

UTFPR - Universidade Tecnológica
Federal do Paraná

Estruturas de índices

Material adaptado de
Leyza Baldo Dorini



Programação da aula

- * Conceitos;
- * Estruturas de Índices;
- Tipos de Índices:
 - Primários;
 - Secundários;
 - Clustering;
 - Multiníveis;
 - Árvore B e B+;
 - Hash;
- Exemplos

Próxima aula: Índices no MySQL/Postgres



Índices

- Da aula passada:
 - arquivos com organizações primárias:
 - * desordenada (heap)
 - * ordenada
 - * clustering
 - ...

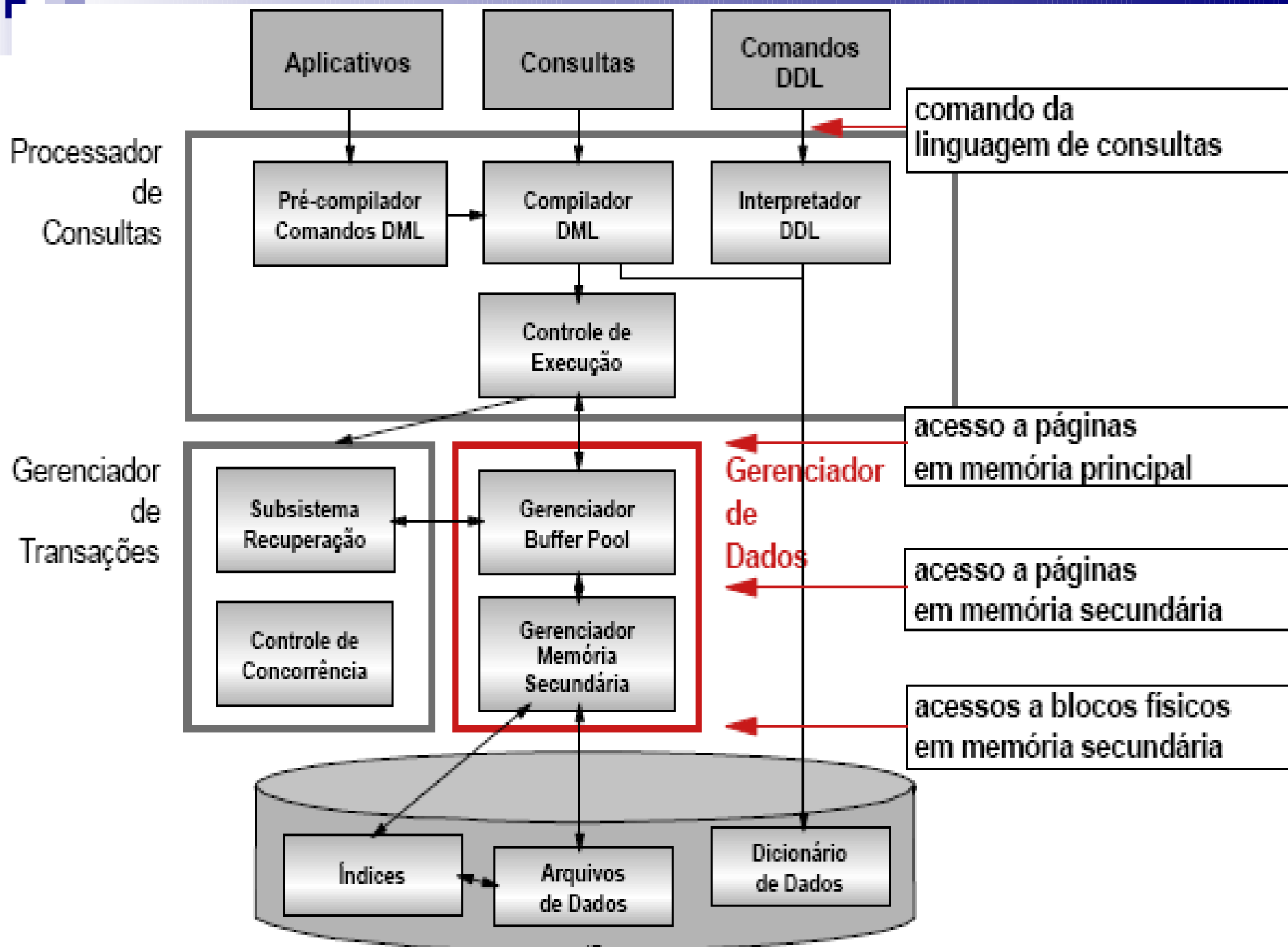


Índices

- Nesta aula:

Índices: estruturas de acesso adicionais auxiliares.

- aumentam a velocidade na recuperação de registros em certas condições de busca.
- caminhos de acesso secundário.
- não afetam a posição física dos registros





Índices - Conceitos

→ Índices: acesso a registros baseado em certos campos, chamados campos (**atributos**) de indexação.



Índices - Conceitos

- Índices: acesso a registros baseado em certos campos, chamados campos (atributos) de indexação.
- **qualquer campo** da relação pode ser utilizado para criar um índice.



