UFU/FACOM/BCC
GBC053 - Gerenciamento de Banco de Dados - 2015/1
Trabalho de Implementação - Protótipo de um SGBD
Prof. Ilmério Reis da Silva

O trabalho semestral da disciplina em epígrafe tem como objetivo a implementação de um protótipo de Sistema Gerenciador de Banco de Dados-SGBD. O trabalho deve ser implementado e apresentado em grupos de até três alunos cada. O trabalho está dividido em três etapas, a saber: Etapa I - Formatação de Arquivos; Etapa II - Implementação de Índices; e Etapa III - Implementação de Operadores da Álgebra Relacional ou Etapa III Alternativa - Comparação de Varredura Sequencial com Acesso Indexado.

1 Etapa I - Formatação de Arquivos

A **Etapa I - Formatação de Arquivos** tem como objetivo gerar um arquivo não ordenado*(heap file)* para cada tabela do Banco de Dados.

O arquivo deve ser criado à partir de dados armazenados em um arquivo do tipo texto e de metadados armazenados em um catálogo, conforme arquitetura da Etapa I descrita na Figura 1.

A Etapa I pode ser subdividida em:

1. **Programa criaBD:** gera catálogo.

   O catálogo conterá os metadados do Banco de Dados. A saída do *criaBD* será apenas o nome do Esquema de Banco de Dados armazenado em um arquivo qualquer, que chamaremos de *catalogo*.

2. **Programa defineTabela:** insere metadados de uma tabela no catálogo.

   O *defineTabela* deverá inserir no *catalogo* os metadados de uma ou mais tabelas do Esquema de Banco de Dados. Para cada tabela deve-se inserir seu nome e uma lista de atributos com seus respectivos tipo e tamanho. O tipo deve ser inteiro de 4 bytes ou cadeia de caracter de tamanho variável.

3. **Programa carregaTabela:** insere dados de uma tabela em arquivo formatado.

O *carregaTabela* deverá inserir os dados obtidos de um arquivo texto em um arquivo com páginas de tamanho fixo. Considerando que a tabela poderá ter atributos do tipo: *Integer- inteiro de 4 bytes, VARCHAR(n)- cadeia de caracteres de tamanho variável com até n caracteres*, o formato da página deve permitir o armazenamento de registros de tamanho variável. A descrição dos atributos deverá ser armazenada no catálogo e a formatação do registro dependerá do tipo de atributos:

- para atributos de tamanho fixo, inteiros, basta armazenar o valor
- para atributos do tipo *VARCHAR* será necessário armazenar o tamanho da cadeia de caracteres e a própria cadeia, ou algum formato alternativo para registros de tamanho variável.

Uma sugestão de formato de registros e páginas é apresentada nas Figuras 2 e 3, respectivamente.

O resultado será um arquivo em disco com os dados do arquivo de entrada formatados de acordo com os tipos definidos no catálogo.

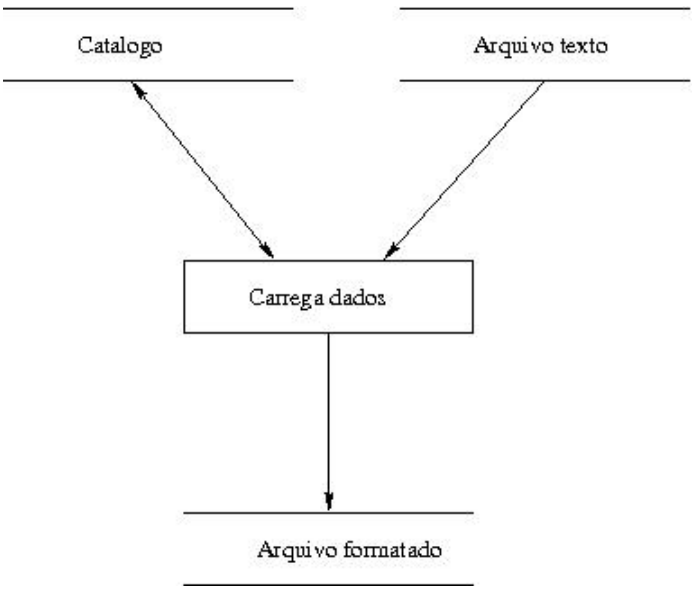

Figura 1: Arquitetura da Etapa I: carrega dados de um arquivo textual em um arquivo formatado de acordo com metadados descritos no catálogo.

