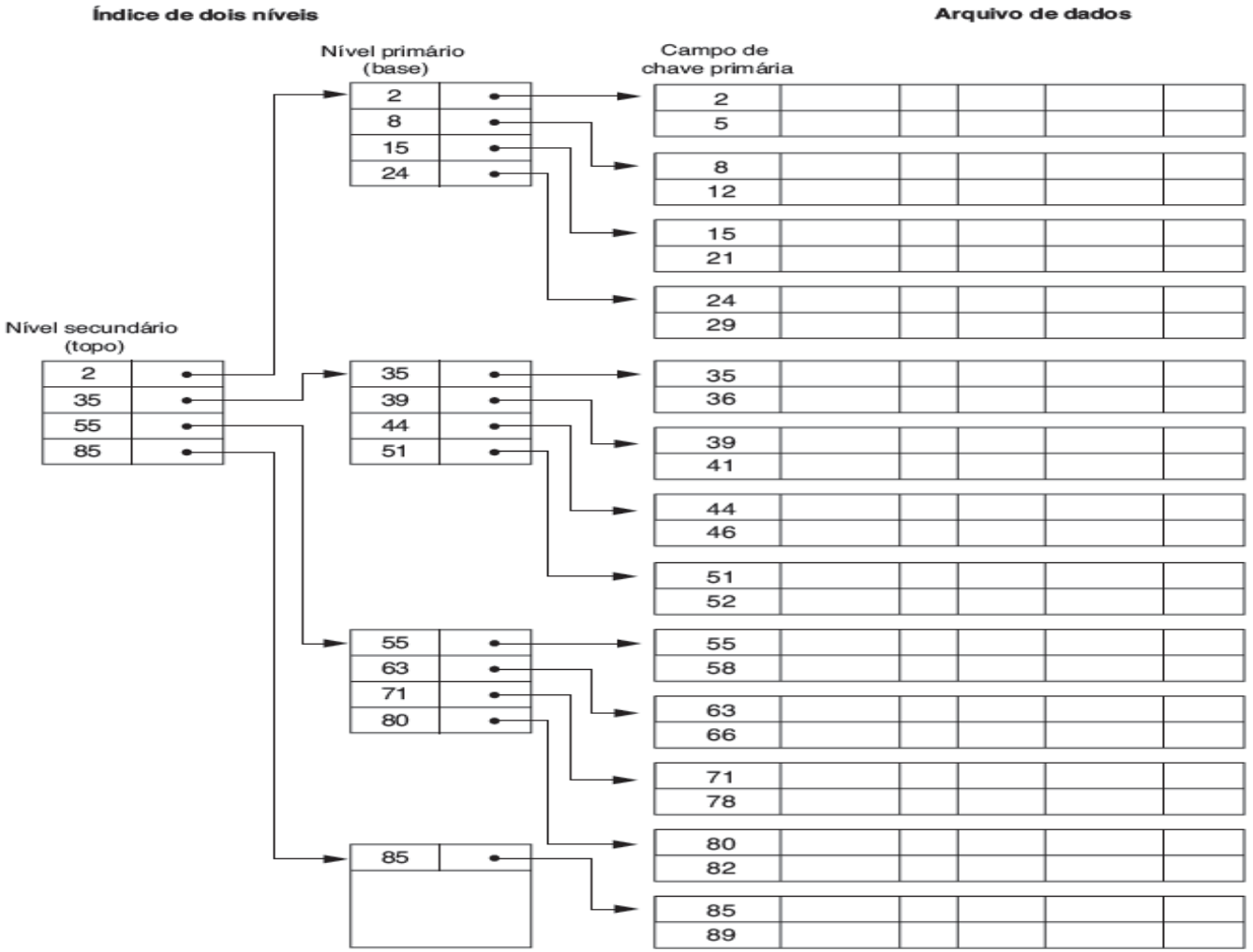

Figura 18.6
Um índice primário de dois níveis semelhante à organização ISAM (*Indexed Sequential Access Method*).

Índices em Múltiplos Níveis

Indices em múltiplos níveis reduz o número de blocos acessados, em relação a índices um nível único, quando a busca explora o campo de indexação.

Contudo, há um custo adicional para tratar com inserções e deleções no índice, pois todos os níveis são arquivos fisicamente ordenados.