Índices - Conceitos

- Índices: acesso a registros baseado em certos campos, chamados campos (atributos) de indexação.
- qualquer campo da relação pode ser utilizado para criar um índice.
- mais que um campo da relação pode ser utilizado (**índices múltiplos**)

Índices - Conceitos

- Índices: acesso a registros baseado em certos campos, chamados campos (atributos) de indexação.
- qualquer campo da relação pode ser utilizado para criar um índice.
- mais que um campo da relação pode ser utilizado (índices múltiplos)
- Um índice **permite localizar** um registro sem ter que examinar mais que uma pequena fração dos registros possíveis;
- O(s) campo(s) cujos valores o índice se baseia formam a **chave de pesquisa**;



Índices

- São um modo comum de **melhorar o desempenho** do BD
- Permite encontrar e recuperar tuplas específicas muito **mais rapidamente**
- Como também **produzem trabalho adicional** para o SGBD, devem ser usados com sensatez

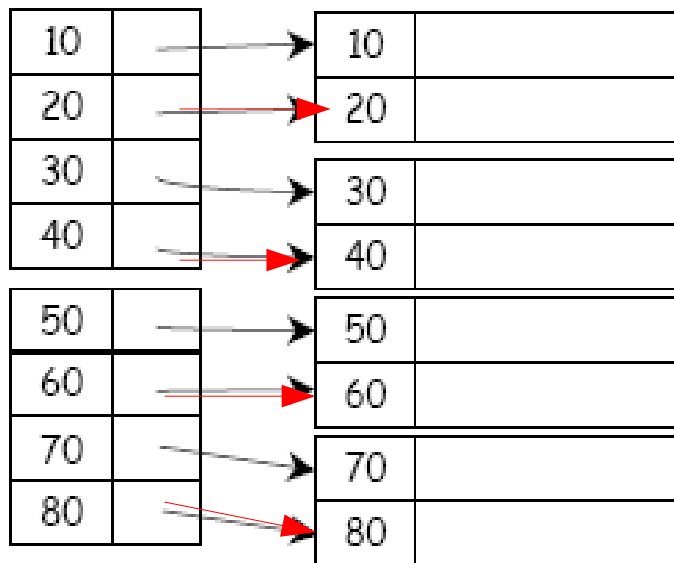


Estruturas de índices

- **Índice primário** - baseado na chave primária de ordenação;
- **Índice de agrupamento** (clustering) - baseado no campo de ordenação não-chave de um arquivo;
- **Índice secundário** - baseado em qualquer campo não ordenado de um arquivo;
- **Índices multiníveis;**
- **Árvores B e B+;**
- **Tabelas Hash;**

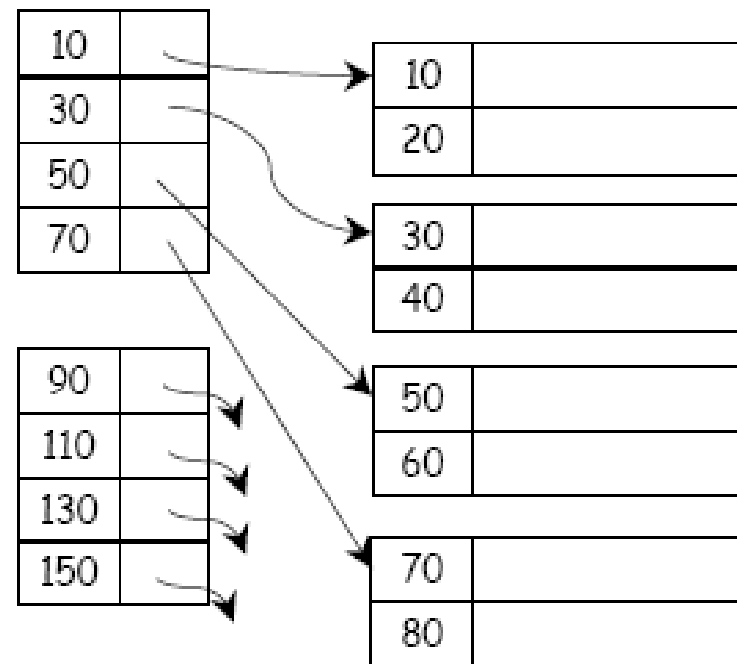
Índices sobre arquivos sequenciais (tipos de índices)

Densos: uma entrada no arquivo de índices p/cada registro no arquivo de dados



Um índice denso sobre um arquivo de dados sequenciais

Esparsos: apenas alguns registros de dados são representados no arquivo de índices



Um índice esparsos sobre um arquivo de dados sequenciais

Tipos de índices

- **Índice Denso**
 - Seqüência de blocos contendo apenas as chaves dos registros e os ponteiros para os próprios registros
 - Índice denso = (chave-ponteiro, registro)
- **Índice Esparso**
 - Usa **menos espaço** de armazenamento que o índice denso ao **custo** de um tempo um pouco maior para localizar um registro dada a sua chave
 - Índice esparsa = (chave-ponteiro, blocos de dados)

1) Índices primários

- Arquivo ordenado cujos registros são de tamanho fixo e contém 2 campos.
 - O primeiro é do mesmo tipo de dados do campo chave de classificação (chamado chave primária) do arquivo de dados
 - O segundo é um inteiro para um bloco de disco (endereço de bloco)
- Há uma entrada de índice (ou registro de índice) para cada bloco do arquivo de dados

ARQUIVO DE DADOS

(CAMPO-CHAVE
PRIMÁRIO)

