**Trabalho Gerenciador de Banco de Dados**

Bruno Paiva de Oliveira 11511BCC013

Leonardo Sergi Molina 11511BCC041

Vinícius Gonzaga 11511BCC019

Victor Pignataro 11511BCC023

**Estruturação do projeto**

Nosso projeto é responsável por simular um gerenciador de banco de dados. E para representar o que ocorre na realidade, criamos as seguintes estruturas:

* **Gerenciador de espaço em disco (GED):** ele é o responsável por gerenciar o próprio disco, que no caso dessa simulação, será um arquivo .txt. O GED é encarregado de administrar os blocos, alocandoe desalocando blocos quando necessário. Além disso, quando precisamos da persistência dos dados, é ele quem garante que tal bloco foi realmente escrito no disco.
* **Gerenciador de buffer pool (GBP):** ele é o responsável por gerenciar o espaço de memória, administrando as informações do buffer pool (área de memória particionada em uma coleção de páginas). É o GBP o encarregado solicitar transferência de blocos do disco para a mémoria via GED, entre outras funções. Além disso, nosso GBP implementará a política de substituição Least Recently Used (LRU) para substituição dos frames.
* **Gerenciador de arquivos (GA):** ele é o responsável por gerenciar os arquivos referentes as tabelas. O tipo de arquivo utilizado nessa simulação é o heap file, ou seja, os dados dentro da página não são ordenados. E para gerenciar esses arquivos, será utilizado método de criação de um diretório de páginas para cada heap file.
* **Página:** estrutura criada para representar uma página, que é a menor unidade de dados, possuindo o mesmo tamanho de um bloco e um frame. Portanto, a unidade de informação que é lida ou escrita no disco é a página.
* **Registro:** estrutura criada para representar um registro, que são informações inseridas em um arquivo. Exemplo: (“BrunoP”), (“Victor”).

Algumas especificações que fixamos para nosso gerenciador de banco de dados foram:

* Nosso disco será um arquivo .txt, como dito anteriormente.
* A memória será um conjunto de páginas.
* Cada bloco conterá 2 registros de tamanho 6 bytes cada.
* Cada frame conterá 2 slots.
* No total a memória conterá 5 frames.

**Comunicação dos nossos 3 gerenciadores**

Memória

GBP

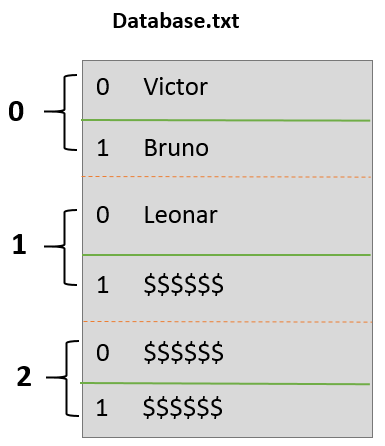
GA

GED

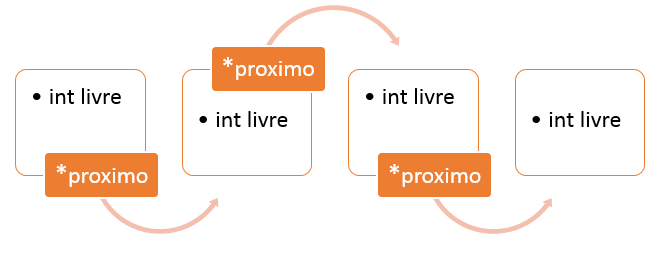
Disco

**Diagramação das estruturas**

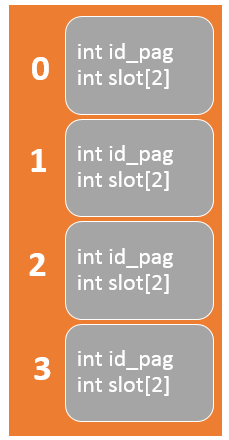
Disco: que no projeto é um arquivo .txt



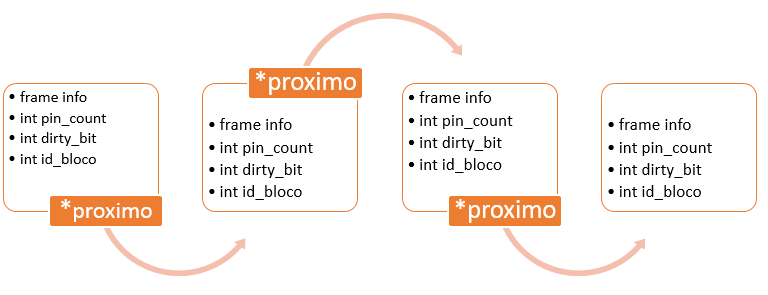
Gerenciador de espaço em disco:



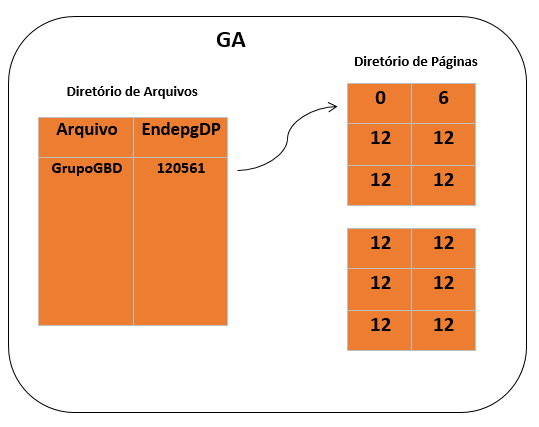
Buffer pool:



Gerenciador de buffer pool:



Gerenciador de arquivos:

****

**Definição de funções e estruturas**

**registro.h**

**Estrutura:**

registro

**Campos:**

*char str[6] –* Responsável por guardar um character de tamanho 6, pois cada registro no projeto terá um tamanho de 6 bytes.

**pagina.h**

**Estrutura:**

pagina

**Campos:**

*int id\_pagina* – Responsável por guardar uma identificação da página.

*registro slot[TAM\_FRAME]* – Reponsável por armazenar os registros, sendo que cada frame conterá no máximo 2 slots, pois *TAM\_FRAME* é uma variável fixada.

**Funções:**

*int cria\_registro(registro reg);*

**Função:** Responsável por inserir um novo registro em uma página.

**Entrada:**

r*egistro reg –* registro a ser inserido.

**Saída:** um inteiro, onde será 0 caso a operação ocorreu com sucesso e 1 caso contrário.

**ged.h**

**Estruturas:**

**Estrutura:**

bloco

**Campos:**

*int livre –* Informações sobre espaço do bloco: 0 se estiver cheio, 6 se conter um campo livre e 12 caso esteja vazio.

**Estrutura:**

ged

**Campos:**

*bloco\* blocos\_disco* - vetor responsável por guardar cada bloco.

*int controle –* indica se existe blocos que podem ser alocados.

**Funções:**

*int inicializa\_ged(int num\_blocos)*;

**Função:** Inicializa o GED, alocando *num\_blocos*, sendo este valor escolhido de forma arbitrária pelo usuário.

**Entrada:**

*int num\_blocos -* variável do tipo inteiro que indica o número de blocos definidos para a criação do GED.

**Saída:** variável do tipo inteiro, sendo 0 caso a operação ocorreu sucesso e 1 caso contrário.

*int aloca\_bloco(void)*;

**Função:** Tem a função de criar um novo bloco e retornar o endereço do mesmo, sendo sempre um bloco livre, possuindo 12 bytes de espaço.

**Entrada:** vazio

**Saída:** variável do tipo inteiro, retornando identificador do bloco.

*int desaloca\_bloco(int id\_bloco)*;

**Função:** Responsável por desalocar um bloco.

**Entrada:**

*int id\_bloco* - identificador do bloco.

**Saída:** variável do tipo inteiro, sendo 0 caso a operação ocorreu com sucesso e 1 caso contrário.

*int escreve\_bloco(pagina pagina)*;

**Função:** Responsável por escrever uma página no bloco.

**Entrada:**

*pagina pagina -* variável do tipo “pagina”, que identificará a página a ser escrita.

**Saída:** variável do tipo inteiro, sendo 0 caso a operação ocorreu com sucesso e 1 caso contrário.

*pagina ler\_bloco(int id\_bloco)*;

**Função:** Responsável por ler um bloco e retornar uma página correspondente.

**Entrada:**

*int id\_bloco* - identificador do bloco a ser carregado pra memória.