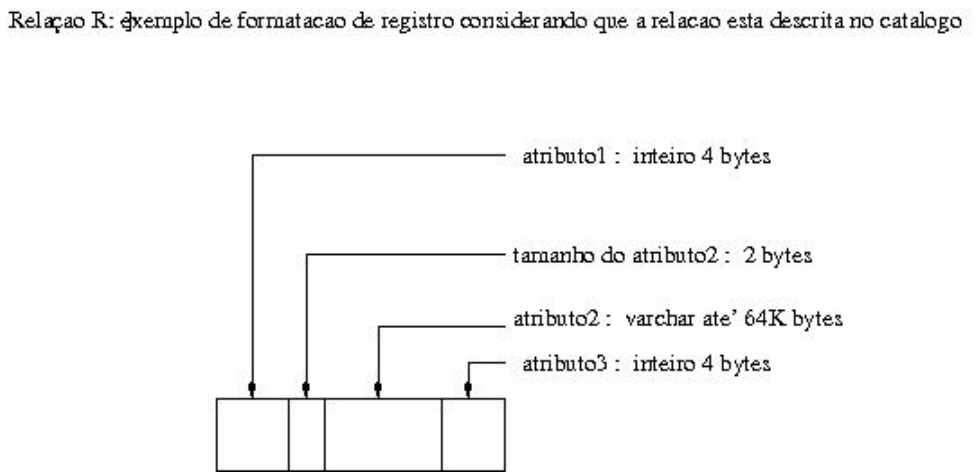

Figura 2: Um formato de registros de tamanho variável

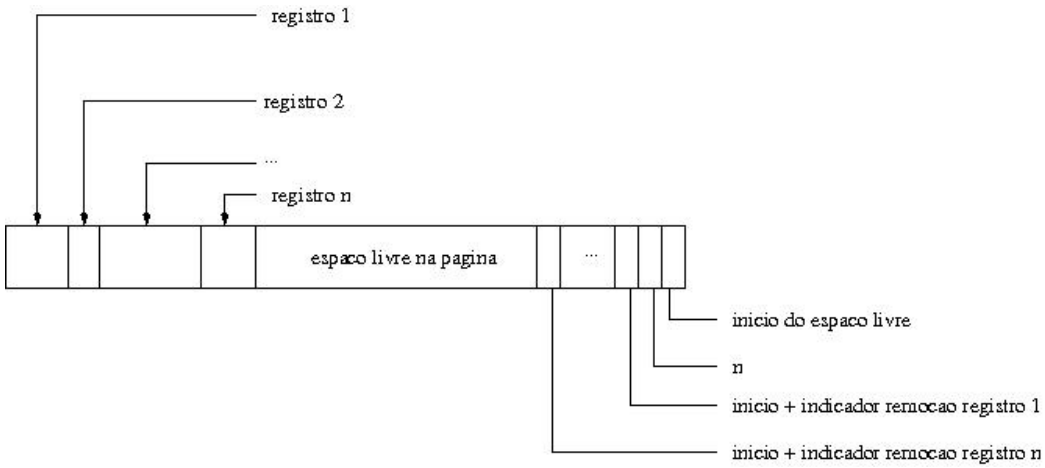

Figura 3: Um formato de armazenamento de páginas que permite registros de tamanho variável

2 Etapa II - Implementação de Índices

A **Etapa II - Implementação de Índices** tem como objetivo gerar um arquivo de índice associado ao arquivo não ordenado *(heap file)* criado na Etapa I. A Arquitetura dessa Etapa é apresentada na Figura 4. Observe na referida figura que o grupo tem opção de implementar um algoritmo de *bulkloading* ou um algoritmo com operações registro a registro. Sugestões de algoritmos de índice são apresentados nos slides apresentados em aula e na bibliografia da disciplina.

As principais caracterísitcas do índice são:

- A estrutura do índice deve ser dinâmica: *Árvore B+*, *Hash Extensível* ou *Hash Linear*.
- Em qualquer caso a entrada de dados do índice usará Alternativa 2, ou seja, $K^* =< K, RID >$, onde $RID =< PageId, SlotId >$.
- Considere que o arquivo de entrada é suficientemente grande para que o índice não caiba na memória primária, logo deverá ser construído e armazenado em disco. No caso de *Hash Extensível* considere que o diretório cabe na memória. No caso de *Àrvore B+* considere que todos os níveis, exceto as folhas, também cabem na memória.
- Deve-se implementar operações de busca, inserção e remoção.
- Deve-se implementar a construção do índice à partir do arquivo previamente carregado[1].
- A chave do índice será simples, ou seja, formada por apenas um atributo da tabela.
- A chave do índice será do tipo inteiro.
- A chave do índice será única no arquivo, ou seja, não existem chaves duplicadas.
- O índice será denso e não agrupado.

O índice será utilizado para implementação de algoritmos da Etapa III.