NOME

SSN

DATANASC

CARGO

SALARIO

SEXO

Aaron, Ed					
Abbott, Diane					
⋮					
Acosta, Marc					

Adams, John					
Adams, Robin					
⋮					
Akers, Jan					

Alexander, Ed					
Alfred, Bob					
⋮					
Allen, Sam					

Allen, Troy					
Anders, Keith					
⋮					
Anderson, Rob					

Anderson, Zach					
Angeli, Joe					
⋮					
Archer, Sue					

Arnold, Mack					
Arnold, Steven					
⋮					
Atkins, Timothy					

ARQUIVO DE ÍNDICE
(entradas <K(i),P(i)>)VALOR DA
CHAVE PRIMÁRIA
DA ÂNCORA
DO BLOCOPONTEIRO DO
BLOCO

Aaron, Ed		•
Adams, John		•
Alexander, Ed		•
Allen, Troy		•
Anderson, Zach		•
Arnold, Mack		•
⋮		

⋮

⋮		
Wong, James		•
Wright, Pam		•

⋮

1) Índices primários

- O número de entradas no índice é o mesmo número de blocos de disco do arquivo de dados ordenado
- O primeiro registro em cada bloco do arquivo de dados é chamado de **registro âncora** de bloco (âncora do bloco)
- Um índice primário é um **índice esparsa**

1) Índices primários

- Vantagens:
 - Há **menos entradas** de índice que registros no arquivo de dados
 - Cada entrada de índice é tipicamente menor em tamanho que o registro de dados (apenas 2 campos). Assim, uma busca binária no arquivo de índice exige **menos acessos** a bloco que uma busca binária no arquivo de dados.

Exemplo (busca sem índice)

- $r = 30000$ (número de registros)
- $B = 1024$ (tamanho do bloco em bytes)
- $R = 100$ bytes (tamanho do registro)
- Registros por bloco (brf):
$$\text{brf} = \text{floor}(B/R) = \text{floor}(1024/100) = 10$$

* Blocos necessários:

$$b = \text{ceil}(r/\text{brf}) = \text{ceil}(30000/10) = 3000$$

Busca binária exige aproximadamente $\text{ceil}(\log_2 b) = \text{ceil}(\log_2 3000) = 12$ acessos a blocos

Exemplo (busca com índice)

- $r_i = 3000$ (número total de entradas de índice é igual ao número de blocos)
- $B = 1024$ (tamanho do bloco)
- $R_i = (9 \text{ bytes para campo da chave de classificação} + 6 \text{ bytes para ponteiro do bloco}) = 15 \text{ bytes}$
- $brfi = \text{floor}(B/R_i) = \text{floor}(1024/15) = 68$ entradas por bloco (fator divisão por blocos)
- $b_i = \text{ceil}(r_i/brfi) = \text{ceil}(3000/68) = 45$ blocos necessários

Busca binária exige aproximadamente $\text{ceil}(\log b) = \text{ceil}(\log_2 45) = 6$ acessos a bloco + 1 acesso adicional ao bloco do arquivo de dados

1) Índices primários

- Pelo exemplo, percebe-se a **redução no número de acesso** a blocos necessários
- Maior **problema: inclusão e exclusão** de registros (como em qualquer arquivo ordenado).
 - Com um índice primário, o problema é composto porque, se tentarmos incluir um registro em sua posição correta no arquivo de dados, deve-se não apenas mover o registro para atribuir espaço, mas também **alterar algumas estruturas de índice** (alguns âncoras podem mudar)

2) Índices clustering

- Se os registros de um arquivo são fisicamente **ordenados segundo um campo que não seja um campo chave** (ie, não possui valores distintos) esse campo é chamado de campo de clustering
- Podemos criar um índice de clustering para aumentar a velocidade de recuperação de registros que tenham o **mesmo valor** para o campo de clustering

2) Índices clustering

- Também é um **arquivo ordenado com 2 campos**: um do mesmo tipo do campo de clustering do arquivo de dados e o segundo é um ponteiro de bloco
- Há um entrada no índice de clustering para cada valor distinto do campo de clustering contendo o valor e o ponteiro para o primeiro registro com aquele valor para o campo de clustering

(CAMPO DE
CLUSTERING)

NUM_DEPARTAMENTO NOME SSN CARGO DATANASC SALARIO

1					
1					
1					
2					

2					
3					
3					
3					

3					
3					
4					
4					

5					
5					
5					
5					

6					
6					
6					
6					

6					
8					
8					
8					

ARQUIVO DE ÍNDICE
(entradas <K(i), P(i)>)VALOR DO CAMPO
DE CLUSTERINGPONTEIRO
DE BLOCO

1		•
2		•
3		•
4		•
5		•
6		•
8		•

2) Índices clustering

- Para **aliviar o problema da inclusão/exclusão** é comum reservar um (conjunto) de blocos para cada valor do campo de clustering
- Todos os registros são dispostos naquele bloco, tornando a **inclusão/exclusão relativamente diretas**

(CAMPO DE
CLUSTERING)

ARQUIVO DE DADOS

NUM_DEPARTAMENTO NOME SSN CARGO DATANASC SALARIO

1					
1					
1					
ponteiro de bloco					

ponteiro nulo

2					
2					
ponteiro de bloco					

ponteiro nulo

3					
3					
3					
3					
ponteiro de bloco					

3					
ponteiro de bloco					

ponteiro nulo

4					
4					
ponteiro de bloco					


ponteiro nulo

5					
5					
5					
5					

ARQUIVO DE ÍNDICE
(entradas <K(i), P(i)>)

