Sistemas de Banco de Dados

Fundamentos em Bancos de Dados Relacionais

Wladmir Cardoso Brandão
www.wladmirbrandao.com

ORGANIZAÇÃO DE DADOS

Organização de Dados

A forma como dados são dispostos em memória secundária impacta o desempenho do SGBD para recuperação e manipulação desses dados

- Tipicamente dados são organizados como **ARQUIVOS DE REGISTROS**

REGISTRO → coleção de valores relacionados a fatos sobre o minimundo, tais como atributos, instâncias de entidades e relacionamentos

12345678900	Roberto Machado	M	1200.00	1

ARQUIVO → coleção de registros relacionados

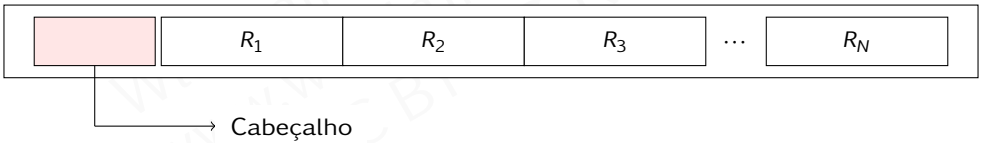
12345678900	Roberto Machado	M	1200.00	1	⋯	32145678900	Ana Maria Freitas	F	7500.00	2

Registros devem ser organizados de forma a serem rapidamente localizados

Organização de Dados

Um **ARQUIVO** possui um **cabeçalho (descritor)** contendo metadados úteis aos programas que acessam seus registros

- Ordem, tipo e tamanho de campos dos registros
- Endereços dos blocos que armazenam registros do arquivo
- Códigos de caracteres especiais, como separadores de campos



Arquivos de Registros

Cada valor de um **registro** está restrito a um **tipo de dado**, sendo que o número de bytes para cada tipo é fixo, dependendo do sistema computacional

- Booleano: 1 byte
- Inteiro: 4 bytes
- Número real: 4 bytes
- Inteiro longo: 8 bytes
- Data: 10 bytes (formato DD-MM-AAAA)
- String: n bytes, onde n é o número de caracteres
- BLOB: $p + n$ bytes, onde p é o tamanho do ponteiro no registro para o endereço de bloco onde o objeto binário de tamanho n está armazenado

Arquivos de Registros

Arquivos podem ser compostos por registros de **tamanho**:

- Fixo → cada registro no arquivo tem o mesmo tamanho
- Variável → registros no arquivo possuem tamanhos diferentes
  - Campos de tamanho variável → VARCHAR, TEXT
  - Campos opcionais → NULL
  - Campos multivalorados
  - Arquivos mistos com registros de instâncias de entidades diferentes

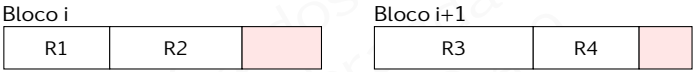
Tipicamente, todos os registros em um arquivo referem-se às instâncias de uma mesma entidade

- Arquivo Professor → Entidade Professor

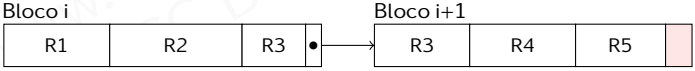
Arquivos de Registros

Um **ARQUIVO** é alocado em diferentes blocos de disco, sendo que seus **REGISTROS** podem estar alocados em um ou vários blocos

- NÃO ESPALHADO → registro não pode atravessar o limite de um bloco



- ESPALHADO → registro é armazenado em múltiplos blocos
  - Ponteiro no fim de cada bloco aponta para o bloco de continuidade do registro



Blocagem, ou **Fator de Bloco**, ou **Fator de Blocagem** de um arquivo é a quantidade de registros desse arquivo que cabem em um bloco de disco

Considere:

- Blocos com t bytes
- Registros de r bytes, sendo $r \leq t$
- Fator de Bloco $\rightarrow F = \left\lfloor \frac{t}{r} \right\rfloor$

Em arquivo com registros de tamanho fixo, r é o tamanho do registro, enquanto em arquivo com registros de tamanho variável, considera-se r o tamanho médio de registros

