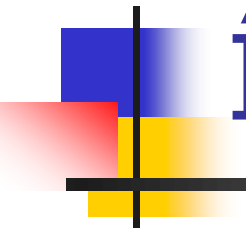


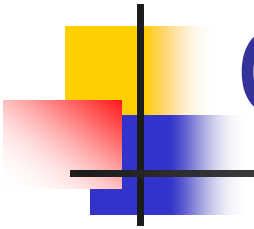
Índices

Estruturas de Dados II

Slides cedidos pelo Prof. Gustavo Batista (ICMC-USP)
[http://www.icmc.usp.br/~gbatista/]

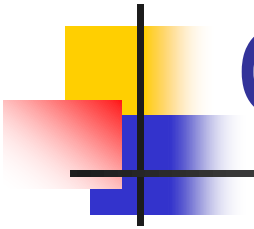
Busca e Ordenação

- Ordenação:
  - Visa facilitar o acesso aos elementos.
- Busca:
  - **Sequencial:** funciona para qualquer organização de arquivo. Lenta *O(n)*.
  - **Binária:** mais complexa para arquivos com registros de tamanho variável (requer estrutura adicional). Requer que o arquivo esteja ordenado. Mais eficiente *O(log n)*.
  - Para realizar uma busca binária é necessário ordenar o arquivo.



Ordenação

- Se o arquivo é pequeno, pode-se carregá-lo em memória principal, ordená-lo e gravá-lo.
  - Acessa-se o arquivo duas vezes (para leitura, antes de ordenar, e para escrita, depois de ordenado), mas esse acesso é o melhor possível: sequencial.
- Entretanto, deve-se encontrar um método de ordenação capaz de ordenar grandes arquivos.
- Um método de ordenação interna diretamente adaptado para ordenação externa poderia fazer um número de acesso ao disco muito grande.

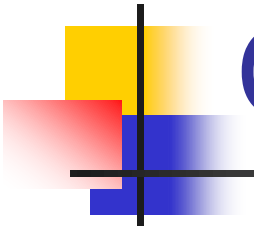


Ordenação

- Vimos que o único modo de recuperar ou encontrar um registro com alguma rapidez é procurar por ele utilizando seu **RRN** (para registros de tamanho fixo).
- Mas e se não conhecemos o RRN ou o byte offset do registro e a única informação que temos é o valor de um campo do registro (ex.: nome, RA, telefone, etc.)?