VALOR DO
CAMPO DE
CLUSTERING PONTEIRO
DE BLOCO

1	
2	
3	
4	
5	
6	
8	



Fatores de eficiência no uso de índices

- O número de blocos de índices em geral é **pequeno** quando comparado com o número de blocos de dados;

Fatores de eficiência no uso de índices

- O número de blocos de índices em geral é **pequeno** quando comparado com o número de blocos de dados;
- Tendo em vista que as chaves são ordenadas, a **pesquisa é rápida** (pode-se usar um algoritmo de pesquisa binária);

Fatores de eficiência no uso de índices

- O número de blocos de índices em geral é **pequeno** quando comparado com o número de blocos de dados;
- Tendo em vista que as chaves são ordenadas, a **pesquisa é rápida** (pode-se usar um algoritmo de pesquisa binária);
- O índice pode ser pequeno o bastante para ser **mantido permanentemente em buffers da memória principal**. Nesse caso, uma pesquisa para uma determinada chave envolve apenas acessos à memória principal, sem precisar de operação de I/O.

3) Índices secundários

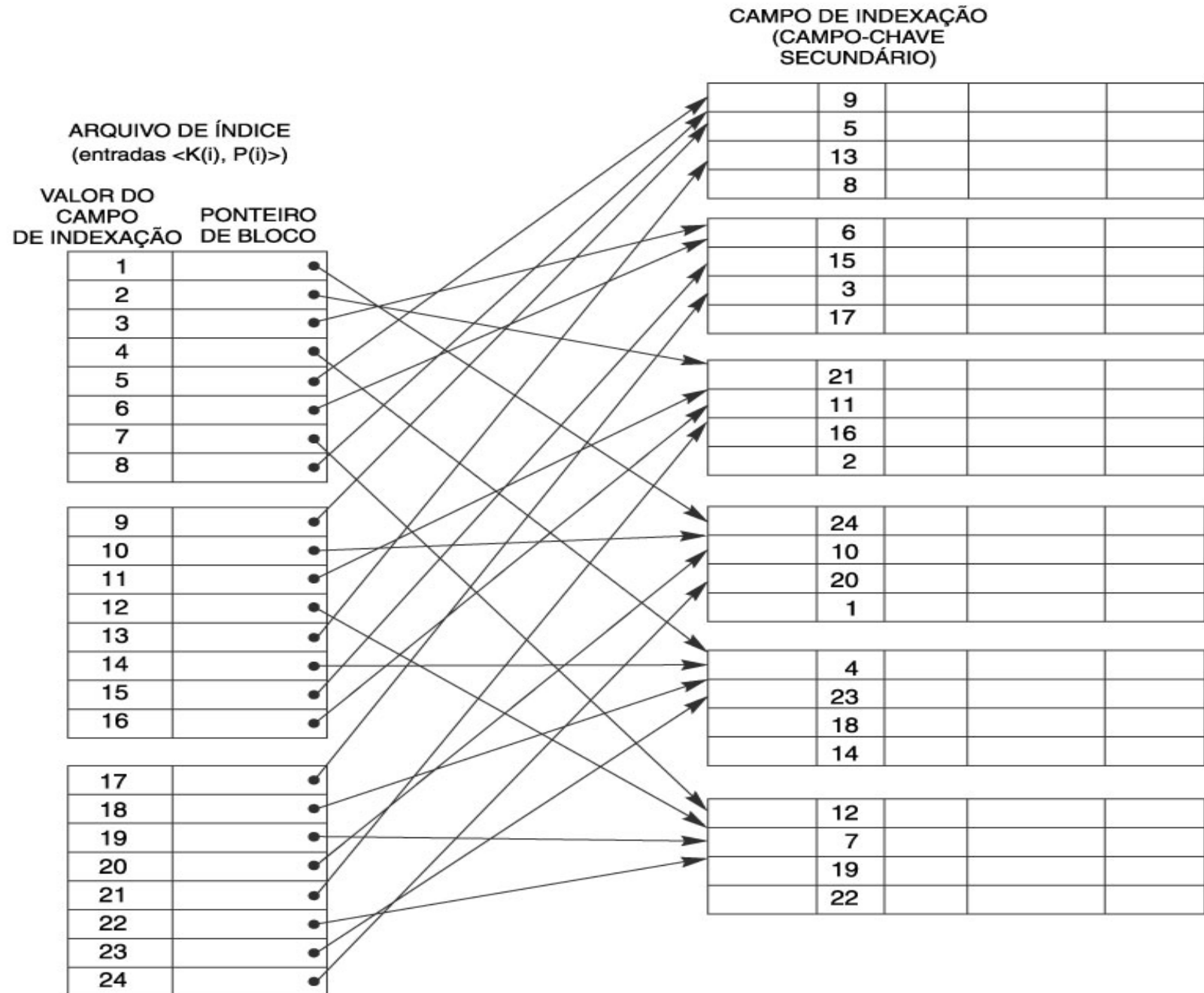
- Fornece um **meio secundário de acesso** a um arquivo para o qual já existe algum acesso primário
- Índice secundário é um arquivo ordenado baseado em 2 campos:
 - Do mesmo tipo de dados de algum campo que não seja o de classificação do arquivo de dados que é um campo de indexação
 - Ponteiro de bloco ou ponteiro de registro