Para amenizar tal problema, índices em múltiplos níveis podem:

>> deixar (alocar antecipadamente) algum espaço em cada bloco para a inserção de novas entradas;

>> usar estruturas e algoritmos dedicados (mais apropriados) para inserção/exclusão de blocos quando o arquivo de dados cresce ou encolhe; por exemplo, conhecendo-se estatísticas sobre os dados.

É comum usar estruturas dos tipos árvores B e B+.

Árvores de Busca

Uma **árvore de busca** é um tipo especial de árvore que é utilizada para orientar a busca de um registro, dado o valor de um dos campos do registro.

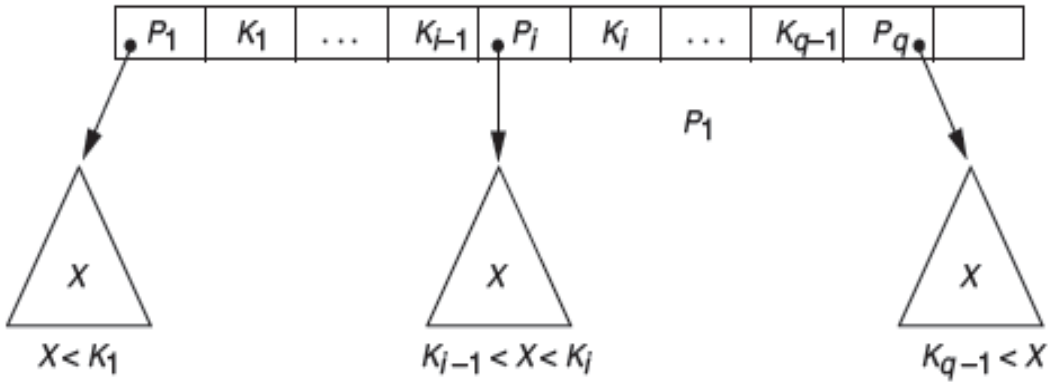

Figura 18.8

Um nó em uma árvore de pesquisa com ponteiros para subárvores abaixo dela.

Árvores de Busca

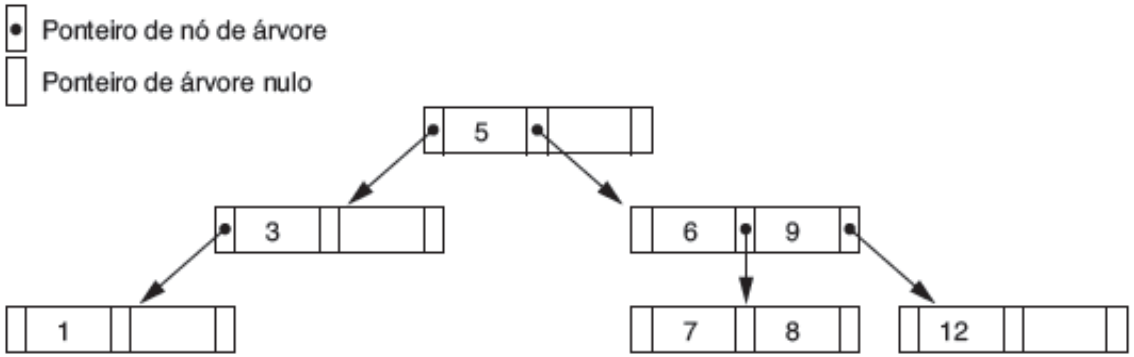

Figura 18.9

Uma árvore de pesquisa de ordem $p = 3$.

Índices usando Árvores B

Uma **árvore-B** (*B-tree*) é uma árvore de busca balanceada que tenta equilibrar o espaço perdido por exclusão.

Cada nó interno possui a estrutura

<P1, <K1, Pr1>, P2, <K2, Pr2>, ..., <Kq–1, Prq–1>, Pq>

Pi é um ponteiro para um nó filho na árvore-B;

Pri é um ponteiro para o registro cujo valor do campo chave de pesquisa é igual a Ki (ou ao bloco de arquivo de dados contendo esse registro).

Dentro de cada nó: K1 < K2 < ... < Kq−1.

Cada nó possui no máximo **q** ponteiros (árvore de ordem **q**).