Ordenação por chave

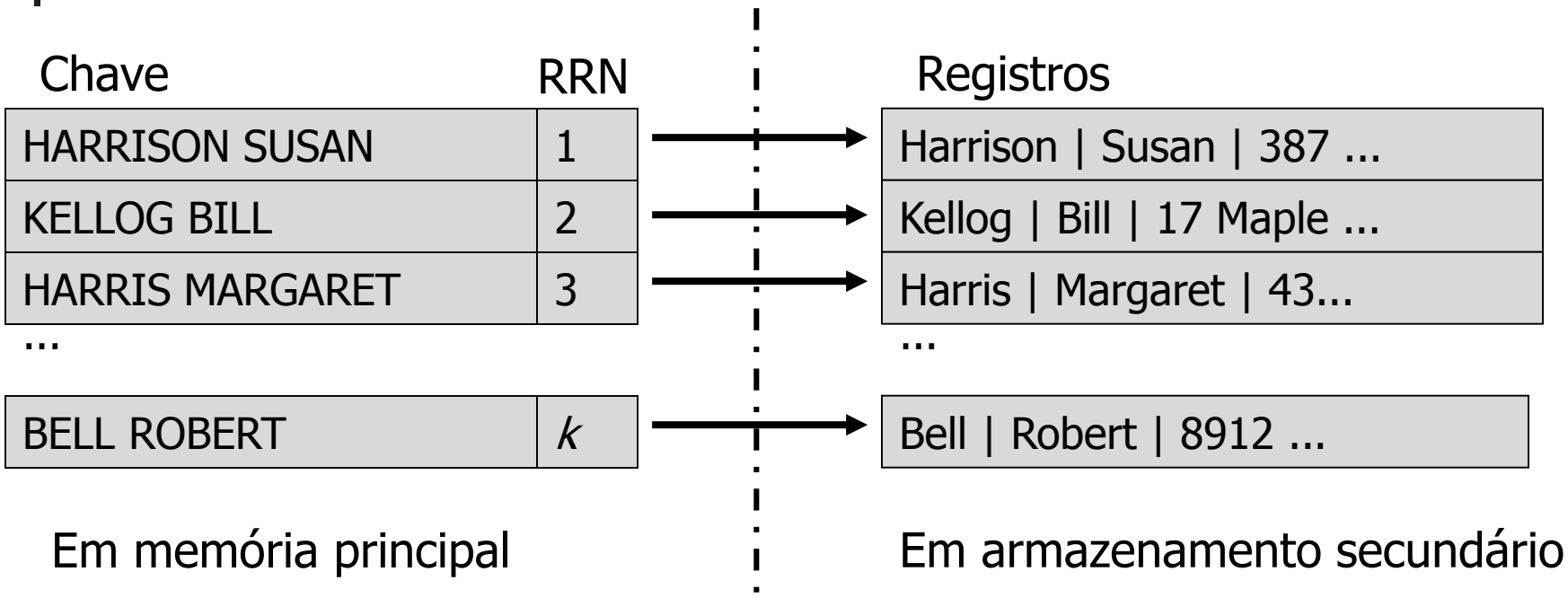
Ordenação: Keysorting



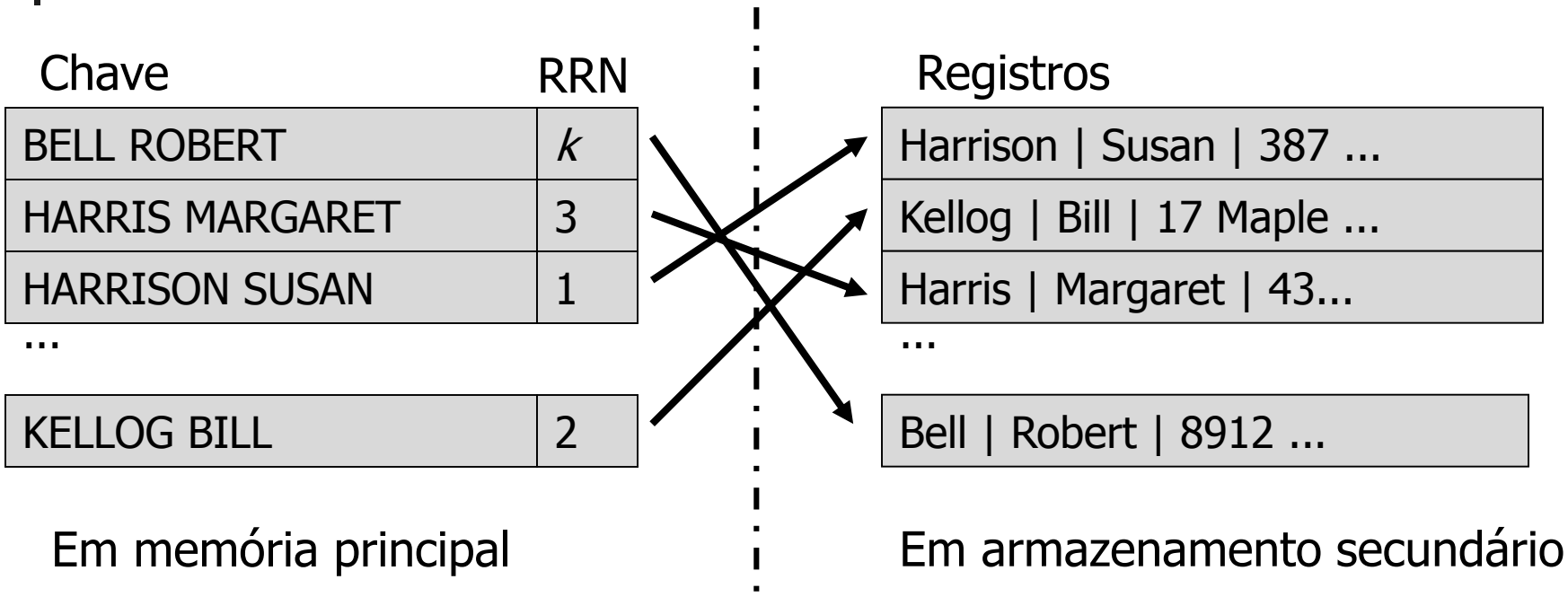
- *Keysorting* ou *tag sort* é baseado na ideia que o importante é ordenar as chaves e, portanto, o arquivo não precisa ser integralmente carregado em memória.
- Dessa forma, somente as chaves e os RRNs (supondo um arquivo de registros de tamanho fixo) são carregados em memória principal e ordenados com um método de ordenação interna.

Keysorting x Internal Sorting

Algoritmo Keysorting



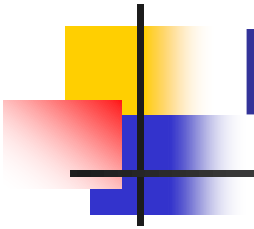
Algoritmo Keysorting



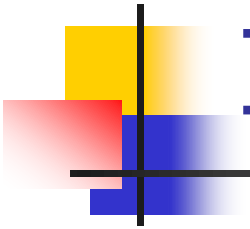
Limitações Keysorting

- Principais limitações:
  - Necessidade de ler todo o arquivo de entrada duas vezes;
  - A segunda leitura não é sequencial, mas aleatória: custo de posicionamento das cabeças é grande.
  - A gravação do arquivo de saída é aparentemente sequencial. Mas os *seeks* fazem com que o custo de gravar o arquivo seja alto.