3) Índices secundários

- Em outras palavras: o **índice secundário** pode ser usado sobre um campo que é uma **chave candidata** (que possui um valor único em cada registro) ou sobre um campo que não é chave e que possui valores duplicados
- Um **índice secundário** em uma **chave candidata** se parece com um **índice primário denso**, **exceto** pelo fato de que os registros apontados por valores sucessivos no índice **não estão armazenados sequencialmente**

3) Índices secundários

- Primeiro, consideremos uma estrutura de acesso de índice secundário em um campo chave. Tal campo é comumente denominado chave secundária.
- Há uma entrada de índice para cada registro do arquivo de dados, que contém o valor da chave secundária no registro e um **ponteiro** para o **bloco** no qual o **registro** está armazenado **ou** para o **próprio registro**.
- Assim, tal índice é denso



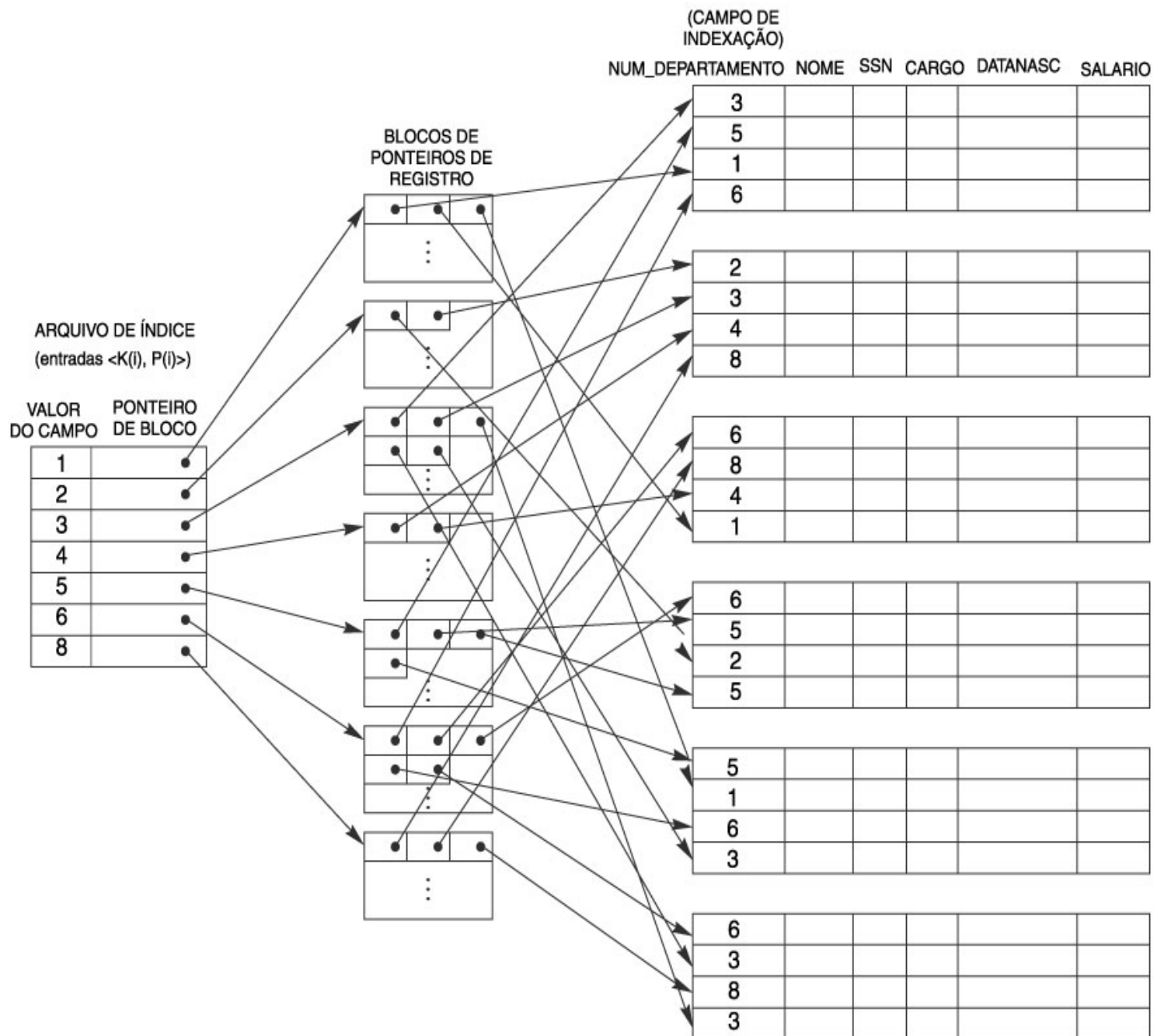
Exemplo (busca sem índice)

- $r = 30000$ (número de registros)
- $B = 1024$ (tamanho do bloco em bytes)
- $R = 100$ bytes (tamanho do registro)
- Blocos necessários:

$$b = \text{ceil}(r/brf) = \text{ceil}(30000/10) = 3000$$

Pesquisa linear gastaria quanto em média?