Se r é suficientemente grande, tal que $r > t/2$, o espaço não usado em disco pode ser grande e o espalhamento de registros passa a ser vantajoso para reduzir esse *"espaço perdido"*

- Espaço não usado $\rightarrow U = t - (F \times r)$

O **número de blocos** (B) necessários para armazenar um arquivo é:

- $B = \left\lceil \frac{n}{F} \right\rceil$, onde n é o número de registros do arquivo

Por exemplo, considere um arquivo de PROFESSOR armazenado em um disco com blocos de $t = 4KB$, onde:

- $r = 185B$
- $n = 10.000$

Nesse caso, teremos:

- $F = \left\lfloor \frac{4KB}{185B} \right\rfloor = \left\lfloor \frac{4 \times 1.024B}{185B} \right\rfloor \approx \lfloor 22,14 \rfloor = 22$
- $U = 4KB - (22 \times 185B) = 4.096B - 4.070B = 26B$
- $B = \left\lceil \frac{10.000}{22} \right\rceil \approx \lceil 454,54 \rceil = 455$
- Consumo de espaço $\rightarrow 455 \times 4KB = 1.820KB \approx 1,77MB$

Métodos de Acesso

Grupo de operações que podem ser aplicadas a um arquivo

- Recuperação → localização de registros em arquivo para que valores de campos possam ser lidos e processados, sem que haja alteração nos dados
- Atualização → alteração de arquivo pela inserção ou exclusão de registros, ou pela modificação de valores de campos de registros

A frequência da mudança em arquivos determina a frequência de execução de operações de atualização

- Arquivos Estáticos → operações de atualização são raramente executadas
- Arquivos Dinâmicos → mudam com frequência de forma que operações de atualização são constantemente executadas

Métodos de Acesso

Muitas operações aplicadas a arquivos envolvem **PESQUISA**

- Especifica critérios que registros devem satisfazer
- Tipicamente, critérios envolvem expressões booleanas
- Expressões podem apresentar diferentes graus de complexidade
  - SIMPLES → expressões booleanas simples
    - Exemplo: (Sexo = 'M')
  - COMPLEXAS → expressões booleanas complexas
    - Exemplo: ((Sexo = 'M') ∧ ((Salario > 3.000) ∨ (¬ TemDependente)))

Métodos de Acesso

SGBDs acessam registros utilizando **OPERAÇÕES REPRESENTATIVAS**, em que tipicamente um registro é processado por vez

- OPEN → prepara arquivo para leitura ou escrita
  - Aloca *buffers* para blocos
  - Recupera o cabeçalho do arquivo
  - Posiciona o ponteiro de arquivo no início do arquivo
- RESET → posiciona o ponteiro do arquivo aberto para o início do arquivo
- CLOSE → libera *buffers* alocados e realiza operações de limpeza de memória

Operações representativas

- Find (Locate) → procura o primeiro registro que satisfaça uma condição
  - Transfere o bloco que contém o registro para um *buffer* alocado
  - Posiciona o ponteiro de arquivo no registro, tornando-o o registro atual
- FindNext → procura o próximo registro que satisfaça uma condição
  - Transfere o bloco que contém o registro para um *buffer* alocado
  - Posiciona o ponteiro de arquivo no registro, tornando-o o registro atual
- Read (Get) → copia o registro do *buffer* para uma variável de programa
  - Posiciona o ponteiro no próximo registro, tornando-o o registro atual

Operações representativas

- Delete → remove o registro atual
  - Transfere o *buffer* de volta ao bloco no disco
- Modify → modifica valores de campos do registro atual
  - Transfere o *buffer* de volta ao bloco no disco
- Insert → insere um novo registro no arquivo
  - Localiza o bloco onde o registro deve ser inserido
  - Transfere o bloco para um *buffer*
  - Escreve o registro no *buffer*
  - Transfere o *buffer* de volta ao bloco no disco