Melhorias Keysorting

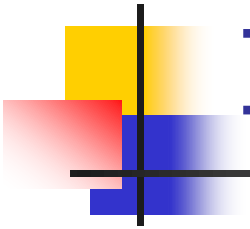


- Uma melhoria para a técnica do *keysorting* consiste em, ao invés de escrever o arquivo ordenado no disco, gerar um arquivo com as chaves ordenadas, que passa a atuar como um índice para o arquivo original.
- Deste modo, a pesquisa por um registro em particular pode ser feita por pesquisa binária, na memória RAM (o índice cabe!): essa pesquisa fornece a posição exata do registro desejado, o qual é obtido com um único acesso a disco.



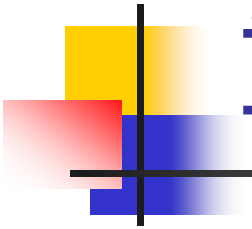
Índice

- O uso de índices (**indexação**) pode ser uma alternativa à ordenação quando precisamos organizar um arquivo para ser pesquisado por chaves.



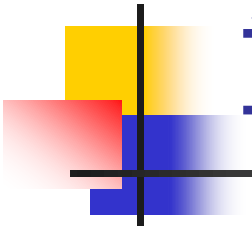
Índice

- Um índice sempre aparece nas páginas finais de um livro, na forma de uma tabela com uma lista de tópicos (chaves) e os números de páginas nas quais os tópicos podem ser encontrados (campos de referência). No livro, o índice fornece uma maneira de localizar um tópico rapidamente.
- Todos os índices são baseados nos mesmos conceitos básicos: chaves e campos de referência.



Índice

- Índices são estruturas utilizadas para facilitar a localização de informações.
- No caso de arquivos, as principais vantagens são:
  - Permite impor uma ordem no arquivo sem rearranjar o arquivo;
  - Possibilita múltiplas visões sobre o arquivo por meio da criação de mais de um índice;
  - Fornece acesso rápido a arquivos com registros de tamanho variável.



Índice simples

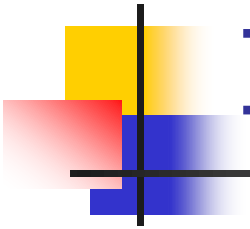
- Ex.: uma enorme coleção de CDs.
- Registros de tamanho variável com os campos:
  - **ID Number:** Número de identificação;
  - **Title**: Título;
  - **Composer**: Compositor(es);
  - **Artist**: Artista(s);
  - **Label**: Rótulo (código da gravadora).
- Chave primária: combinação de **Label** e **ID Number.**
  - **Label ID**. Sua forma canônica consiste do rótulo em letras maiúsculas, seguido pela representação do número de identificação em ASCII, como por exemplo: LON2312.

Índice simples

Rec. addr.	Label	ID number	Title	Composer(s)	Artist(s)
32†	LON	2312	Romeo and Juliet	Prokofiev	Maazel
77	RCA	2626	Quartet in C Sharp Minor	Beethoven	Julliard
132	WAR	23699	Touchstone	Corea	Corea
167	ANG	3795	Symphony No. 9	Beethoven	Giulini
211	COL	38358	Nebraska	Springsteen	Springsteen
256	DG	18807	Symphony No. 9	Beethoven	Karajan
300	MER	75016	Coq d'or Suite	Rimsky-Korsakov	Leinsdorf
353	COL	31809	Symphony No. 9	Dvorak	Bernstein
396	DG	139201	Violin Concerto	Beethoven	Ferras
442	FF	245	Good News	Sweet Honey in the Rock	Sweet Honey in the Rock

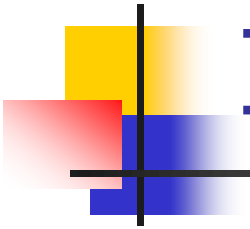
†Assume there is a header record that uses the first 32 bytes.

FIGURE 6.2 Sample contents of *Datafile*.



Índice simples

- Como organizar esse arquivo para garantir um acesso rápido a um registro qualquer, dada a sua chave primária?
- Poderíamos pensar em ordenar o arquivo e fazer uma busca binária pelo registro, mas existe um problema (qual?).
- Uma alternativa seria construir um índice para o arquivo.



Índice simples

- O índice é ele próprio um arquivo com registros de tamanho fixo.
- Cada registro tem 2 campos de tamanho fixo:
  - um campo contém a chave;
  - o outro informa a posição inicial (RRN/*byte offset*) do registro no arquivo de dados.
- Cada registro do arquivo de dados possui um registro correspondente no arquivo índice.
- O índice está ordenado, apesar do arquivo de dados não estar:
  - Em geral, o arquivo de dados está organizado segundo a ordem de entrada dos registros - ***entry sequenced file***.