E com índices secundários? Suponha uma chave com 9 bytes e um ponteiro com tamanho de 5 bytes.



Tipos de índices

	Campo de Ordenação	Campo Não Ordenado
Campo chave	Índice Primário	Índice Secundário (Chave)
Campo Não Chave	Índice de Agrupamento	Índice Secundário (Não chave)

Propriedades dos índices

Tipo de Índice	Número de Entradas	Denso ou Esparso
Primário	Número de blocos no arquivo de dados	Esparso
Agrupamento	Número de valores distintos do campo de indexação	Esparso
Secundário (chave)	Número de registros no arquivo de dados	Denso
Secundário (não chave)	Número de registros no arquivo de dados ou	Denso
	Número de valores distintos do campo de indexação	Esparso



4) Índices de múltiplos níveis

- Até aqui, discutimos esquemas de indexação que envolviam **um arquivo de índice ordenado**

4) Índices de múltiplos níveis

- Até aqui, discutimos esquemas de indexação que envolviam um arquivo de índice ordenado
- **Busca binária** é aplicada ao índice para localizar os ponteiros para um bloco do disco ou para um registro do arquivo que possua um valor específico no campo de indexação

4) Índices de múltiplos níveis

- Até aqui, discutimos esquemas de indexação que envolviam um arquivo de índice ordenado
- Busca binária é aplicada ao índice para localizar os ponteiros para um bloco do disco ou para um registro do arquivo que possua um valor específico no campo de indexação
- Busca binária exige aproximadamente (\log_{bi}) acessos a blocos em índices com bi blocos



4) Índices em múltiplos níveis

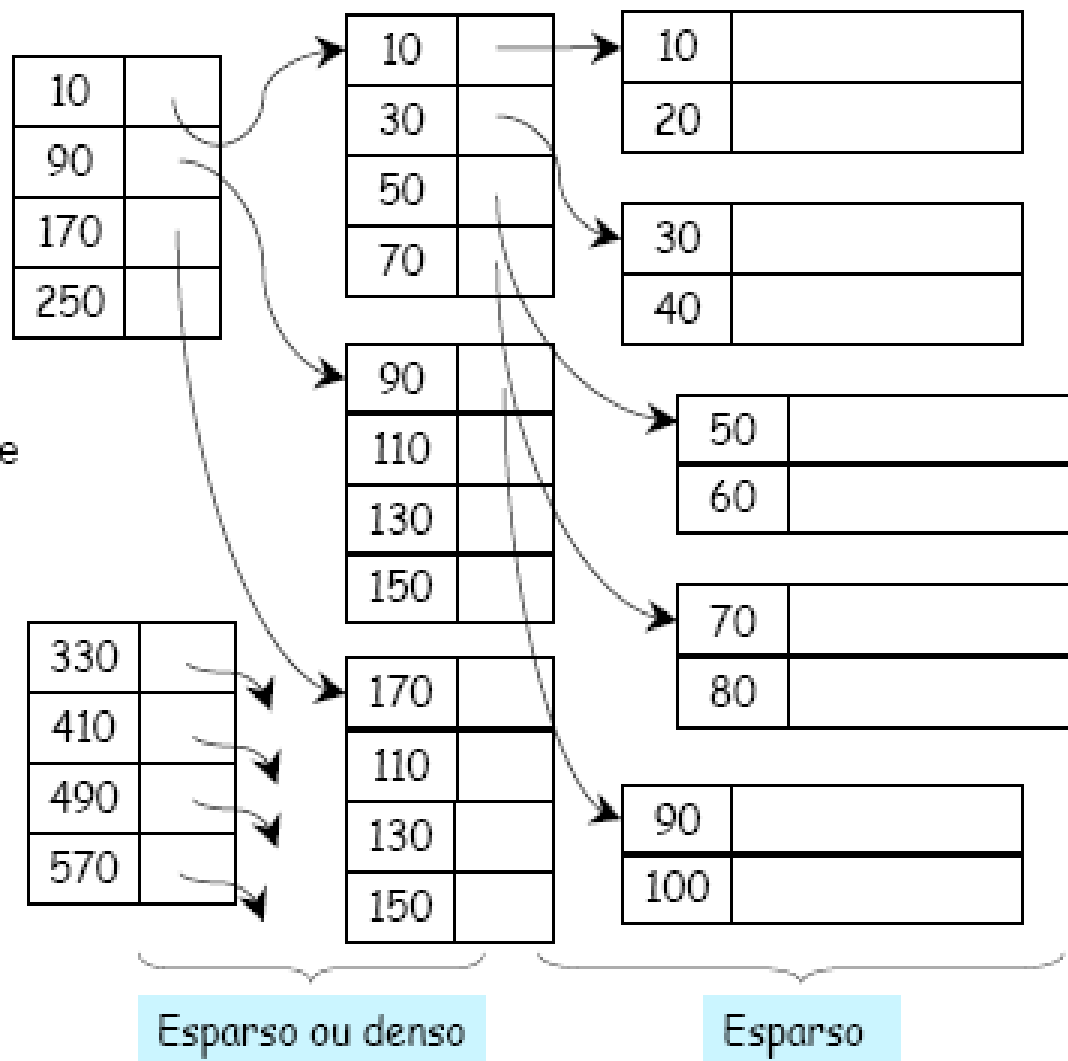
- A idéia de um índice multi-nível é **reduzir a parte do arquivo de índice** na qual devemos prosseguir a busca, subdividindo-o.