Operações representativas

- Scan → combinação das operações Find, FindNext e Read
  - Se uma condição é especificada
    - Se o ponteiro de arquivo estiver posicionado no início do arquivo, reposiciona-o no primeiro registro que satisfaça a condição
    - Se o ponteiro de arquivo estiver posicionado em algum registro, reposiciona-o no próximo registro que satisfaça a condição
  - Caso contrário
    - Se o ponteiro de arquivo estiver posicionado no início do arquivo, reposiciona-o no primeiro registro
    - Se o ponteiro de arquivo estiver posicionado em algum registro, reposiciona-o no próximo registro

Métodos de Acesso

Existem **operações representativas** de nível mais alto que podem ser aplicadas a conjuntos de registros

- FindAll → procura o conjunto de registros que satisfaça uma condição
- FindOrdered → procura, em uma ordem específica, o conjunto de registros que satisfaça uma condição
- FindN → procura os *N* primeiros registros que satisfaçam uma condição
- Reorganize → reorganiza os blocos e registros de um arquivo

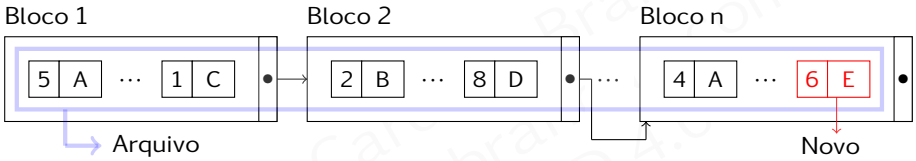
Tipos de Arquivo

Métodos de acesso operam de maneira diferente dependendo da forma como arquivos são organizados, especialmente de como os registros encontram-se dispostos dentro dos arquivos

- **Arquivo Heap (Pilha)** → registros posicionados sem ordem, com novos registros acrescentados ao final do arquivo
- **Arquivo Sequencial** → registros posicionados ordenadamente por um ou mais campos, denominados *campos de ordenação*
- **Arquivo Hash** → registros posicionados a partir da aplicação de uma *função hash* sobre um ou mais campos, denominados *campos hash*

Tipos de Arquivo: Heap

Arquivo organizado de forma que os registros são dispostos desordenadamente



- Pesquisa → linear, varrendo todos os registros do arquivo no pior caso
  - Endereço do primeiro bloco do arquivo é recuperado do cabeçalho
  - Começando do primeiro, cada bloco é copiado para um *buffer*, onde deve-se verificar se cada registro do bloco satisfaz os critérios de pesquisa
  - Complexidade → $O(n)$, onde n é o número de blocos do arquivo

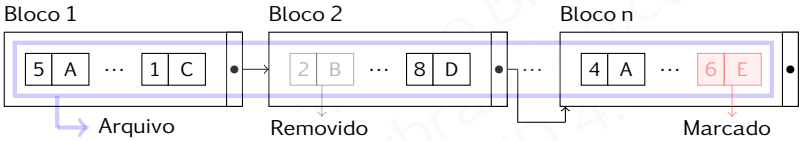
Tipos de Arquivo: Heap

Inserção eficiente, mas operações de alteração demandam pesquisa para encontrar o registro a ser alterado

- Inserção → registro arquivado na ordem em que é inserido
  - Endereço do último bloco do arquivo é recuperado do cabeçalho
  - Bloco copiado para um *buffer*, onde o novo registro é acrescentado, e o *buffer* é copiado de volta ao bloco
- Alteração → pode resultar em exclusão seguida de inclusão, caso o registro aumente de tamanho
  - Endereço do bloco do arquivo é recuperado via pesquisa
  - Bloco copiado para um *buffer*, onde o registro é modificado, e o *buffer* é copiado de volta ao bloco

Tipos de Arquivo: Heap

Operações de exclusão resultam em desperdício de espaço no bloco, demandando reorganização periódica do arquivo



- Exclusão → efetuada diretamente ou por marcação
  - Endereço do bloco do arquivo é recuperado via pesquisa
  - Direta → bloco copiado para um *buffer*, onde o registro é removido, e o *buffer* é copiado de volta ao bloco
  - Marcação → cada registro possui um byte extra, denominado marcador de exclusão. Assim, o bloco é copiado para um *buffer*, onde o marcador de exclusão do registro é modificado, e o *buffer* é copiado de volta ao bloco

