Indexação
Múltiplas chaves, lista invertida e *binding*

6897/9895 – Organização e Recuperação de Dados
Profa. Valéria D. Feltrim

UEM – CTC – DIN

Slides preparados com base no Cap. 6 do livro FOLK, M.J. & ZOELLICK, B. *File Structures*. 2nd Edition, Addison-Wesley Publishing Company, 1992, e nos slides disponibilizados pelo Prof. Pedro de Azevedo Berger (DCC/UnB)

Acesso por múltiplas chaves

Chave	*Referência*
ANG3795	167
LON2312	77
RCA2626	32
WAR23699	132

Índice Primário

```
 32 43RCA|2626|A Rush of Blood to the Head|Coldplay...
 77 53LON|2312|É Proibido Fumar|Skank|...
132 33WAR|23699|Creep|Radiohead...
167 42ANG|3795|Shiver|Coldplay...
```

Arquivo de Dados

- Buscas via chave primária são raras
  - "Encontre o registro com a chave WAR23699"
- Rotineiramente, se faz busca por chave secundária
  - "Encontre o registro da música "Creep"
- Podemos criar novos índices para o catálogo de músicas, por ex., um índice por título, um por compositor e um por artista
  - Esses índices seriam secundários!
  - Lembre-se que chaves secundárias admitem duplicação

Índices secundários

- Um índice secundário é um arquivo distinto e, em geral, uma chave secundária se relaciona à uma chave primária (e não diretamente ao *offset* do registro de dados)
- Podemos criar tantos índices secundários quantas sejam as chaves de acesso
- Cada índice dá uma visão diferente do arquivo de dados

Índices secundários

- **Exemplo de índice secundário**: um arquivo que relaciona a chave Artista com a chave RÓTULO+ID
  - **RÓTULO+ID** é a chave primária e **Artista** é uma chave secundária (duplicidade de chaves permitida)

Chave secundária	*Chave primária*
CHARLIE BROWN JR	FF245
COLDPLAY	ANG3795
COLDPLAY	DG139201
COLDPLAY	DG18807
COLDPLAY	RCA2626
RADIOHEAD	WAR23699
ROBERTO CARLOS	COL31809
SKANK	COL38358
SKANK	LON2312
SKANK	MER75016

- ✓ Note que a **referência** é para a **chave primária** e não para o *byte-offset* do registro no arquivo de dados. Isso significa que o índice primário deve ser consultado para se encontrar o *byte-offset* do registro
- ✓ Existem vantagens em não vincular a chave secundária a um endereço físico
- ✓ **Perceba que o índice é ordenado pela chave secundária**
- ✓ **Perceba que as chaves duplicadas são ordenadas pela chave primária**

Busca em índice secundário

```
procedure busca-no-secundario(CHAVE)
   Pesquise por CHAVE no índice secundário
   Assim que CHAVE for encontrada, faça CHAVE_PRIMARIA receber
      a chave primária que está no campo de referência de CHAVE
   Chame busca(CHAVE_PRIMARIA) para obter o registro de dados
end procedure
```

Busca por combinação de chaves

- O uso de múltiplos índices secundários facilita as buscas por combinação de chaves
  - Exemplo
    - Encontre todas as gravações com artista "SKANK" e com título "É Proibido Fumar"

Artista="Skank"	Título="É proibído..."	Resultado (AND)
COL38358	COL31809	**LON2312**
LON2312	**LON2312**	
MER75016		

- Operações de interseção e união de listas são facilitadas se as listas estiverem ordenadas → os índices são sempre ordenados!
- Com o uso de índices secundários, esse tipo de busca é simples e rápida, pois as operações lógicas (AND e OR) serão realizadas nos índices, que estarão em memória

Inserir registros (Múltiplas chaves)

- Na inserção de um novo registro de dados, as respectivas entradas devem ser inseridas no índice primário e nos índices secundários
- Inserção em índice secundário:
  - O custo é similar ao de inserir um registro no índice primário
    - Decresce muito se os índices estiverem em memória
  - A chave secundária é armazenada na forma canônica
    - Uso de registro de tamanho fixo se deseja-se usar busca binária
  - Há uma duplicação natural de chaves secundárias
    - As chaves duplicadas são agrupadas e devem estar ordenadas pelo campo de referência (i.e., chave primária)
    - A ordenação por chave primária facilita a busca por combinação de chaves (interseção e união)

