Organização arquivos: organização lógica

Prof. Tiago A. Almeida
talmeida@ufscar.br

Agenda

- ✓ Organização em **campos**
- ✓ Organização em **registros**
- ✓ Métodos de acesso

Organização de arquivos

- ✓ **Organização lógica:** modo como os dados (registros e campos) estão internamente armazenados
- ✓ A organização lógica do arquivo é **definida pelo SO ou pelo programador** (aplicativo) no momento da criação do arquivo
- ✓ A escolha da estrutura de dados pode variar em função do tipo de informação contida no arquivo

✓ Pontos de decisões

- Organização em **campos**
- Organização em **registros**
- Métodos de acesso

Organização de arquivos

- ✓ Escolher uma organização lógica de arquivos é uma **decisão de projeto**

- ✓ **Critérios importantes que devem ser analisados**
  - **Velocidade de acesso** a um registro ou a um conjunto de registros relacionados
  - **Facilidade** em inserir, atualizar e remover registros
  - Eficiência (velocidade) de **armazenamento**
  - **Redundância** como garantia contra a corrupção de dados

Organização de arquivos

- ✓ Dependendo de como a informação é mantida no arquivo, **campos lógicos sequer podem ser recuperados**

- ✓ **Exemplo**
  - Suponha que desejamos armazenar em um arquivo os nomes e endereços de várias pessoas
  - Suponha que decidimos representar os dados como uma sequência de bytes (sem delimitadores, contadores etc.)

AmesJohn123MapleStillOK74075MasonAlan90EastgateAdaOK74820

- ✓ **Não há como recuperar** porções individuais (nome ou endereço)
  - **Perde-se a integridade** das unidades fundamentais de organização dos dados

Organização em Campos

Métodos de organização em campos

- ✓ **Campo: menor unidade lógica de informação** em um arquivo

- ✓ Existem diversas maneiras de organizar um arquivo mantendo a identidade dos campos
  - A organização do exemplo anterior não proporciona isso

- ✓ **Métodos para organização em campos**
  - Comprimento fixo
  - Indicador de comprimento
  - Delimitadores
  - Uso de *tags* (etiquetas)

Campos com tamanho fixo

- ✓ Cada campo ocupa no arquivo um **tamanho fixo**, pré-determinado

- ✓ O fato do tamanho ser conhecido garante que **é possível recuperar cada campo**

Luciana	Rua 1	123	Sorocaba
Alexandre	Rua 2	321	São Carlos
Pedro	Rua 3	213	Itu

Campos com tamanho fixo

✓ Desvantagem dessa abordagem

- O espaço alocado (e não usado) aumenta o tamanho do arquivo desnecessariamente (**desperdício**)
- Solução **inapropriada** quando se tem uma grande quantidade de **dados com tamanho variável**
- Razoável apenas se o comprimento dos campos é realmente fixo ou apresenta pouca variação

Campos com indicador de tamanho

- O **tamanho de cada campo** é armazenado imediatamente antes do dado
  - Se o tamanho do campo é inferior a 256 bytes, o espaço necessário para armazenar a informação de comprimento é um único byte

07Luciana05Rua 10312308Sorocaba

09Alexandre05Rua 20332110São Carlos

05Pedro05Rua 30321303Itu

Campos separados por delimitadores

- ✓ **Caractere(s) especial(ais)** (que não fazem parte do dado) são escolhido(s) para ser(em) inserido(s) ao final de cada campo
  - Ex.: para o campo *nome* pode-se utilizar `/`, `tab`, `#`, etc...
  - Espaços em branco não servem na maioria dos casos

Luciana|Rua 1|123|Sorocaba|

Alexandre|Rua 2|321|São Carlos|

Pedro|Rua 3|213|Itu|

Tag do tipo “keyword=value”

- **Vantagem:** cada campo fornece informação semântica sobre si próprio
  - Fica mais fácil identificar o conteúdo do arquivo