**Saída:** variável do tipo “pagina”, sendo a página correspondente ao bloco.

**gbp.h**

Possui a definição de uma variável fixa MEMSZ 5, que define o número de páginas (frames) que podem ser comportadas na memória.

Possui uma variável *pagina \*memoria*, que é um Array de páginas, com o objetivo de simular uma memória.

**Estrutura:**

frame\_info

**Campos:**

*int pin\_count* – Responsável por manter o número de resquisições do frame.

*int dirty\_bit* – Responsável por indicar se o frame foi modificado (valor 1) ou não (valor 0).

*int id\_bloco* – Guardar endereço do bloco correspondente a esse frame.

**Estrutura:**

gbp

**Campos:**

*frame\_info frames[MEMSZ]* - vetor que define o número de frames que existem e realiza o controle deles.

**Funções:**

*pagina verifica\_quant\_frames(int arquivo\_id);*

**Função:** Checa se existem frames livres e retorna uma página(se for necessário criar uma nova página, esta função chama o ‘cria\_pagina’).

**Entrada:**

*int arquivo\_id*  - identificador do arquivo para checar as páginas correspondente a ele.

**Saída:** Identificador da página livre.

*int inserir\_registro(int arquivo\_id, registro reg);*

**Função:** Responsavel por inserir um registro em uma tabela. Comunica com o Gerenciador de Arquivos (GA) e chama a função ‘checa\_espaco\_bloco’.

**Entrada:**

*int arquivo\_id* – Identificador do arquivo para saber em qual tabela adicionar.

*registro reg* – Registro a ser inserido.

**Saída:** variável do tipo inteiro

*int gbp\_lru(void);*

**Função:** Responsável por aplicar a política de substituição Least Recently Used (LRU).

**Entrada:** void

**Saída:** variável do tipo inteirosendo 0 caso a operação ocorreu com sucesso e 1 caso contrário.

*int habilitar\_transferencia(void);*

**Função:** Chama a funcao ‘ler\_bloco’ do GED, tem o objetivo de transferir o bloco para a memória.

**Entrada:** void

**Saída:** variável do tipo inteiro.

*int busca\_registro(int arquivo\_id, int id\_pagina,char \*str);*

**Função:** Responsável por retornar o número identificador do slot de um registro.

**Entrada:**

*int arquivo\_id* - identificador do arquivo que se deseja buscar o registro.

i*nt id\_pagina –* identificador da página onde o registro se encontra.

*char \*str* – conteúdo a ser buscado.

**Saída:** variável do tipo inteiro, retornando o número do slot.

*int atualiza\_registro(int arquivo\_id, int id\_pagina, registro reg, int slot);*

**Função:** Responsável por atualizar um registro em um frame.

**Entrada:**

*int arquivo\_id* - identificador do arquivo onde contém o registro.

*int id\_pagina* - identificador da página onde contém o registro.

*registro reg* – novo registro que substituirá o anterior.

*int slot* - slot a ser atualizado.

**Saída:** variável do tipo inteiro, sendo 0 caso a operação ocorreu com sucesso e 1 caso contrário.

*int remover\_registro(int arquivo\_id, int id\_pagina, int slot);*

**Função:** Responsável por remover um registro e atualizar espaço livre.

**Entrada:**

*int arquivo\_id* - identificador do arquivo que contém o registro a ser removido.

*int id\_pagina* - identificador da página que contém o registro a ser removido.

*int slot* – localização do registro no frame.

**Saída:** variável do tipo inteiro, sendo 0 caso a operação ocorreu com sucesso e 1 caso contrário.

*int escrever\_pagina(pagina pagina);*

**Função:** Responsável por escrever uma página no disco.

**Entrada:**

*pagina pagina* - recebe a página a ser escrita no disco.

**Saída:** variável do tipo inteiro, sendo 0 caso a operação ocorreu com sucesso e 1 caso contrário.

*int pagina\_na\_memoria(int arquivo\_id, int id\_pagina);*

**Função:** Responsável verificar se uma página já esta carregada na memória.

**Entrada:**

*int arquivo\_id* - identificador do arquivo que contém a página.