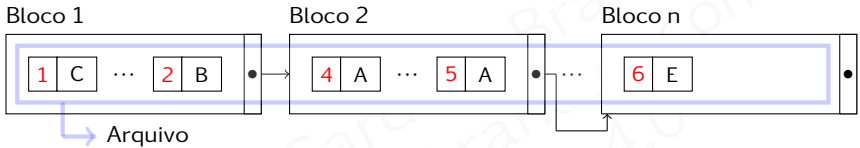
Tipos de Arquivo: Heap

Arquivo Direto (Relativo) → arquivo heap com registros de tamanho fixo, não espalhados, com blocos em alocação contígua

- Acesso simples a qualquer registro pela posição relativa no arquivo
- Não ajuda na pesquisa baseada em critérios
- Facilita a construção de índices no arquivo

Tipos de Arquivo: Sequencial

Arquivo organizado de forma que os registros são dispostos ordenadamente

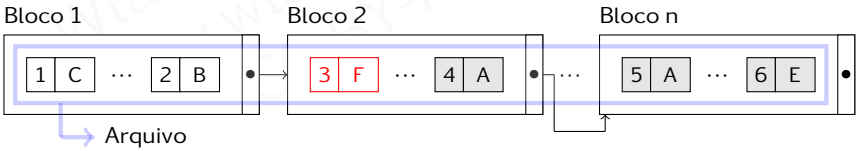


- Blocos em cilindros contíguos, minimizando tempo de busca
- Pesquisa → binária, varrendo pequena quantidade de registros se a pesquisa for feita com operadores $< \leq = > \geq$ sobre os campos de ordenação
  - Blocos intermediários são recuperados e segmentos são descartados até se encontrar registros que satisfaçam os critérios de pesquisa
  - Complexidade → $O(\log_2 n)$, onde n é o número de blocos do arquivo

Tipos de Arquivo: Sequencial

Operações de alteração são onerosas, pois podem demandar reorganização dos registros para preservação de ordem

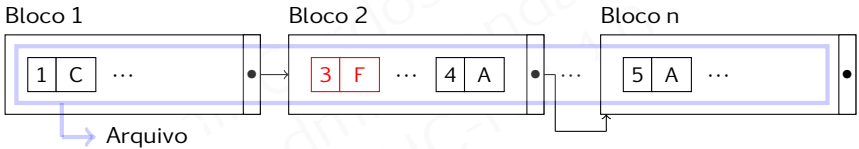
- Inserção → registro deve ser inserido na posição correta
  - Endereço do bloco onde registro deve ser inserido é recuperado via pesquisa
  - Deslocam-se registros para posições subsequentes, abrindo-se espaço para o registro a ser inserido
  - Blocos modificados nos deslocamentos são gravados de volta no disco



Tipos de Arquivo: Sequencial

Existem alternativas para desonerar a inclusão

- Espaços Vazios → diminui deslocamentos, mas problema reaparece com espaços vazios totalmente preenchidos



- Arquivo Temporário (Overflow) → arquivo heap onde novos registros são inseridos a um baixo custo
  - Periodicamente arquivo temporário é mesclado ao arquivo principal
  - Maior complexidade de pesquisa com necessidade de pesquisa linear no arquivo temporário caso o registro não seja encontrado no arquivo principal

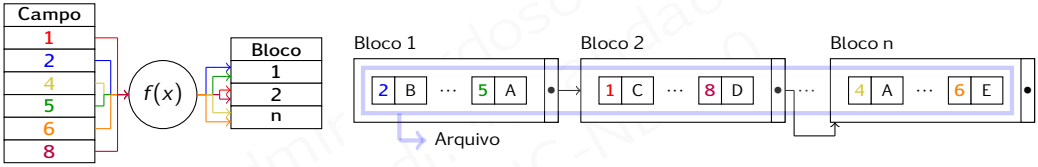
Tipos de Arquivo: Sequencial

Alteração no registro pode demandar seu reposicionamento

- Alteração → dependente da condição de pesquisa e do campo a ser alterado
  - Endereço do bloco do arquivo é recuperado via pesquisa
    - Pesquisa binária → critério envolver os campos de ordenação
    - Pesquisa linear → caso contrário
  - Bloco copiado para um *buffer*
    - Campo de ordenação não modificado → modifica-se o registro no *buffer*, copiando o *buffer* de volta ao bloco
    - Campo de ordenação modificado → deslocam-se registros gravando todos os blocos modificados
- Exclusão → igualmente dependente da condição de pesquisa