Índices usando Árvores B

Cada nó, exceto o nó raiz, tem pelo menos ⌈ **(p / 2)** ⌉ ponteiros de árvore:

>> O nó raiz tem pelo menos dois ponteiros de árvores (a menos que seja o único nó na árvore).

Todos os nós folha são do mesmo nível. Nós folha têm a mesma estrutura dos nós internos, exceto que todos os seus ponteiros **Pi** para nós filhos são nulos.

Índices usando Árvores B

Deleção. Se a deleção de um valor resultar que um nó possua menos que ⌈ (p / 2) ⌉ ponteiros de árvore, tal nó é combinado com os seus nós vizinhos, e essa redução é propagada para cima, podendo chegar até o nó raiz. Assim, a deleção pode reduzir o número de níveis da árvore.

A Figura 18,10 (b) ilustra uma árvore-B de ordem p = 3. Observe que todas os valores na árvore-B são únicos, porque o exemplo apresenta uma estrutura de acesso em um campo de chave.

Se usarmos uma árvore-B em um campo não-chave, temos de mudar a definição do arquivo de ponteiros **Pri** para apontar para um bloco (ou um conjunto de blocos) que contem os ponteiros para os registros do arquivo de dados (nível extra de indireção).

Índices usando Árvores B

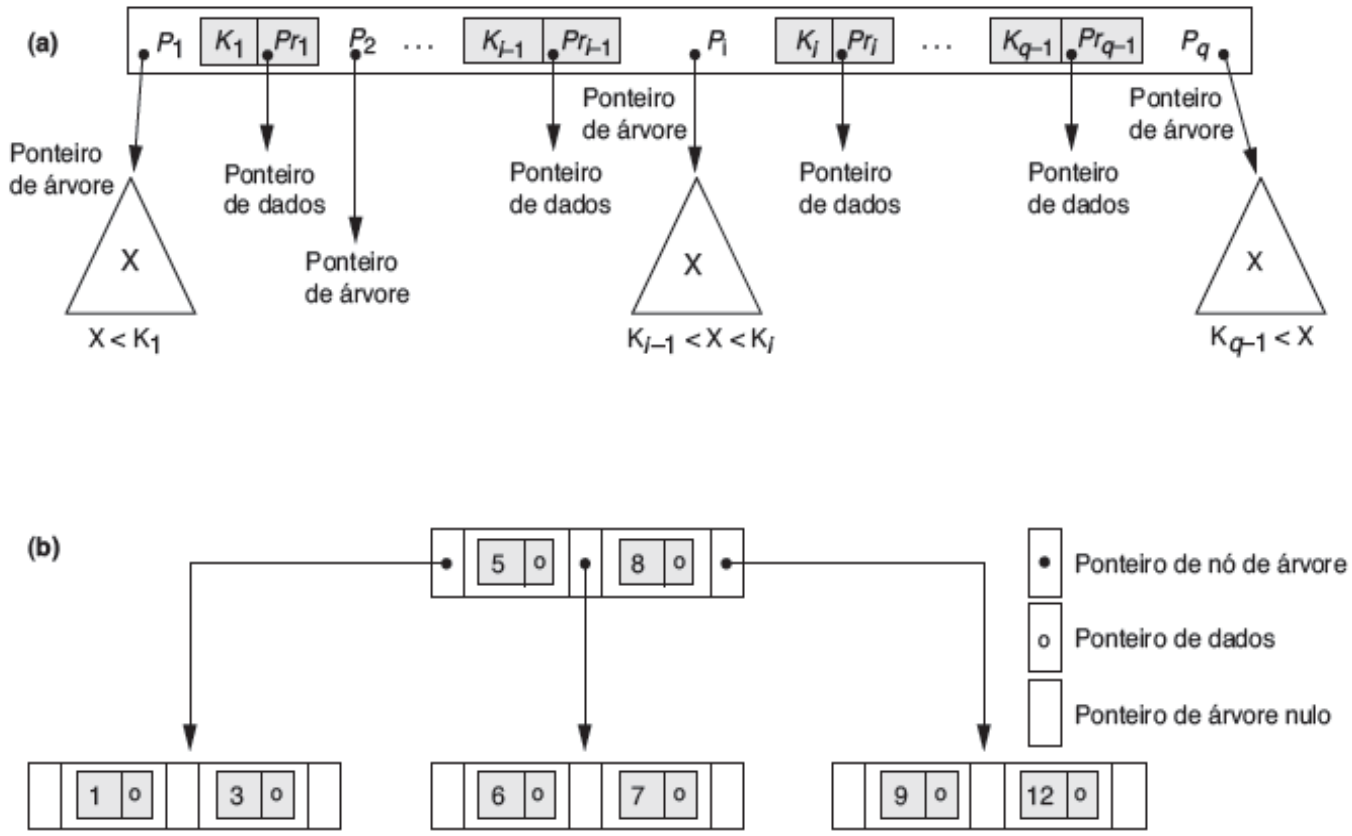

Figura 18.10