4) Índices em múltiplos níveis

- A idéia de um índice multi-nível é reduzir a parte do arquivo de índice na qual devemos prosseguir a busca, subdividindo-o.
- Para utilizar este esquema, o primeiro nível deve possuir valores distintos para as entradas e elas devem ser de tamanho fixo

4) Índices de múltiplos níveis

- Motivação: se o arquivo de índices se torna muito grande para ser armazenado em bloco de disco, é interessante indexá-lo em mais de um nível
- Vantagem: índice pequeno pode ser mantido em memória e o tempo de busca é mais baixo
- Desvantagem: muitos níveis de índices podem aumentar a complexidade do sistema (talvez seja melhor usar a árvore-B)



4) Índices de múltiplos níveis

Um índice multinível é um “Índice de índice”.

- Primeiro nível: **arquivo ordenado pela chave de indexação**, valores distintos, entradas de tamanho fixo.

4) Índices de múltiplos níveis

Um índice multinível é um “Índice de índice”.

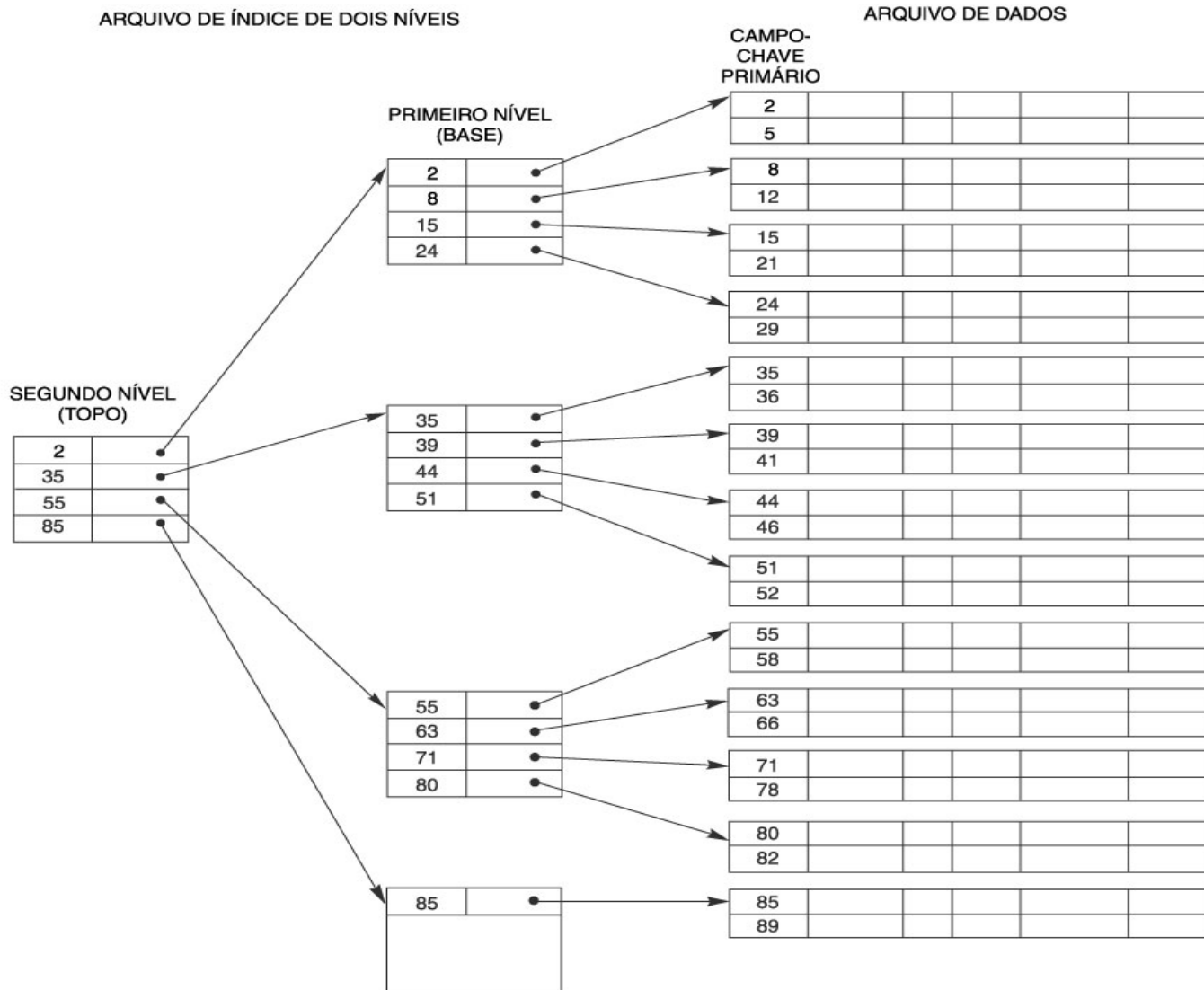
- Primeiro nível: arquivo ordenado pela chave de indexação, valores distintos, entradas de tamanho fixo.
- Demais níveis: **índice primário sobre o índice do nível anterior** e assim sucessivamente até que no último nível o índice ocupe apenas um bloco.

4) Índices de múltiplos níveis

Um índice multinível é um “Índice de índice”.