Índice simples

Indexfile

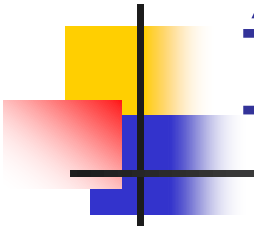
Key	*Reference field*	*Address of record*
ANG3795	167	32
COL31809	353	77
COL38358	211	132
DG139201	396	167
DG18807	256	211
FF245	442	256
LON2312	32	300
MER75016	300	353
RCA2626	77	396
WAR23699	132	442

Datafile

Actual data record

LON ¦ 2312 ¦ Romeo and Juliet ¦ Prokofiev . . .
RCA ¦ 2626 ¦ Quartet in C Sharp Minor . . .
WAR ¦ 23699 ¦ Touchstone ¦ Corea . . .
ANG ¦ 3795 ¦ Symphony No. 9 ¦ Beethoven . . .
COL ¦ 38358 ¦ Nebraska ¦ Springsteen . . .
DG ¦ 18807 ¦ Symphony No. 9 ¦ Beethoven . . .
MER ¦ 75016 ¦ Coq d'or Suite ¦ Rimsky . . .
COL ¦ 31809 ¦ Symphony No. 9 ¦ Dvorak . . .
DG ¦ 139201 ¦ Violin Concerto ¦ Beethoven . . .
FF ¦ 245 ¦ Good News ¦ Sweet Honey In The . . .

FIGURE 6.3 Sample index with corresponding data file.

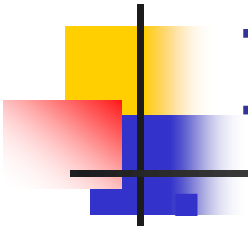


Índice simples

- Usar o índice para localizar um registro, dado o seu **LabelID**, é muito simples. O índice permite, inclusive, a aplicação (indireta) de pesquisa binária no arquivo de dados, cujos registros têm tamanho variável!

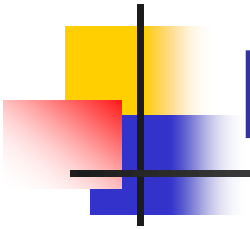
procedure retrieve-record(KEY)
  encontre posição de KEY no arquivo índice (pode usar P.B.!)
  pegue o valor do *byte offset* correspondente ao registro (+ o seu tamanho, se necessário)
  posicione sobre o registro usando SEEK e o *byte offset*
  leia o registro do arquivo de dados
end procedure

Índice simples



- Usa-se dois arquivos:
  - o arquivo índice (*index file*);
  - e o arquivo de dados (*data file*).
- O arquivo índice é:
  - mais fácil de trabalhar, pois usa registros de tamanho fixo;
  - pode ser pesquisado com busca binária (em memória principal);
  - é muito menor do que o arquivo de dados.
- Registros de tamanho fixo no arquivo índice impõem um limite ao tamanho da chave primária.
- Os registros do índice poderiam conter outros campos além da chave/*offset* (por exemplo, o tamanho do registro).

Índice simples em memória principal



- A inclusão de registros será muito mais rápida se o índice pode ser mantido (manipulado) em memória e o arquivo de dados é *entry sequenced*.
- Dados a chave e o *offset*, um único *seek* (i.e., um único acesso a disco) é necessário no arquivo de dados para recuperar o registro correspondente.

Operações básicas no índice

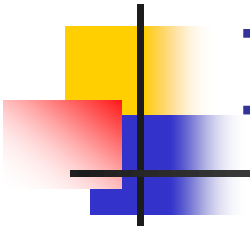
- Para índices que cabem em memória:
  - **Criação dos arquivos índice e de dados**;
    - criados como dois arquivos (em disco) inicialmente vazios, apenas com os registros *header*.
  - **Carregar índice para memória**;
    - carrega registros do *arquivo índice* no *vetor de registros INDEX[]*.
  - **Adição de registro**:
    - A inserção deve ser feita no arquivo de dados;
    - E também no índice, que provavelmente será reorganizado.
  - **Eliminação de registro**:
    - Remove do arquivo de dados, usando algum dos mecanismos de remoção;
    - Remove também do índice. A remoção do registro do índice pode exigir a sua reorganização, ou pode-se simplesmente marcar os registros como removidos.

Operações básicas no índice

- Para índices que cabem em memória:
  - **Atualização de registro**: a atualização cai em duas categorias:
    - Muda o valor da chave;
      - pode exigir uma reorganização do arquivo índice e do arquivo de dados. Conceitualmente, pode ser tratada como uma remoção seguida de inserção de novo registro.
    - Muda o conteúdo do registro.
      - se a chave não foi alterada, pode-se precisar alterar o arquivo de dados, sem necessidade de mexer no índice, a não ser que as posições dos registros mudem.....
  - **Atualizar índice no disco**: caso sua cópia em memória tenha sido alterada:
    - É imperativo que o programa se proteja contra índices desatualizados.

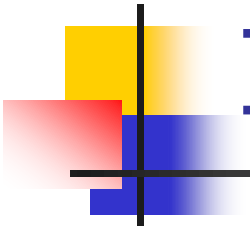
Como evitar índices desatualizados

- Deve haver um mecanismo que permita saber se o índice está atualizado em relação ao arquivo de dados.
- Possibilidade: um *status flag* é setado no arquivo índice mantido em disco assim que a sua cópia na memória é alterada.
- Esse *flag* pode ser mantido no registro *header* do arquivo índice, e atualizado sempre que o índice é reescrito no disco.
- Se um programa detecta que o índice está desatualizado, uma função é ativada e reconstrói o índice a partir do arquivo de dados.