Estruturas B-tree. (a) Um nó em uma B-tree com $q - 1$ valores de pesquisa. (b) Uma B-tree de ordem $p = 3$. Os valores foram inseridos na ordem 8, 5, 1, 7, 3, 12, 9, 6.

Índices usando Árvores B

Exemplo 3:

Suponha que o campo de pesquisa é um campo chave que não ordena o arquivo de dados, a árvore-B possui p = 23.

Suponha que cada nó da árvore-B está 69% completo. Cada nó terá, em média, p * 0,69 = 23 * 0,69 (ou seja, aproximadamente, 16 ponteiros e, portanto, 15 valores de busca).

Raiz:	1 nó	15 valores	16 ponteiros
Nível 1:	16 nós	240 valores	256 ponteiros
Nível 2:	256 nós	3840 valores	4096 ponteiros
Nível 3:	4096 nós	61.440 valores	

Índices usando Árvores B+

Árvore-B+ é uma variação da árvore-B.

Na árvore-B+, os ponteiros de dados são armazenados apenas nos nós de folhas da árvore.

Os nós folha tem uma entrada para cada valor do campo de pesquisa, juntamente com um ponteiro de dados para o registro (ou do bloco que contém esse registro, se o campo de pesquisa for um campo chave).

Para um campo de pesquisa não-chave, o ponteiro aponta para um bloco que contém ponteiros para os registros do arquivo de dados (nível extra de indireção).

Índices usando Árvores B+

Cada nó interno tem a forma

<P1, K1, P2, K2, ..., Pq – 1, Kq –1, Pq>

onde **q ≤ p** e cada **Pi** é um ponteiro de árvore.