- Primeiro nível: arquivo ordenado pela chave de indexação, valores distintos, entradas de tamanho fixo.
- Demais níveis: índice primário sobre o índice do nível anterior e assim sucessivamente até que no último nível o índice ocupe apenas um bloco.
- **Número de acessos a bloco: um a cada nível de índice, mais um ao bloco do arquivo de dados**

4) Índices de múltiplos níveis



4) Índices de múltiplos níveis

- Problema dos índices multiníveis: índices são arquivos fisicamente ordenados, portanto, **ineficientes na inserção e remoção.**
- Solução?

4) Índices de múltiplos níveis

- Problema dos índices multiníveis: índices são arquivos fisicamente ordenados, portanto, ineficientes na inserção e remoção.
- Solução:
 - Índice multi-nível dinâmico, frequentemente implementado por meio de estruturas tais como **árvores B** e **árvores B+** (e suas variações)



Ver slides sobre árvores B.

5) Hashing

- Em uma organização de arquivo em hashing, **obtemos diretamente o endereço do bloco de disco** que contém um registro desejado por meio da aplicação de uma função sobre o valor da chave de procura do registro

5) Hashing

- Em uma organização de arquivo em hashing, **obtemos diretamente o endereço do bloco de disco** que contém um registro desejado por meio da aplicação de uma função sobre o valor da chave de procura do registro
- Seja K o conjunto de valores de chave e B os endereços de buckets. **Uma função hash h é uma função de K para B**

5) Hashing

- As **piores funções** hash possíveis mapeiam todos os valores da chave de procura para o mesmo bucket
- Idealmente, a função hash satisfaz:
 - A **distribuição é uniforme**
 - A **distribuição é aleatória** (o valor de hash não está correlacionado a qualquer ordem visível externamente dos valores de chave de procura)

5) Hashing (exemplo)

- As funções hash típicas executam **cálculos sobre a representação binária interna**. Ex: soma das representações binárias dos caracteres de uma chave e retorna o módulo da soma pelo número de buckets

5) Índices hash

- Um índice hash **organiza as chaves de procura**, com seus ponteiros associados, em uma estrutura de arquivo hash
- A função hash do exemplo a seguir calcula a soma dos dígitos da conta e retorna o módulo 7 da soma. O índice de hash possui 7 buckets, cada um de tamanho 2

Exemplo

bucket 0

bucket 1

A-215	
A-305	

bucket 2

A-101	
A-110	

bucket 3

A-217	
A-102	

A-201	

bucket 4

A-218	

bucket 5

bucket 6

A-222	

A-217	Brighton	750
A-101	Downtown	500
A-110	Downtown	600
A-215	Mianus	700
A-102	Perryridge	400
A-201	Perryridge	900
A-218	Perryridge	700
A-222	Redwood	700
A-305	Round Hill	350

5) Hashing estático: problemas

- **Overflow de buckets** (buckets insuficientes e desequilíbrio)
- Para hashing estático, 3 opções:
 - Escolher função hash com base no tamanho atual do arquivo;
 - Ou em uma previsão de tamanho
 - Reorganizar a estrutura hash periodicamente em resposta ao crescimento do arquivo



5) Hashing dinâmico

- Permite **modificar a função hash dinamicamente** para acomodar o crescimento ou a diminuição do BD
- Exemplo: hashing expansível: trata as mudanças no tamanho do BD através da divisão e fusão de buckets



Comparação entre indexação ordenada e hashing

- Quais tipos de consultas tem maior probabilidade de serem propostas?

Comparação entre indexação ordenada e hashing

- Quais tipos de consultas tem maior probabilidade de serem propostas?
 - Para as consultas da forma a seguir, **hashing é preferível** (a procura em um índice ordenado é da ordem de \log (qtde itens c em A)... com hashing é uma constante)

```
select  $A_1, A_2, \dots, A_n$   
from  $r$   
where  $A_i = c$ 
```

Comparação entre indexação ordenada e hashing

- As técnicas de indexação ordenada são preferíveis a hashing nos casos em que uma **faixa de valores é especificada na consulta**:

```
select  $A_1, A_2, \dots, A_n$   
from  $r$   
where  $A_i \leq c_2$  and  $A_i \geq c_1$ 
```

Lembre-se que, com hashing, (idealmente) os valores estão distribuídos aleatoriamente (não é trivial determinar a sequência)



6) Índices em chaves múltiplas

- Até aqui, fizemos a suposição que as chaves de procura eram atributos únicos
- Em muitos pedidos de recuperação/atualização vários atributos estão envolvidos
- Se uma certa combinação de atributos for usada com frequência, pode ser vantajoso definir uma estrutura de acesso que forneça acesso eficiente por meio de um valor chave que seja combinação daqueles atributos.



Exemplo:

- Liste os empregados do departamento de número 4 cuja idade é 59. Estratégias:
 - Se NRD tem índice
 - Se IDADE tem índice
 - Se ambos os atributos tiverem índices, a intersecção dos conjuntos de registros retornados pelos índices dá o resultado
- As 3 estratégias retornam resultados corretos. São eficientes? Se apenas algumas tuplas satisfazem a condição composta, não.



Técnicas para tratar chave de busca com múltiplos atributos

- Índices ordenados em atributos múltiplos
- Hashing particionado
- Arquivos grid