Remover registros (Múltiplas chaves)

- Implica na remoção do registro do arquivo de dados e de todos os índices
- Duas abordagens: (1) ***Excluir-todas-referências*** e (2) ***Excluir-algumas-referências***
- Abordagem 1: *Excluir-todas-referências*
  - Remove todas as referências no índice primário e nos índices secundários
    - Implica em reorganizar todos os índices → muito custoso se os índices não cabem todos na memória
  - **Caso os índices secundários façam referência direta ao *byte-offset* do registro de dados, essa abordagem é obrigatória!**

Remover registros (Múltiplas chaves)

- Abordagem 2: *Excluir-algumas-referências*
  - Faz uso da "indireção" índice secundário → índice primário
    - Remove-se o registro do arquivo de dados e a sua referência no índice primário, sem modificar os índices secundários
  - A busca por chave secundária vai continuar funcionando corretamente, pois quando for feita a busca no índice primário, o retorno será "chave inexistente"
    - Como o índice secundário aponta para o índice primário, a condição pode ser testada sem causar inconsistência
    - **Mesmo que existam "n" índices secundários, apenas uma remoção em índice será necessária (a do índice primário)**
  - Vantagem → economia de tempo (evitando as remoções) quando vários índices secundários estão associados ao índice primário
  - Desvantagem → desperdício de espaço que é ocupado por registros inválidos, além da queda do desempenho da busca com o passar do tempo por causa dos "falsos candidatos"
    - Uma alternativa é fazer "coletas de lixo" periódicas nos índices secundários, minimizando essa desvantagem

Atualizar registros (Múltiplas chaves)

- Se a chave secundária for alterada
  - Provavelmente o respectivo índice secundário precisará ser reordenado
- Se a chave primária for alterada
  - O índice primário deverá ser reordenado e os campos de referência (chave primária) de todos os índices secundários precisam ser atualizados
  - Pode ser necessária uma "reordenação local" nos índices secundários, uma vez que chaves duplicadas devem estar ordenadas pela chave primária
- Quando outro campo (não-chave) é alterado
  - Se o registro for de tamanho variável, pode ser necessário tratar como uma remoção seguida de inserção no arquivo → mudança de *byte-offset*
    - Atualiza o *byte-offset* no índice primário
    - Os índices secundários não precisam ser reordenados!

Melhorando a estrutura de índices secundários

- A estrutura de índice secundário vista até agora apresenta dois problemas:
  1. 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 índice secundário
     - Movimentação de registros nos vetores de índices para abrir espaço para a nova chave
  2. A repetição das chaves secundárias resulta em estruturas maiores do que o necessário
     - Chaves secundárias idênticas, mas com campos de referência diferentes
     - Além de desperdiçar espaço em disco, faz com que os arquivos tenham menos chances de caber na memória principal

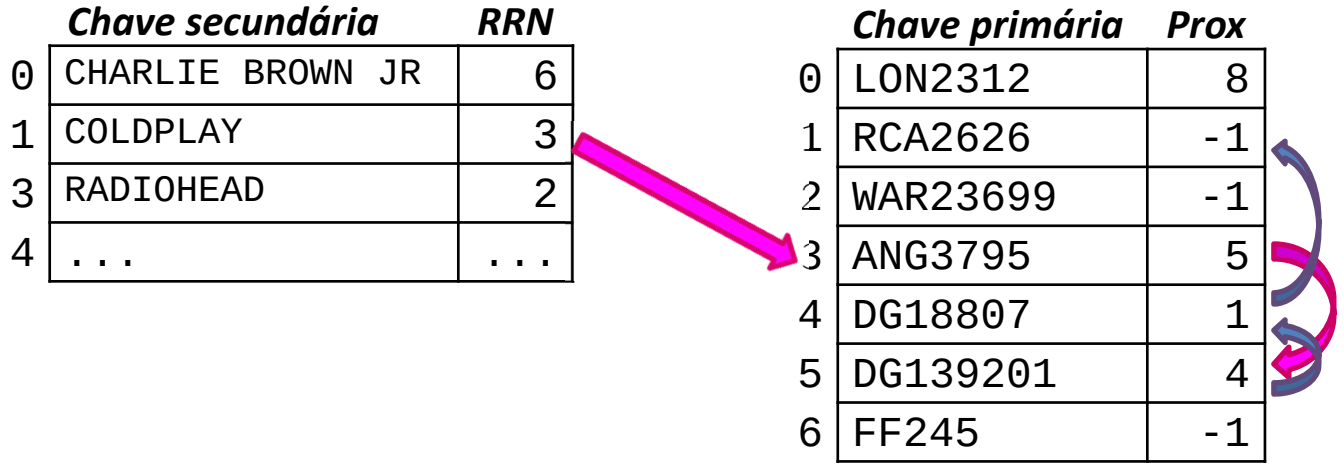
Como melhorar a estrutura de índices secundários de forma que eles ocupem menos espaço em disco e exijam menos ordenação?