Nome=Luciana|Endereco=Rua 1|Numero=123|Cidade=Sorocaba|

Nome=Alexandre|Endereco=Rua 2|Numero321|Cidade=São Carlos|

Nome=Pedro|Endereco=Rua 3|Numero=213|Cidade=Itu|

- **Desvantagem:** as *keywords* podem ocupar uma porção significativa do arquivo

Organização em Registros

Organização em registros

- ✓ **Registro lógico:** conjunto de campos agrupados
  - Assim como o conceito de campo, um registro lógico é uma ferramenta conceitual, que não necessariamente existe no sentido físico
- ✓ **Registro físico:** o armazenamento de um arquivo é feito, via de regra, por blocos de registros lógicos
- ✓ Um bloco corresponde a quantidade de dados transferidos em um acesso simples
- ✓ Um bloco de registros lógicos corresponde a um registro físico;
- ✓ Em cada operação de leitura ou gravação é lido ou gravado um bloco e não apenas um registro lógico
- ✓ A organização física do arquivo é definida pelo SO

Organização em registros

✓ Métodos para organização em registros

- Tamanho fixo
- Número fixo de campos
- Indicador de tamanho
- Uso de índice
- Utilizar delimitadores

Registros de tamanho fixo

- Analogamente ao conceito de **campos de tamanho fixo**, assume-se que todos os registros têm o mesmo tamanho, com campos de tamanho fixo ou não
- Um dos métodos mais comuns de organização de arquivos

Registro de tamanho fixo e campos de tamanho fixo:

Luciana	Rua 1	123	Sorocaba
Alexandre	Rua 2	321	São Carlos
Pedro	Rua 3	213	Itu

Registro de tamanho fixo e campos de tamanho variável:

Luciana|Rua 1|123|Sorocaba|	Espaço vazio |
Alexandre|Rua 2|321|São Carlos|	Espaço vazio |
Pedro|Rua 3|213|Itu|	Espaço vazio |

Registros de tamanho fixo

- ✓ Exemplo de registro com campos de tamanho fixo

```
struct {
  char Nome[50];
  char Rua[50];
  char Numero[6];
  char Cidade[50];
} Registro;
```

Registros com número fixo de campos

- ✓ Ao invés de especificar que cada registro contém um tamanho fixo, podemos especificar um **número fixo de campos**
  - O tamanho do registro é **variável**
  - Neste caso, os campos seriam separados por **delimitadores**

Registro com número fixo de campos:

Luciana|Rua 1|123|Sorocaba|Alexandre|Rua 2|321|São Carlos|Pedro|Rua 3|213|Itu|

Indicador de tamanho para registros

- ✓ O indicador que precede o registro fornece o seu **tamanho total**
  - Os campos são separados internamente por delimitadores
  - Boa solução para registros de tamanho variável

Registro com campos iniciados por indicador de tamanho:

23Luciana|Rua 1|123|Sorocaba|27Alexandre|Rua 2|321|São Carlos|16Pedro|Rua 3|213|Itu|

Índice de registros

- Um **índice externo** poderia indicar o deslocamento de cada registro relativo ao início do arquivo
  - Pode ser utilizado também para calcular o **tamanho dos registros**
  - Os campos seriam separados por **delimitadores**

Arquivo de dados + arquivo de índices

Dados: Luciana|Rua 1|123|Sorocaba|Alexandre|Rua 2|321|São Carlos|Pedro|Rua 3|213|Itu|

Índice: 00 26 57

Utilizar delimitadores

- Separar os registros com **delimitadores** análogos aos de fim de campo
  - O delimitador de campo é mantido, sendo que o método combina os dois delimitadores
  - Note que delimitar fim de campo é diferente de delimitar fim de registro

Registro delimitado por marcador (#)

