Indexação
Índices lineares → operações básicas

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)

O que é um índice?

- Um índice é uma alternativa para se encontrar a informação buscada sem fazer busca diretamente no arquivo de dados
- Todos os índices são baseados no mesmo conceito básico → chaves + referências
- Por enquanto, trataremos de índice lineares
  - Vetores contendo estruturas do tipo {chave, referência}
- Propriedades dos índices:
  - Trabalham por indireção → permitem que uma ordem seja imposta ao arquivo de dados sem rearranjá-lo fisicamente
    - Facilita a inserção no arquivo de dados
    - Elimina o problema dos registros "fixos" (*pinned records*) na ordenação
  - Proporcionam múltiplas "visões" ordenadas de um mesmo arquivo
    - Índices diferentes referenciando um mesmo arquivo de dados (p.e., um catálogo de uma biblioteca com índices por autor, título, tema, etc.)
  - Permitem acesso direto por chave a registros de tamanho fixo e variável

Exemplo de arquivo de dados

- **Exemplo**: Catálogo de músicas
  - Registros de tamanho variável precedidos por um campo de tamanho (2 bytes)
    - Gravadora, ID, Título, Artista, Compositor, Ano
  - **Chave primária** = Gravadora + ID
    - Ex.: **LON2312**

```
 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...
    ...
```

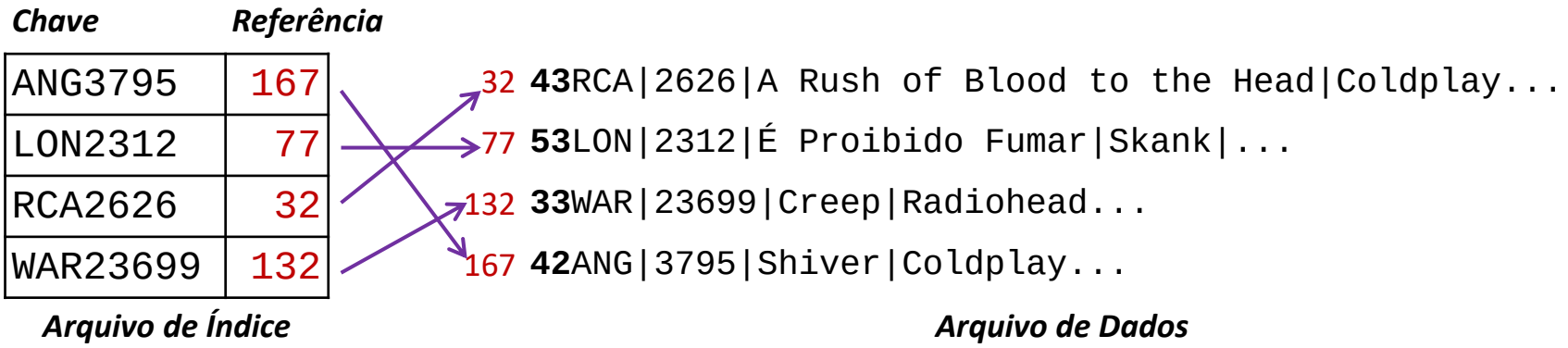
Byte-offset do registro

Como organizar este arquivo para garantir acesso rápido a um registro qualquer, dada a sua chave primária?

- Estamos assumindo que o cabeçalho utiliza os 32 primeiros bytes do arquivo e que o campo de tamanho dos registros ocupa 2 bytes

Índices lineares

- Índice linear primário → arquivo com registros de tamanho fixo, em que cada registro tem dois campos de tamanho fixo:
  - Chave primária = gravadora+ID
  - Referência = *byte-offset* do registro de dados correspondente (RRN se os registros tem tamanho fixo)
  - Cada registro do arquivo de dados corresponde a um único registro no índice
    - **Relação 1 para 1**
  - **O índice deve ser ordenado pela chave**
    - O arquivo de dados permanece organizado segundo a ordem de inserção



Índices lineares

- Usar o índice para buscar um registro pela chave é simples
  - O índice permite a aplicação (indireta) de busca binária no arquivo de dados, mesmo que ele tenha registros de tamanho variável

```
procedure busca(CHAVE)
   encontre posição de CHAVE no arquivo de índice (usando B.B.)
   pegue o valor do byte-offset correspondente ao registro
   posicione o ponteiro de L/E do arquivo de dados sobre o
       registro usando SEEK e o byte-offset
   leia o registro do arquivo de dados
end procedure
```

Índices lineares

- Considerações
  - O arquivo de índice é mais fácil de trabalhar
    - Usa registros de tamanho fixo, além de ser muito menor que o arquivo de dados (viabiliza a busca binária)
  - A inserção de registros será rápida se o índice for mantido na RAM
    - Índices que não podem ser mantidos na RAM devem ser organizados de outra forma
  - O uso de registros de tamanho fixo no arquivo de índice impõe um limite ao tamanho da chave primária
    - Isso pode acarretar problemas práticos → o truncamento da chave pode fazer com que ela deixe de ser única
  - Os registros do índice poderiam conter outros campos além do par {chave, *byte-offset*}
    - Por ex., o tamanho do registro