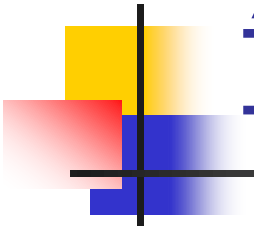
Índices muito grandes

- Se o índice não cabe inteiro na memória, o seu acesso e manutenção precisa ser feito em memória secundária.
- Não é mais aconselhável usar índices simples, uma vez que:
  - a busca binária pode exigir vários acessos a disco;
  - a necessidade de deslocar registros nas inserções e remoções de registros tornaria a manutenção do índice excessivamente cara.



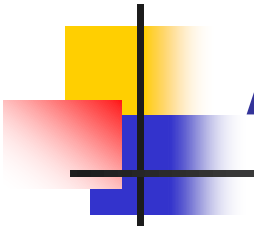
Índices muito grandes

- Utiliza-se outras organizações para o índice:
  - *Hashing,* caso a velocidade de acesso seja a prioridade máxima;
  - Árvores-B, caso se deseje combinar acesso por chaves e acesso sequencial eficientemente.



Índices muito grandes

- Mesmo que o índice não caiba na memória, índices simples são úteis em comparação a um único arquivo ordenado por chave, pois:
  - Permite a utilização de P.B. para obter acesso a chaves de registros de tamanho variável;
  - Se os registros dos índices são substancialmente menores do que os do arquivo de dados, ordenar e manter índices pode ser menos caro do que ordenar e manter o arquivo de dados;
  - Se existem *pinned records* no arquivo de dados, o índice nos permite reorganizar as chaves sem mover os registros de dados.



Pinned Records

- Registros cuja posição no arquivo é importante por serem referenciados em outro lugar (por exemplo, em uma lista Dispo ou em um índice)
- Pelo fato de que a posição do registro é importante, os registros não podem ser mudados de posição (ordenados!)
- Nesses casos, se os registros forem mudados os ponteiros associados ficarão inválidos (*dangling pointers*)

Acesso por múltiplas chaves

- Como saber qual é a chave primária do registro que se quer acessar?
- Normalmente, o acesso a registros não se faz por chave primária, e sim por chaves secundárias.
- Solução: cria-se um índice que relaciona uma chave secundária à chave primária (e não diretamente ao registro).

Acesso por múltiplas chaves

- Índices permitem muito mais do que simplesmente melhorar o tempo de busca por um registro.
- Múltiplos índices secundários:
  - permitem manter diferentes visões dos registros em um arquivo de dados;
  - permitem combinar chaves associadas e, deste modo, fazer buscas que combinam visões particulares.

Acesso por múltiplas chaves

Composer index

Secondary key	*Primary key*
BEETHOVEN	ANG3795
BEETHOVEN	DG139201
BEETHOVEN	DG18807
BEETHOVEN	RCA2626
COREA	WAR23699
DVORAK	COL31809
PROKOFIEV	LON2312
RIMSKY-KORSAKOV	MER75016
SPRINGSTEEN	COL38358
SWEET HONEY IN THE R	FF245

FIGURE 6.6 Secondary key index organized by composer.

Title index

Secondary key	*Primary key*
COQ D'OR SUITE	MER75016
GOOD NEWS	FF245
NEBRASKA	COL38358
QUARTET IN C SHARP M	RCA2626
ROMEO AND JULIET	LON2312
SYMPHONY NO. 9	ANG3795
SYMPHONY NO. 9	COL31809
SYMPHONY NO. 9	DG18807
TOUCHSTONE	WAR23699
VIOLIN CONCERTO	DG139201

FIGURE 6.8 Secondary key index organized by recording title.

Alterações nas operações básicas

- **Adição de registro:**
  - Quando um novo registro é inserido no arquivo, devem ser inseridas as entradas correspondentes nos índices primário e secundários;
  - O campo **chave** no índice secundário deve ser armazenado na forma canônica, e o valor pode ser truncado porque o tamanho da chave deve ser mantido fixo;
  - Uma diferença importante entre os índices primário e secundários é que, nos últimos, pode ocorrer duplicação de chaves. Chaves duplicadas devem ser mantidas agrupadas e ordenadas segundo a chave primária (Fig. 6.6).

Alterações nas operações básicas

- **Eliminação de registro:**
  - Implica na remoção do registro do arquivo de dados e de todos os índices;
  - Índices primário e secundários são mantidos ordenados segundo a chave e, consequentemente, a remoção envolve o rearranjo dos registros remanescentes para não deixar “espaços vagos”.