Listas invertidas

- Índices em que uma chave leva a uma lista encadeada de referências são chamados de listas invertidas
  - Em vez de buscarmos nos registros pelas ocorrências de uma chave, temos a chave ligada a uma lista de registros nos quais ela ocorre → daí o nome de lista invertida
  - Já que temos um conjunto de chaves primárias associado a cada chave secundária, podemos ligar cada chave secundária a uma lista de chaves primárias
- 2ª solução
  - Organizar o arquivo de índice secundário de forma que cada registro contenha a chave secundária e um ponteiro para uma lista encadeada de chaves primárias

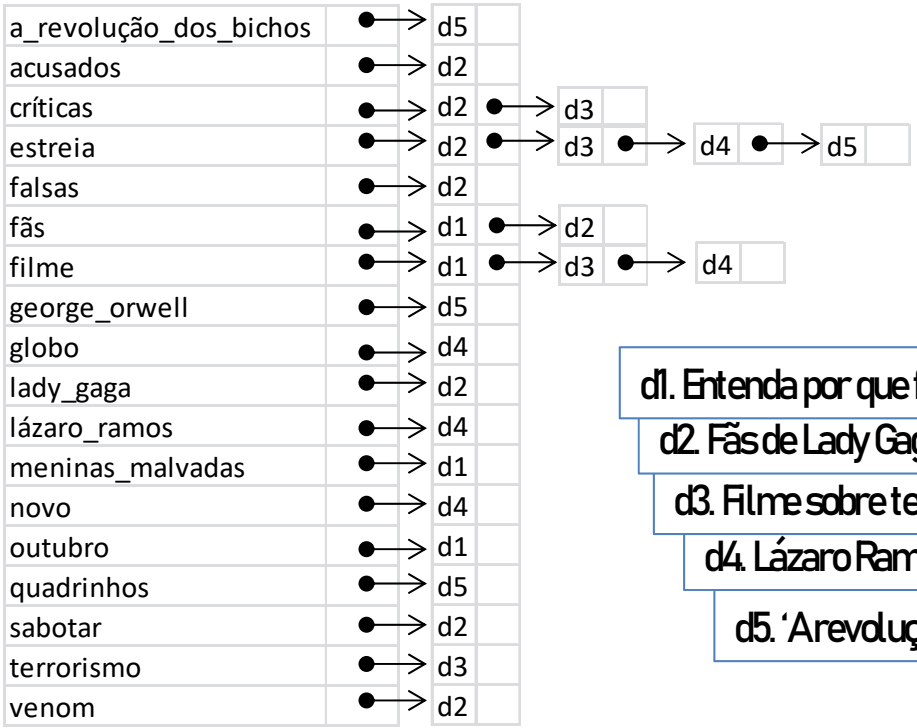
Listas invertidas

- **Índice secundário com lista invertida:** uso de dois arquivos
  1. Índice → Registros com dois campos de tamanho fixo
     - Chave, que armazena a chave secundária (no exemplo, Artista)
     - RRN, que armazena o RRN do registro da 1ª chave primária na lista invertida
  2. Lista invertida → Registros com dois campos de tamanho fixo
     - Chave, que armazena a chave primária (no exemplo, ROTULO+ID)
     - PROX, que armazena o RRN da próxima chave primária na lista invertida
       - A lista invertida é mantida em um arquivo sequencial separado, **organizado segundo a ordem de inserção dos registros**



Listas invertidas

- Listas invertidas podem ser usadas para implementar buscas por palavras-chaves em conjuntos de documentos (textos)
  - Em vez de buscarmos em todos os documentos pela ocorrência de uma palavra-chave, temos cada palavra-chave ligada a uma lista de documentos nos quais ela ocorre



d1. Entenda por que fãs do filme 'Meninas malvadas' comemoram o 3 de outubro.

d2. Fãs de Lady Gaga são acusados de sabotar estreia de 'Venom' com falsas críticas.

d3. Filme sobre terrorismo recebe críticas antes mesmo da estreia.

d4. Lázaro Ramos estreia novo filme dia 11 na Globo.

d5. 'A revolução dos bichos', de George Orwell, estreia em quadrinhos.