Índices lineares

- Vantagens
  - O arquivo de dados é *sequencial*
    - Os registros são inseridos sempre no final do arquivo, de acordo com ordem de entrada (ou em espaço disponível para reutilização, se houver uma LED)
    - Facilita a inserção e a manutenção
  - Qualquer chave pode ser localizada rapidamente no índice usando busca binária (supondo que o mesmo está em memória)
  - Uma vez localizada a chave no índice, tem-se acesso ao *byte-offset* do registro correspondente e um único acesso é necessário no arquivo de dados
    - Mais rápido do que fazer busca binária em um arquivo ordenado armazenado em disco

Operações básicas

- Por enquanto, vamos assumir que o índice pode ser carregado inteiro do disco para a memória RAM na forma de um vetor de registros que chamaremos de INDEX[]
- Operações para manter um arquivo de dados junto com seu índice:
  1. Criar arquivos de índice e de dados
  2. Carregar o índice para a memória
  3. Regravar o índice depois de usá-lo
  4. Inserir registros no arquivo de dados e no índice
  5. Remover registros do arquivo de dados e do índice
  6. Atualizar registros
     - Com mudança de chave primária
     - Sem mudança de chave primária
  7. Buscar registro pelo valor de chave (**procedure** *busca*(CHAVE))

Operações básicas

1. Criar arquivos de índice e de dados
   - Os arquivos de dados e de índice são criados como arquivos vazios (contendo apenas o cabeçalho, se for o caso)
   - Esses arquivos serão posteriormente carregados com seus dados
2. Carregar o arquivo de índice para a memória
   - Leia os cabeçalhos dos arquivos de índice e de dados
   - Verifique se os cabeçalhos do índice e do arquivo de dados são compatíveis
     - Se o índice não for válido, gere um novo índice
   - Leia os registros do arquivo de índice para o vetor INDEX[]
     - INDEX[] deve ser definido de modo que cada elemento do vetor tenha a mesma estrutura dos registros no arquivo de índice
     - A leitura do arquivo de índice é sequencial e deve ser feita de forma eficiente, por ex., lendo-se blocos de registros

Operações básicas

3. Regravar o arquivo de índice depois de usá-lo
   - A regravação só é necessária se houver alteração no vetor INDEX[]

```
procedure regravar-indice( )
   verifique a flag que sinaliza se o vetor INDEX[] foi alterado
   se a flag está setada (houve modificações)
      abra o arquivo de índice como um novo arquivo (vazio)
      atualize o registro de cabeçalho e escreva-o no novo arquivo
      escreva o conteúdo de INDEX[] no novo arquivo de índice
   feche o arquivo de índice
end procedure
```

Operações básicas

4. Inserir registros nos arquivos de dados e de índice
   - Na inserção de registros no arquivo de dados, o vetor INDEX[] também deve ser atualizado
     - Deve ser criada uma nova entrada no índice
   - A chave primária na sua forma canônica e o respectivo *byte-offset* devem ser inseridos na ordem correta no vetor INDEX[]
     - Pode envolver reorganização do vetor (o que pode ser feito sem grandes custos, pois o índice está na memória RAM)
   - Se espaço disponível no arquivo de dados foi reaproveitado, a LED deve ser atualizada

Operações básicas

5. Remover registros dos arquivos de dados e de índice
   - Ao remover (logicamente) um registro do arquivo de dados, a LED deve ser atualizada para posterior reutilização do espaço
   - A entrada correspondente em INDEX[] deve ser removida
     - INDEX[] é reorganizado, mantendo-o ordenado pela chave
       - É possível simplesmente marcar a entrada do índice como removida (remoção lógica) e aguardar uma rotina de manutenção de todo o sistema

Operações básicas

6. Atualizar registros
   - Com mudança de chave primária
     - Envolve mudanças no arquivo de dados e reordenação do vetor INDEX[]
   - Sem mudança de chave primária
     - Não envolve reordenação do vetor INDEX[], mas pode envolver reorganização do arquivo de dados
       - Se o registro alterado couber no espaço do registro atual, então nenhuma mudança é feita em INDEX[]
       - Caso o registro alterado não caiba no espaço atual, deve-se colocar o espaço atual na LED e inserir o registro alterado em uma nova posição no arquivo de dados
         - » Nesse caso, o *byte-offset* da chave correspondente no INDEX[] deve ser atualizado

Índices muito grandes na memória

- E se o índice é muito grande para ser mantido em memória?

- Uso de algoritmos especializados que requerem um número menor de *seeks*
  - *Árvores-B*
    - Uso de estruturas não lineares para organizar os índices
    - Podem combinar acesso por chave e acesso sequencial de forma eficiente
  - *Hash*
    - Uso de funções de “espalhamento”
    - Para os casos em que a velocidade de acesso é a maior prioridade