Alterações nas operações básicas

- **Eliminação de registro:**
  - **Alternativa:** Atualizar apenas o índice primário - não eliminar a entrada correspondente ao registro do índice secundário (por isso o secundário aponta para o primário!).
    - **Vantagem:** economia de tempo substancial quando vários índices secundários estão associados ao arquivo, principalmente quando esses índices são mantidos em disco.
    - **Custo:** o espaço ocupado por registros inválidos. Poder-se-ia fazer "coletas de lixo" periódicas nos índices secundários.
      - Mas ainda será um problema se o arquivo é muito volátil:
        - outra solução: índice em árvore-B.

Alterações nas operações básicas

- **Atualização de registro** - Existem 3 situações possíveis:
  - a atualização alterou uma chave secundária: o índice secundário para esta chave precisa ser reordenado;
  - alterou a chave primária: reordenar o índice primário e corrigir os índices secundários (os campos de referência). A vantagem é que a atualização dos índices secundários requer apenas uma reorganização local (no caso de registros repetidos requerem reorganização);
  - alteração em outros campos: não afeta os índices primário e secundários.

Busca usando múltiplas chaves

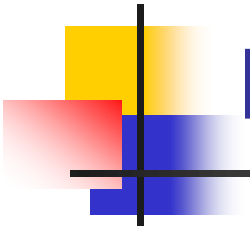
- Uma das aplicações mais importantes das chaves secundárias envolve o uso de uma ou mais chaves para localizar conjuntos de registros do arquivo de dados, fazendo uma busca em vários índices e uma combinação (AND, OR, NOT) dos resultados.
- Ex.: encontre todos os registros de dados com:
  - composer = “BEETHOVEN” AND title = “SYMPHONY NO. 9”.

Busca usando múltiplas chaves

Find all data records with:
composer = "Beethoven" AND title = "Symphony No. 9"

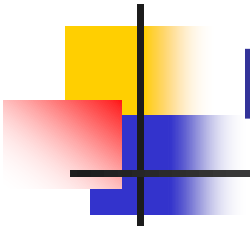
Compositor	Título	Matched list
ANG3795	ANG3795	ANG3795
DG139201	COL31809	DG18807
DG18807	DG18807	
RCA2626		

ANG	3795	Symphony No. 9	Beethoven	Giulini
DG	18807	Symphony No. 9	Beethoven	Karajan



Melhoria de índices secundários

- Dois problemas com as estruturas de índices vistas até agora são:
  - a repetição das chaves secundárias
    - Resulta em arquivos maiores e, portanto, com menores chances de caber na memória;
  - a necessidade de reordenação dos índices cada vez que um novo registro é inserido no arquivo, mesmo que esse registro tenha um valor de chave secundária já existente no arquivo



Melhoria de índices secundários

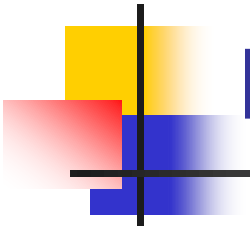
- **Solução 1:** Associar um vetor de tamanho fixo a cada chave secundária
  - Não é necessário reordenar o índice a cada inserção de registro (somente dentro do próprio vetor)
  - Limitado a um número fixo de repetições.
  - Ocorre fragmentação interna no índice - que talvez não compense a eliminação da duplicação de chaves

Melhoria de índices secundários

Revised composer index

Secondary key	*Set of primary key references*			
BEETHOVEN	ANG3795	DG139201	DG18807	RCA2626
COREA	WAR23699			
DVORAK	COL31809			
PROKOFIEV	LON2312			
RIMSKY-KORSAKOV	MER75016			
SPRINGSTEEN	COL38358			
SWEET HONEY IN THE R	FF245			

FIGURE 6.9 Secondary key index containing space for multiple references for each secondary key.



Melhoria de índices secundários

- **Solução 2:** Manter uma lista de referências - Listas invertidas
  - Já que se tem uma lista de chaves primárias, pode-se associar cada chave secundária a uma lista encadeada (lista invertida) das chaves primárias referenciadas
  - Redefine-se o índice secundário de forma que ele seja composto por registros com 2 campos: um campo chave, e um campo com o RRN do primeiro registro na lista invertida
  - As referências às chaves primárias associadas a cada chave secundária são mantidas em um arquivo sequencial separado, organizado segundo a entrada dos registros