Luciana|Rua 1|123|Sorocaba|#Alexandre|Rua 2|321|São Carlos|#Pedro|Rua 3|213|Itu|

Métodos de Acesso a Registros

Acesso a registros

- ✓ **Arquivos organizados por registros**
- ✓ Como encontrar um registro específico?
  - Cada registro poderia ter uma **identificação única**
    - ★ Aluno de número X
    - ★ Livro de código Y

Chaves

- ✓ Uma **chave (*key*)** está associada a um registro e permite a sua recuperação

- ✓ **Uma chave primária é, por definição, o campo utilizado para identificar unicamente um registro**
  - Exemplos: RA, CPF, Habilitação, Título de Eleitor
  - Sobrenome, por outro lado, não é uma boa escolha para chave primária

- ✓ Uma **chave secundária**, tipicamente, não identifica unicamente um registro, e pode ser utilizada para buscas simultâneas por vários registros
  - Todos os “**Silvas**" que moram em **São Paulo**, por exemplo

Chaves

- O ideal é que exista uma **relação um a um entre chave e registro**
- Se isso não acontecer, é necessário fornecer uma **maneira** do usuário decidir qual dos registros é o que interessa

Escolha da chave primária

- ✓ Preferencialmente, a **chave primária deve ser "*dataless*"**, isto é, não deve ter um significado associado, e **não deve mudar nunca** (outra razão para não ter significado)

- ✓ Uma mudança de significado pode implicar na mudança do valor da chave, o que invalidaria referências já existentes baseadas na chave antiga

Forma canônica da chave

- **Formas canônicas para as chaves:** uma única representação da chave conforme uma regra
  - “Ana”, “ANA”, ou “ana” devem levar ao mesmo registro

- Ex: a regra pode ser 'todos os caracteres maiúsculos'
  - Nesse caso a forma canônica da chave será “ANA”

Desempenho da busca

- Na pesquisa em RAM, normalmente adotamos como medida do trabalho necessário o **número de comparações** efetuadas para obter o resultado da pesquisa
- Na pesquisa em arquivos, o acesso a disco é a operação mais dispendiosa e, portanto, o **número de acessos a disco** efetuados é adotado como medida do trabalho necessário para obter o resultado
- **Mecanismo de avaliação do custo associado ao método:** contagem do número de chamadas à função de leitura de arquivo
- Assumimos (por enquanto) que
  - Cada `READ` lê 1 registro e requer um `SEEK`
  - Todas as chamadas a `READ` tem o mesmo custo

Busca sequencial

- Busca pelo registro que tem uma determinada chave em um arquivo

- **Lê o arquivo registro a registro**, em busca de um registro contendo um certo valor de chave

Busca sequencial

- Uma busca por um registro em um arquivo com **2.000 registros**:
  - Requer, em média, 1.000 leituras
  - 1 leitura se for o primeiro registro, 2.000 se for o último e, portanto, 1.000 em média
- No pior caso, o trabalho necessário para buscar um registro em um arquivo de tamanho *n* utilizando busca sequencial é *O(n)*
- Pode-se melhorar o desempenho da busca, reduzindo o número de acesso ao disco. **Como?**
  - **Blocagem de registros** (reduz o número de *SEEKS*)

Blocagem de registros

- A operação ***SEEK* é lenta**
- A transferência dos dados do disco para a RAM é relativamente rápida...
  - apesar de muito mais lenta que uma transferência de dados em RAM
- O custo de buscar e ler um registro, e depois buscar e ler outro, é maior que o custo de buscar (e depois ler) dois registros sucessivos de uma só vez
- Pode-se melhorar o desempenho da busca sequencial **lendo um bloco de registros por vez**, e então processar este bloco em RAM

Exemplo de blocagem

- ✓ Um arquivo com **4.000 registros**, com registros de **512 bytes**
- ✓ A busca sequencial por um registro, sem blocagem, requer em **média 2.000 leituras**
- ✓ Trabalhando com **blocos de 16 registros**, o número médio de leituras necessárias cai para **125** (dado que há 250 blocos)
- ✓ Cada **`READ`** gasta um pouco mais de tempo, mas o ganho é considerável devido à redução do número de **`READs`** (ou seja, de **`SEEKs`**)