Tipos de Arquivo: Hash

Arquivo organizado de forma que os registros são distribuídos em blocos de acordo com uma *função hash*



- Pesquisa → tempo constante, localizando diretamente o bloco do registro se a pesquisa for feita com operador = sobre o campo *hash*
  - Função aplicada sobre o campo *hash* calcula o endereço do bloco do registro
  - Complexidade → $O(1)$

Tipos de Arquivo: Hash

Operações de alteração de registros eficientes

- Inclusão → pode gerar colisão e necessidade de expansão de arquivo
  - Aplica-se a *função hash* sobre o campo *hash* para calcular o endereço do bloco
  - Bloco copiado para um *buffer*, onde o novo registro é acrescentado, e o *buffer* é copiado de volta ao bloco
- Alteração → dependente da condição de pesquisa e do campo alterado
  - Endereço do bloco do arquivo é recuperado via pesquisa em tempo constante
  - Bloco copiado para um *buffer*
    - Campo *hash* não modificado → modifica-se o registro no *buffer*, copiando o *buffer* de volta ao bloco
    - Campo *hash* modificado → desloca-se o registro para outro bloco gravando os dois blocos modificados
- Exclusão → igualmente dependente da condição de pesquisa

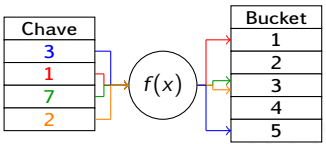
Tipos de Arquivo: Hash

Funções hash podem ser implementadas de formas diferentes

- Idealmente devem ser mantidas em memória primária, tornando muito eficiente o mapeamento de valores
- Implementações robustas distribuem valores de maneira uniforme, consumindo pouca memória primária
- Existem dois problemas muito comuns em implementações *hash*
  - Colisão → diferentes valores são mapeados para o mesmo endereço, que já pode estar ocupado
  - Expansão → não há mais endereços disponíveis para armazenamento de registros e o espaço de endereçamento precisa ser expandido

Tipos de Arquivo: Hash

Hashing Universal → mapeia conjunto de chaves de **tamanho variável** para espaço de tamanho m, tal que a probabilidade de colisão é $1/m$

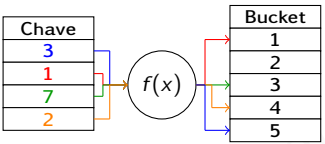


- Colisão → problema frequente quando $n \approx m$ ou $n \geq m$, onde n é o número de chaves
  - Endereçamento Aberto
  - Lista Encadeada
  - Hash Múltiplo
- Fator de Carga → n/m (max 0.75)

- Expansão → fundamental para evitar degradação da estrutura, proveniente do grande número de colisões
  - Múltiplas soluções, da reconstrução até o uso de múltiplas funções *hash*

Tipos de Arquivo: Hash

Hashing Perfeito → mapeia **conjunto fixo de chaves** para espaço de tamanho m, tal que não haja colisão



- Funções ocupam mais espaço em memória, com complexidade linear $O(n)$
- **Mínimo** → $n = m$, onde n é o número de chaves
  - Ordem preservada → $\Omega(n \log n)$
  - Ordem não preservada → $1.44n$

- **Expansão** → expandir significa reconstruir, já que conjunto de chaves é fixo
  - Hashing perfeito dinâmico pode ser a solução, mas complexa e de difícil implementação

Referências Bibliográficas

[1] Elmasri, Ramez; Navathe, Sham. *Fundamentals of Database Systems*. 7ed. Pearson, 2016.

[2] Silberschatz, Abraham; Korth, Henry F.; Sudarshan, S. *Database System Concepts*. 6ed. McGraw-Hill, 2011.

[3] Date, Christopher J. *An Introduction to Database Systems*. 8ed. Pearson, 2004.