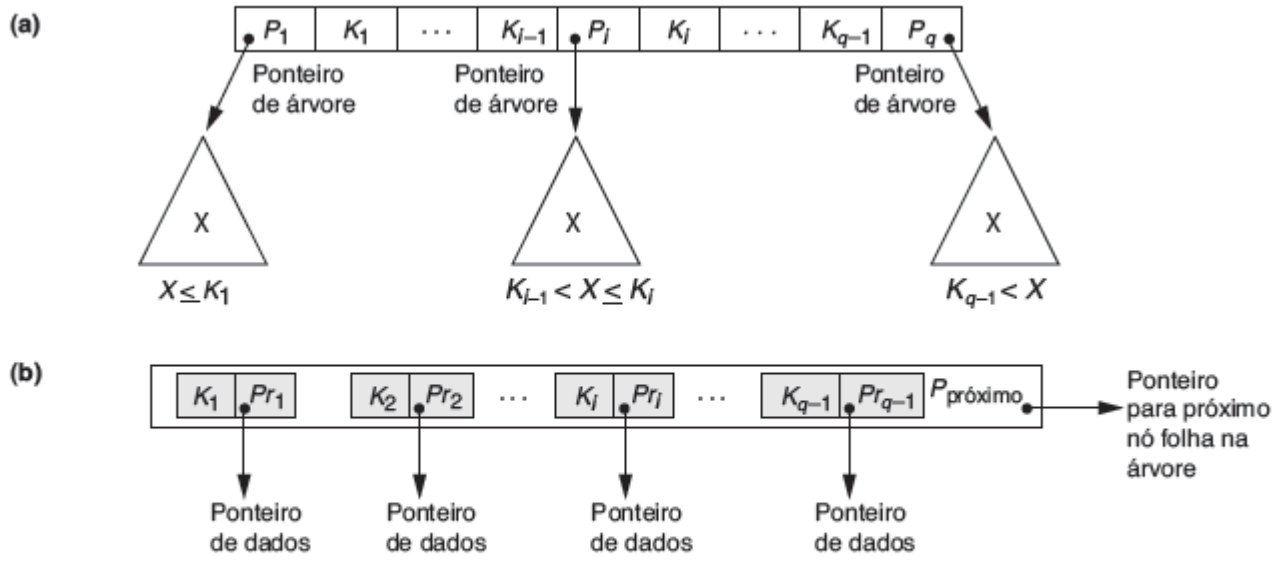

Figura 18.11

Os nós de uma B+-tree. (a) Nó interno de uma B+-tree com $q - 1$ valores de pesquisa. (b) Nó folha de uma B+-tree com $q - 1$ valores de pesquisa e $q - 1$ ponteiros de dados.

Índices usando Árvores B+

Cada nó folha tem a forma

<<K1, Pr1>, <K2, Pr2>, ..., <Kq–1, Prq–1>, Pnext>

onde **q ≤ p**, cada **Pri** is a ponteiro de dados, e **Pnext** aponta para o próximo nó folha.

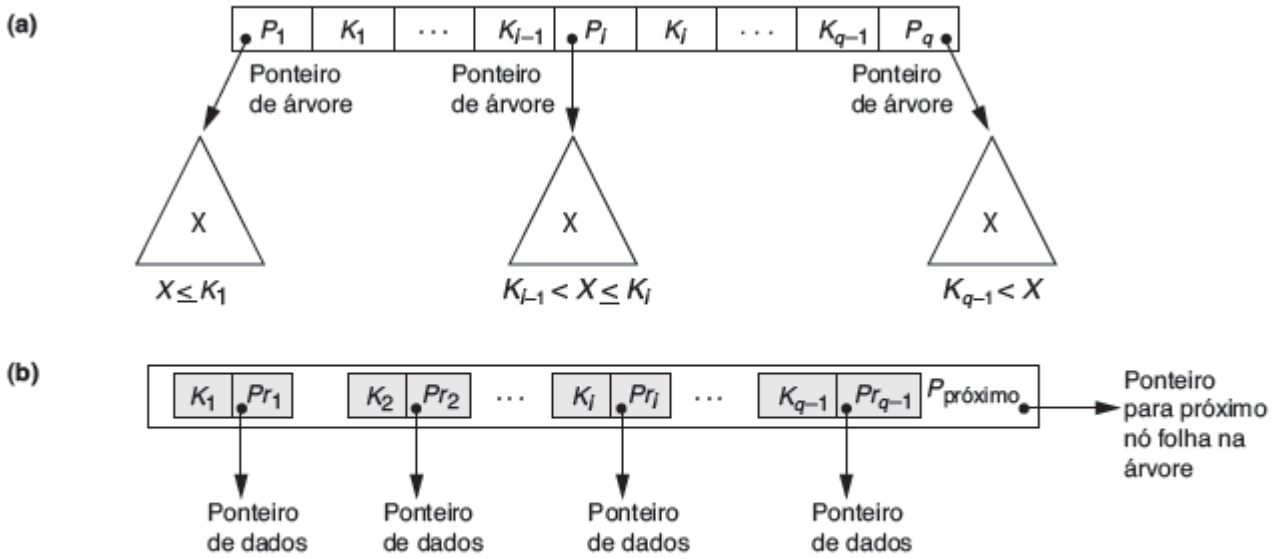

Figura 18.11

Os nós de uma B+-tree. (a) Nó interno de uma B+-tree com $q - 1$ valores de pesquisa. (b) Nó folha de uma B+-tree com $q - 1$ valores de pesquisa e $q - 1$ ponteiros de dados.

Índices usando Árvores B+

Exemplo 4:

Suponha que o campo-chave de busca é V = 9 bytes de comprimento, o tamanho do bloco é B = 512 bytes, um ponteiro de registro é Pr = 7 bytes, e um ponteiro de bloco é Pb = 6 bytes.

Um nó interno da árvore-B+ pode ter até ponteiros árvore **p** e (**p – 1**) valores de campo da pesquisa.

Assim, temos:

$$(p * Pb) + ((p - 1) * V) \leq B$$
$$(p * 6) + ((p - 1) * 9) \leq 512$$
$$(15 * p) \leq 521$$

na árvore-B+, o maior p é 34;
se fosse uma árvore-B, o maior p seria 23.