Blocagem de registros

- ✓ **Melhora o desempenho**, mas o custo continua diretamente proporcional ao tamanho do arquivo, i.e., é ***O(n)***

- ✓ Reflete a **diferença entre o custo de acesso à RAM e o custo de acesso a disco**
  - Aumenta a quantidade de dados transferidos entre o disco e RAM

- ✓ Não altera o número de comparações em RAM

- ✓ Economiza tempo porque **reduz o número de operações `SEEK`**

Busca sequencial

✓ Vantagens

- Fácil de programar
- Requer estruturas de arquivos simples

✓ Quando usar?

- Na busca por uma cadeia em um arquivo ASCII
- Em arquivos com poucos registros
- Em arquivos pouco pesquisados
- Na busca por registros com um certo valor de chave secundária, para a qual se espera muitos registros (muitas ocorrências)

Acesso direto

- A alternativa mais radical ao acesso sequencial é o **acesso direto**

- O acesso direto implica em realizar um **`SEEK` direto para o início do registro** desejado (ou do setor que o contém) e ler o registro imediatamente

- É ***O(1)***, pois um único acesso traz o registro, independentemente do tamanho do arquivo

Posição do início do registro

- ✓ **Como localizar o início do registro no arquivo?**
- ✓ Para localizar a posição exata do início do registro no arquivo, pode-se utilizar um arquivo de **índice** separado
- ✓ Ou pode ter um **RRN (relative record number) (ou byte offset)** que fornece a posição relativa do registro dentro do arquivo

Posição de um registro com RRN

- Para utilizar o **RRN**, é necessário trabalhar com registros de **tamanho fixo**

- Nesse caso, a posição de início do registro é calculada facilmente a partir do seu RRN
  - ***Byte offset* = RRN * Tamanho do registro**

- Por exemplo, se queremos a posição do registro com RRN 546, e o tamanho de cada registro é 128 bytes
  - **O *byte offset* é 546 x 128 = 69.888**

Organização vs acesso

- **Organização de arquivos**
  - registros de tamanho fixo
  - registros de tamanho variável
- **Acesso a arquivos**
  - acesso sequencial
  - acesso direto

Organização vs acesso

- **Considerações a respeito da organização do arquivo**
  - arquivo pode ser dividido em campos?
  - os campos são agrupados em registros?
  - registros têm tamanho fixo ou variável?
  - como separar os registros?
  - como identificar o espaço utilizado e o “lixo”?

- **Existem muitas respostas para estas questões**
  - a escolha de uma organização em particular depende, entre outras coisas, do que se vai fazer com o arquivo

Organização vs acesso

- ✓ Arquivos que devem conter registros com tamanhos muito diferentes, devem utilizar registros de tamanho variável

- ✓ Existem também limitações da linguagem
  - **C** permite acesso a qualquer *byte*, e o programador pode implementar acesso direto a registros de tamanho variável
  - **Pascal** exige que o arquivo tenha todos os elementos do mesmo tipo e tamanho

Registro cabeçalho (*header record*)

- ✓ Em geral, é interessante manter **algumas informações sobre o arquivo** para uso futuro
- ✓ Essas informações podem ser mantidas em um **cabeçalho no início do arquivo**
- ✓ A existência de um registro cabeçalho torna um arquivo um **objeto auto-descrito**

Registro cabeçalho (*header record*)

- **Algumas informações típicas**
  - Número de registros
  - Tamanho de cada registro
  - Nomes dos campos de cada registro
  - Tamanho dos campos
  - Datas de criação e atualização

- Pode-se colocar informações elaboradas

- **Desvantagem dessa abordagem?**
  - O software deve ser mais flexível e, portanto, mais sofisticado