Listas invertidas

- Vantagens
  - O índice secundário só é reordenado quando uma nova chave é inserida ou quando uma chave existente é alterada
  - Remoção ou inserção de registros com chaves existentes implicam apenas em mudanças na lista invertida
    - Ocasionalmente, pode ocorrer alteração do campo RRN do índice secundário
    - Para remover todas as ocorrências de uma chave secundária basta atribuir -1 ao campo de referência no índice secundário
  - O índice secundário será menor e mais fácil de ser manipulado
  - A lista invertida não é ordenada, o que facilita a manutenção
    - Podemos facilmente reutilizar o espaço dos registros removidos do arquivo contendo a lista invertida, pois eles têm tamanho fixo (PED)
    - Podemos inserir no final do arquivo, uma vez que a lista não é ordenada

Listas invertidas

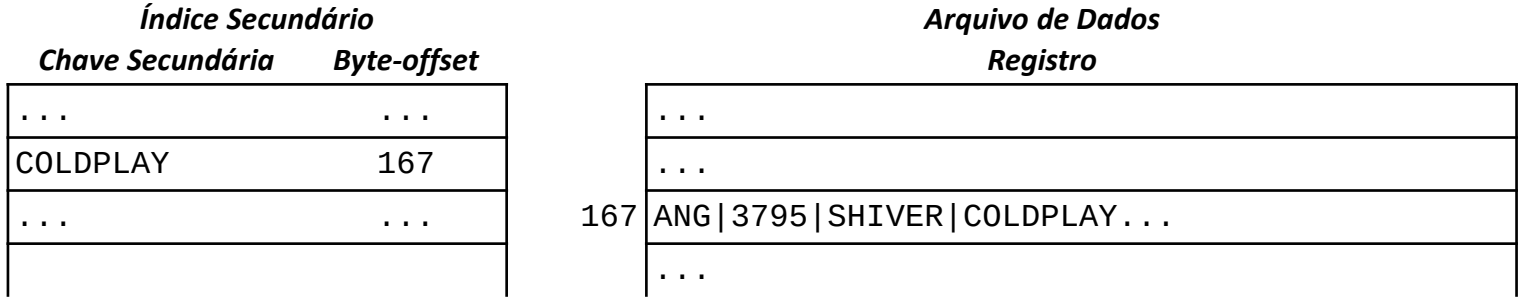
- Desvantagem
  - Perda da localidade:
    - As chaves primárias associadas com uma certa chave secundária geralmente não se encontram agrupadas fisicamente na lista invertida
      - Recuperar toda a lista de chaves primárias associadas a uma chave secundária pode envolver muitos acessos, se lista invertida estiver em disco
    - Se a lista invertida puder ser mantida em memória junto com os índices, o problema do número de acessos deixa de existir
      - Pode-se manter uma única lista invertida para vários índices secundários → adiciona-se um campo PROX para cada índice
      - Aumenta as chances da lista caber na RAM