Índices usando Árvores B+

Exemplo 4:

Nos nós folha da árvore-B+ há o mesmo número de valores e de ponteiros de dados. Há, ainda, um ponteiro para o próximo nó folha:

$(pfolha * (Pr + V)) + Pb \leq B$
$(pfolha * (7 + 9)) + 6 \leq 512$
$(16 * pfolha) \leq 506$

Cada nó folha pode conter até **pfolha = 31** combinações de valores-chave de dados / ponteiro (**<Ki, Pri>**), assumindo que os ponteiros de dados são ponteiros de registros.

Índices usando Árvores B+

Exemplo 5:

Considere os dados do Exemplo 4. Para calcular o número aproximado de entradas na árvore-B+, assume-se que cada nó está 69% completo.

Em média, cada nó interno terá **34 * 0,69** ponteiros, e **22** valores.

Cada nó folha, em média, terá **0,69 * pfolha = 0,69 * 31** ou aproximadamente **21** ponteiros de registros de dados:

Raiz:	1 nó	22 valores	23 ponteiros
Nível 1:	23 nós	506 valores	529 ponteiros
Nível 2:	529 nós	11.638 valores	12.167 ponteiros
Nível Folha:	12.167 nós	255.507 ponteiros para registros	

255.507 (árvore-B+) > 65.535 (árvore-B)

Índices usando Árvores B+

Figura 18.12
Exemplo de uma inserção em uma B+-tree com $p = 3$ e $p_{folha} = 2$.

Índices usando Árvores B+

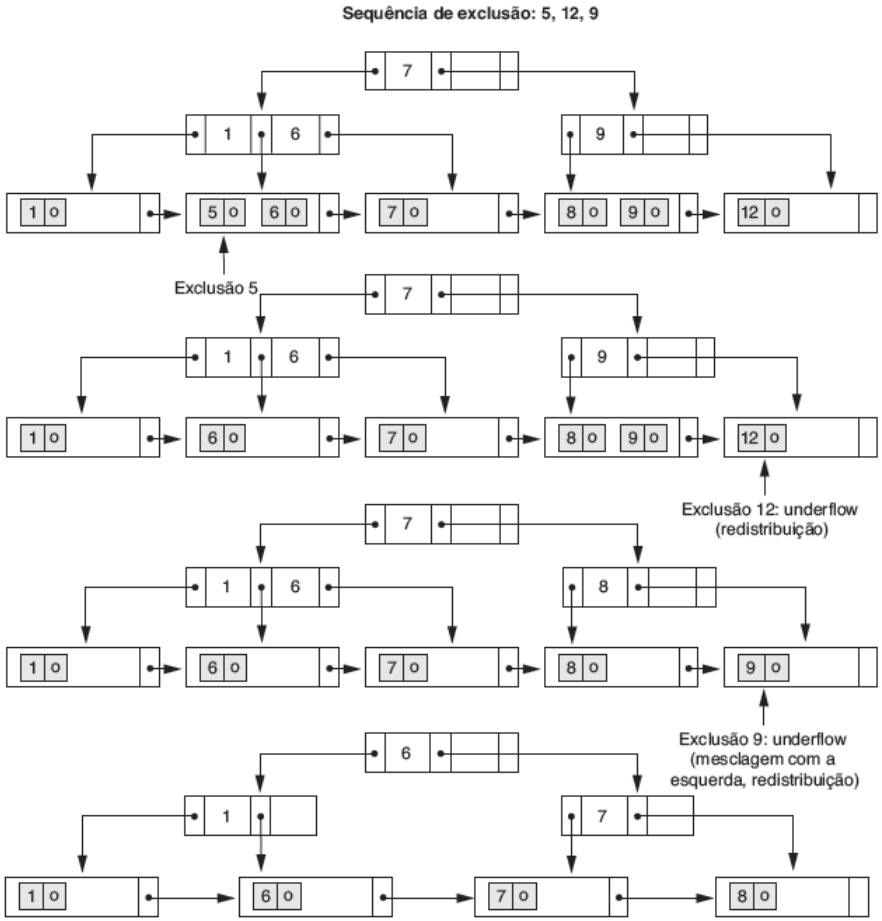

Figura 18.13
Um exemplo de exclusão de uma B+-tree.