3 Etapa III - Implementação de Operadores da Álgebra Relacional

A **Etapa III - Implementação de Operadores da Álgebra Relacional** tem como objetivo implementar e comparar dois algoritmos de junção natural, que lerão dados armazenados em arquivos não ordenados *(heap files)* criados por meio do programa implementado na Etapa I. Dependendo dos algoritmos escolhidos pelo grupo, esta etapa usará índices criados por meio do programa implementado na Etapa II.

[1] A construção do índice pode ser por meio de varredura no arquivo seguida de inserção de entrada correspondente a cada registro, ou seja, basta omitir a inserção do registro no *heap file*. Em caso de *Árvore B+* a construção *bottom-up* também é uma opção de implementação. Uma dica para a construção *bottom-up* é criar as folhas à partir do *heap file* e, recursivamente, criar um nível do índice à partir do nível mais baixo até chegar a um nível com um só nodo, a raiz da árvore.

Durante o processamento da junção, independente do algoritmo implementado, deverá ser considerado que a quantidade de páginas de memória disponíveis para o *bufferpool* não é suficiente para armazenar a menor tabela. Logo uma política de substituição de páginas do *bufferpool* deverá ser implementada.

Considere a junção natural de duas tabelas,

$$R \bowtie S$$

onde, a menor das duas tabelas R e S não cabe no $buffer pool$. Defina B como o número de páginas disponíveis no $buffer pool$, de tal forma que R e S são maiores que B, implementar e comparar o desempenho de dois dos seguintes algoritmos de junção:

- **Laços Aninhados de Blocos**

  Split R em partições de $B-2$ páginas
  foreach *R_block* com $B-2$ *R_pages*
  foreach *S_page*
  $\forall r \in$ *R_block* % No buffer
  $\forall s \in$ *S_page* % No buffer
  If $(r_i = s_j)$ **output** $< r, s >$

- **Hash Join**

  foreach $r \in R$ **add** r **to buffer** $h(r_i)$ % *flush* quando necessário
  foreach $s \in S$ **add** s **to buffer** $h(s_j)$ % *flush* quando necessário
  % *Probing*
  for $l = 1 \cdots k$ { % percorre $k < B-1$ partições
  foreach $r \in R_l$ **add** r **to page** $h_2(r_i)$; % novo *hash* em memória
  foreach $s \in S_l$ { % percorre partição S_l
  compute $h_2(s_j)$;
  forall $(r \in R_l | r_i == s_j)$ % verifica *matchings* em R_l
  output $< r, s >$
  }
  clear *hash_table*
  }

- **Laços Aninhados com Índice**

  foreach *R_page*
  foreach $r \in$ *R_page*
  foreach $s \in S | r_i = s_j$ % via índice
  output $< r, s >$

  OBS: neste caso o grupo deve usar o índice implementado na Etapa II.

- **Sort Merge Join**

proc $smjoin(R, S, R_i, S_j)$
if not $sorted(R, R_i)$ **then** $sort(R, R_i)$;
if not $sorted(S, S_j)$ **then** $sort(S, S_j)$;
$r = first(R)$;
$s = first(S)$;
$g = s$; % grupo(partição) corrente de S
while $(r \neq eof) \wedge (g \neq eof)$ {
    while $(r.i < g.j)$ $r = next(R)$; % percorrendo R;
    while $(r.i > g.j)$ $g = next(S)$; % percorrendo S;
    $s = g$; % necessário se $r_i \neq g_j$
    while $(r.i == g.j)$ {
        $s = g$; % retorna busca na partição g de S
        while $(r.i == s.j)$ {
            output $< r, s >$;
            $s = next(S)$; % percorrendo S;
        }
        $r = next(R)$; % percorrendo R;
    }
    $g = s$; % próxima particao S;
}

A comparação deve ser feita por meio do número de IOs gastos para executar cada algoritmo e tempo de execução obtido experimentalmente na junção de tabelas de exemplo.

4 Etapa III Alternativa - Comparação de Varredura Sequencial com Acesso Indexado

A *Etapa III Alternativa - Comparação de Varredura Sequencial com Acesso Indexado* tem como objetivo implementar e comparar dois algoritmos de acesso aos registros de um aquivo. Deve-se gerar um *heap file* conforme descrito na Etapa I com um grande volume de dados. Esse volume deve ser uma ordem de grandeza superior à memória RAM do equipamento

onde será realizado o experimento. Então, deve-se gerar também para o *heap file* um índice Alternativa 2, conforme descrito na Etapa II.

A Etapa III consiste em comparar o tempo de uma varredura sequencial em todo o *heap file* e o tempo de acesso a todos os registro do arquivo de uma forma aletória.