Binding

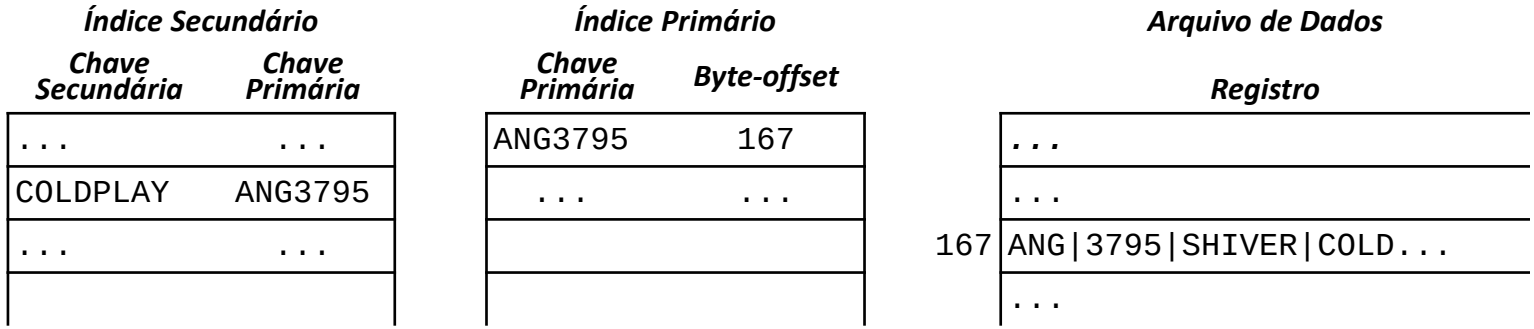
- Ligação ou "*binding*" é o nome dado a associação existente entre uma chave e o endereço físico do registro ao qual ela se refere
  - A associação se dá pelo campo de referência
- Em que momento essa ligação deve ser efetuada?
  - **No índice primário, a ligação sempre se dá no momento da criação do índice**
  - **Nos índices secundários esse momento pode variar**
    - A ligação pode ser feita no momento da criação do índice
      - O índice secundário "aponta" para o arquivo de dados
    - A ligação pode ser feita no momento em que a chave for de fato utilizada, ou seja, na momento de uma busca
      - O índice secundário "aponta" para o índice primário

Binding

- Temos duas opções
  - ***Early-binding*** (ligação precoce) → associação da chave ao endereço físico do registro de dados



  - ***Late-binding*** (ligação tardia) → associação de uma chave à outra chave (em geral, primária) do registro de dados



Ligação tardia (*Late binding*)

- ***Late-binding*** → A associação da chave secundária a um endereço físico ocorre no momento do seu uso (busca)
- **Propriedades**
  - Acesso indireto: chave secundária → chave primária → *byte-offset*
  - Queda no desempenho da busca (mais lenta)
  - Menor custo nas alterações com a abordagem *Excluir-algumas-referências*
  - Mais seguro: modificações referentes à *byte-offset* no arquivo de dados afetam apenas o índice primário
    - Maior prevenção contra erros de codificação (esquecimento de atualizar um dos *n* índices secundários) e falhas do sistema (por ex., queda de energia no momento das atualizações)
  - Menos problemas de inconsistência: se o índice primário estiver consistente, os índices secundários também estarão

Ligação tardia (*Late binding*)

- Principal motivação para o uso do *late binding*
  - "É desejável que modificações importantes sejam feitas em um único arquivo do que em vários arquivos"

Ligação precoce (*Early binding*)

- ***Early-binding*** → A associação da chave secundária a um endereço físico ocorre no momento do criação dos índices
- **Propriedades**
  - Acesso mais rápido a partir dos índices secundários → o impacto será maior quando os índices estiverem em disco
  - **Alto custo nas alterações**
    - Qualquer modificação relativa a *byte-offset* no registro de dados demanda ajustes em todos os índices (primário e secundários)
    - Uso obrigatório da abordagem *Excluir-todas-referências*
  - Consistência garantida somente após o ajuste de todos os índices
  - Menos seguro: modificações referentes à *byte-offset* no arquivo de dados afetam todos os índices
    - Quanto mais índices temos para atualizar, maiores as chances de ocorrer algum problema (erro de codificação, falha do sistema, etc.) na atualização

Ligação precoce (*Early binding*)

- Quando usar *early binding*?
  - “Quando o arquivo é estático ou sofre poucas alterações e o desempenho da busca tem prioridade”

Exercício

- Dado o *arquivo de dados* ao lado, monte **dois índices secundários**: um por *Categoria* e outro por *Departamento*, **usando o mesmo arquivo de *lista invertida*** para os dois índices.
- Considere a *Matrícula* como chave primária e monte também o **índice primário**.

RRN	Matrícula	Categoria	Depto.
0	2000	Analista	Engenharia
1	1040	Técnico	Computação
2	980	Docente	Arquitetura
3	1900	Secretário	Computação
4	2050	Docente	Computação
5	1010	Analista	Computação
6	1550	Docente	Engenharia
7	430	Secretário	Engenharia
8	2200	Técnico	Computação
9	1990	Secretário	Arquitetura
10	1790	Analista	Arquitetura
11	1100	Analista	Engenharia
12	730	Secretário	Computação
13	1620	Técnico	Computação
14	790	Docente	Engenharia
15	690	Docente	Arquitetura