Listas invertidas: visão conceitual

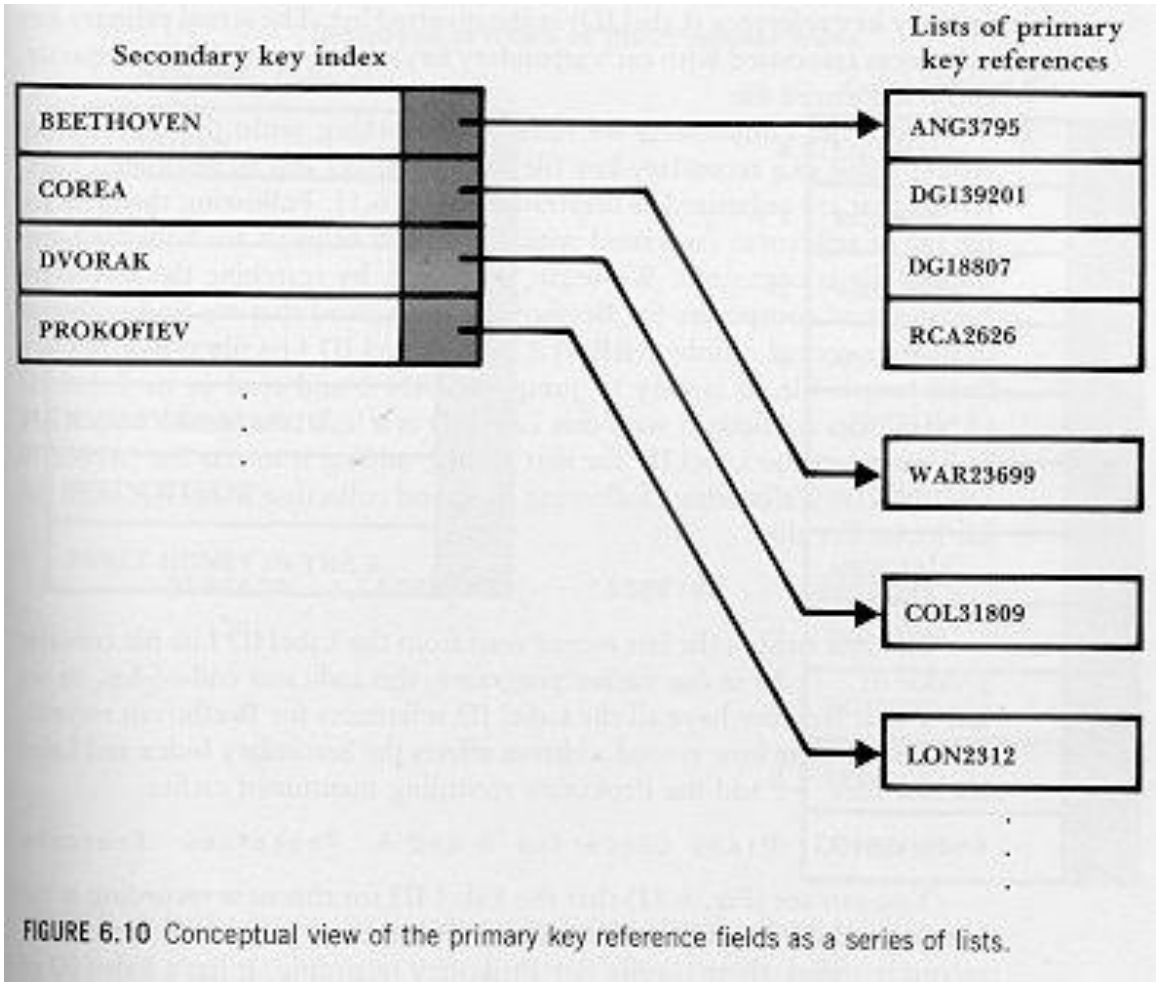
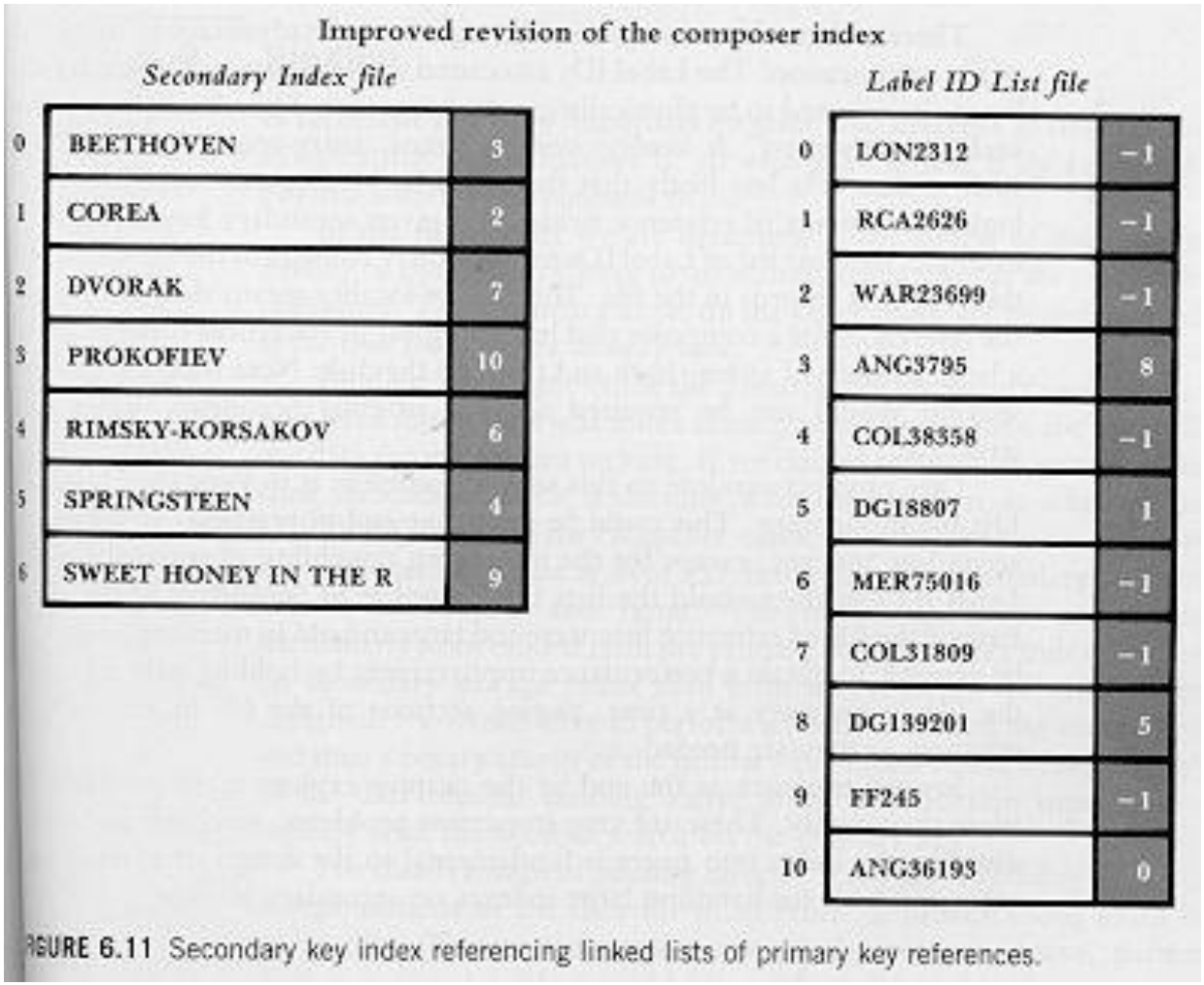