5 Calendário e Pontuação

Os dias letivos do período de 13 a 22 de julho serão reservados para apresentação do trabalho.

A Etapa I será avaliada em 10 pontos e a Etapa II também será avaliada com 10 pontos. A Etapa III será usada como estratégia de recuperação de nota para os alunos frequentes que não obtiveram 60 pontos nas demais atividades.

Todas as apresentações devem ser agendadas com antecedência mínima de uma semana por meio de e-mail enviado ao professor.

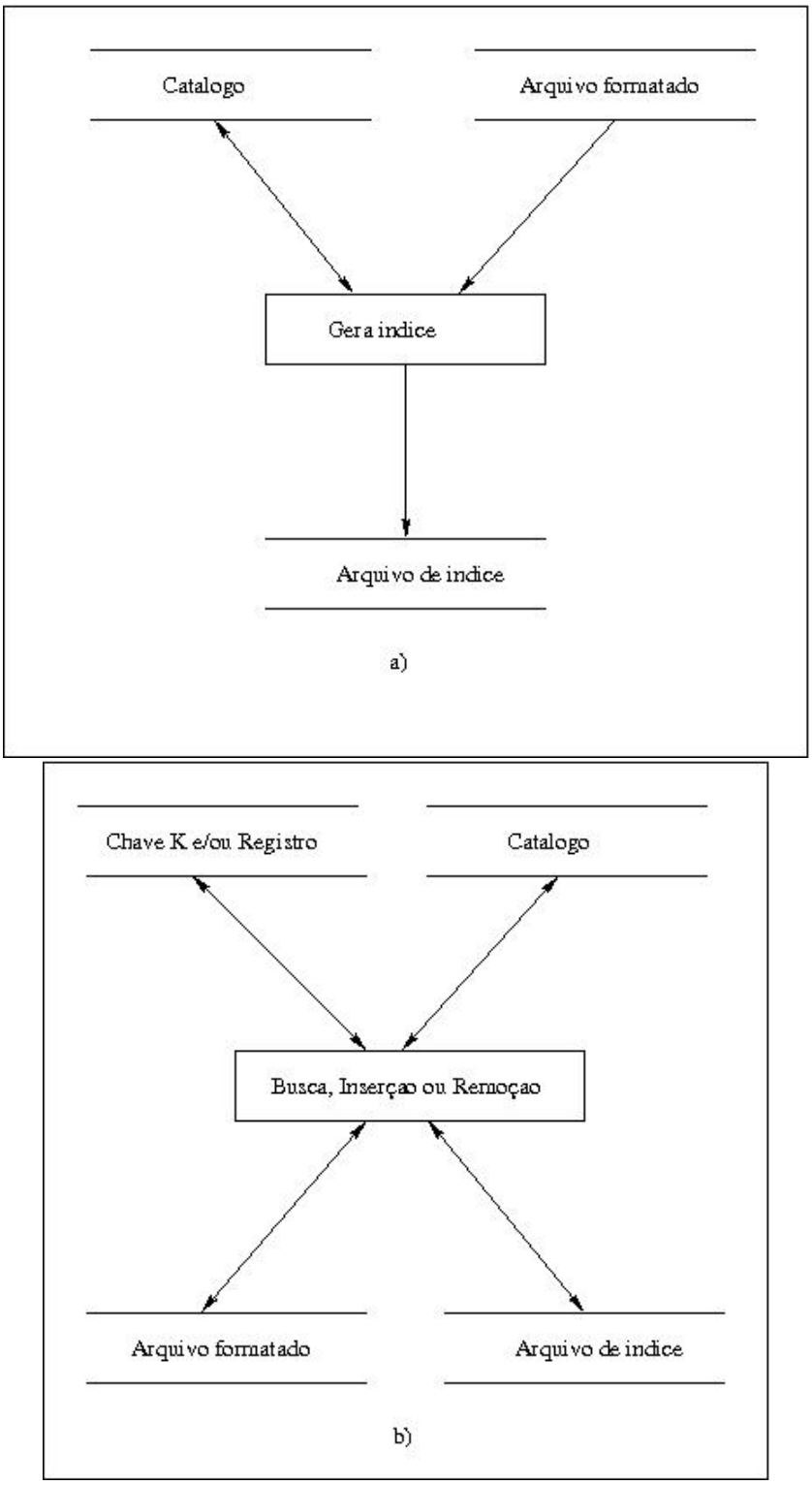

Figura 4: Arquitetura da Etapa II: a) gerar um arquivo de índice à partir de um arquivo formatado de acordo com metadados descritos no catálogo. b) implementar operações de busca, inserção e remoção.