FIGURE 6.10 Conceptual view of the primary key reference fields as a series of lists.

Listas invertidas



Improved revision of the composer index

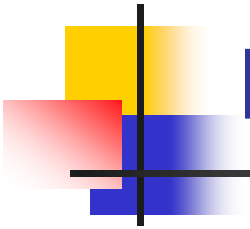
Secondary Index file

0	BEETHOVEN	3
1	COREA	2
2	DVORAK	7
3	PROKOFIEV	10
4	RIMSKY-KORSAKOV	6
5	SPRINGSTEEN	4
6	SWEET HONEY IN THE R	9

Label ID List file

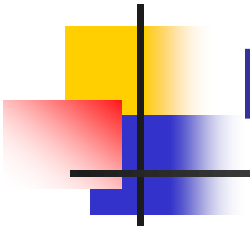
0	LON2312	−1
1	RCA2626	−1
2	WAR23699	−1
3	ANG3795	8
4	COL38358	−1
5	DG18807	1
6	MER75016	−1
7	COL31809	−1
8	DG139201	5
9	FF245	−1
10	ANG36193	0

IGURE 6.11 Secondary key index referencing linked lists of primary key references.



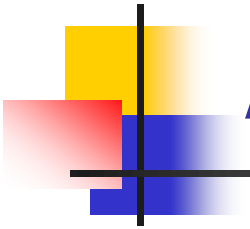
Lista invertida

- **Vantagens:**
  - o índice secundário só é alterado (reorganizado) quando um registro com uma chave inexistente é inserido, ou quando uma chave existente é alterada
  - operações de eliminação, inserção ou alteração de registros já existentes implicam apenas na mudança do arquivo da lista invertida
  - a ordenação do arquivo de índice secundário é mais rápida: menos registros - e registros menores
    - Mesmo se for mantida em memória secundária
  - o arquivo com listas de chaves nunca precisa ser ordenado, pois é "*entry-sequenced*"



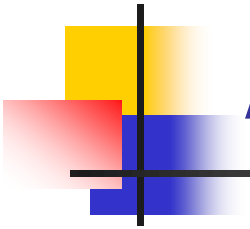
Lista invertida

- **Problema:**
  - Registros associados não estão adjacentes - podem ser necessários vários *seeks* para recuperar a lista invertida
  - O ideal seria poder manter o índice e a lista invertida na memória



Binding

- Nos índices primários vistos a associação (*early binding*) entre a chave primária e o endereço físico do registro a que ela se refere ocorre no momento em que o registro é criado
- Um índice simples fornece acesso direto e, portanto, mais rápido, a um registro, dada a sua chave
- Já as chaves secundárias são associadas a um endereço apenas no momento em que são de fato usadas (*late binding*). Isso implica um acesso mais lento, já que elas se referem ao índice primário, que pode estar em disco.



Binding

- O *late binding* trouxe vantagens: manutenção mais flexível, mais eficiente e confiável

- Ressalta-se que: é sempre desejável manter as modificações localizadas, o que é possível com o *late-binding*. O *early binding* só é aconselhável se o arquivo de dados é estático, ou quase, e o acesso rápido a registros é a maior prioridade