*int id\_pagina* - identificador da página a ser procurada.

**Saída:** variável do tipo inteiro, sendo -1 caso não encontrado e um 1 caso encontrado.

*int numero\_frame(int id\_bloco);*

**Função:** Responsável por retornar o número do frame de um determinado bloco.

**Entrada:**

*int id\_bloco* - identificador do bloco a ser procurado.

**Saída:** variável do tipo inteiro, sendo -1 caso não encontrado e um valor >=0 correspondente ao frame.

*void frame\_modificado(pagina pagina);*

**Função:** Responsável por alterar o campo *dirty\_bit* e dizer que a página foi alterada.

**Entrada:**

*pagina pagina -* recebe a página que teve seu(s) registro(s) alterado(s).

**Saída:** void.

*void pagina\_setUse(pagina pagina, int flag);*

**Função:** Responsável por incrementar o *pin\_count*, caso *flag = 1 s*ignifica que devemos incrementar o pin\_count da página. Se for *flag = 0* deverá decrementar o *pin\_count*.

**Entrada:**

*pagina pagina* - recebe a página para alterar o pin\_count.

*int flag* – com valor sendo 0 ou 1, é responsável por dizer se devemos incrementar o decrementar o pin\_count da página.

**Saída:** void.

**ga.h**

**Estrutura:**

diretorio\_de\_blocos

**Campos:**

*int espaco\_livre* – responsável por indiciar o espaço livre do bloco.

*int id\_bloco –* identificador do bloco.

**Estrutura:**

linha\_tabela

**Campos:**

*int arquivo\_id –* identificador do arquivo.

*diretorio\_blocos\* header –* ponteiro que dirá qual é o primeiro bloco de um arquivo.

**Estrutura:**

diretorio\_arquivos

**Campos:**

*linha\_tabela tabelas[5] -* vetor da estrutura *linha\_tabela* com tamanho 5, dizer que existe apenas 5 entradas no diretório de arquivos.

**Funções:**

*int criar\_tabela(char\* tabela);*

**Função:** Possui o objetivo de criar uma nova tabela e chamar ‘cria\_diretorio’ para alocar um espaço no ‘diretorio\_arquivos’ para essa nova tabela (arquivo).

**Entrada:**

*char \*tabela –* nome da tabela a ser criada.

**Saída:** variável do tipo inteiro, sendo 0 caso a operação ocorreu com sucesso e 1 caso contrário.

*int cria\_diretorio(int arquivo\_id, int id\_bloco);*

**Função:** Responsável por alocar um espaço no diretorio de arquivos, recebendo um *arquivo\_id* e um *id\_bloco* provenientes do GED (função ‘aloca\_bloco’) para ser a header page, e por fim retornar o endereço da nova tabela.

**Entrada:**

*int arquivo\_id* - identificador do arquivo.

*int id\_bloco* - identificador do bloco.

**Saída:** variável do tipo inteiro correspondente ao endereço da nova tabela.

*int checar\_espaco\_blocos(int arquivo\_id, int espaco\_necessario)*;

**Função:** Reponsável por checar se existe bloco com espaço livre (do tamanho do espaco necessário ou menor), e se sim, verificar se o bloco já está carregado na memória, se não, carregá-lo.

**Entrada:**

*int arquivo\_id* - identificador do arquivo para checar bloco.

*int espaco\_necessario* – espaço livre que se deseja procurar em um bloco.

**Saída:** variável do tipo inteiro correspondente ao bloco.

*int cria\_pagina(void);*

**Função:** Responsável por chamar o GED para alocar um novo bloco, além de criar uma nova página e retornar o endereço dessa nova página criada.

**Entrada:** void

**Saída:** variável do tipo inteiro correspondente a nova página.

*int atualiza\_espaco(int arquivo\_id, int id\_pagina, int num);*

**Função:** Reponsável por atualizar o campo *espaco\_livre* de uma página(o campo *num* teria valor 1 ou -1).

**Entrada:**

*int arquivo\_id* - identificador do arquivo que contém a página que se quer atualizar..

*int id\_pagina* - identificador da página que se quer atualizar.

*int num* – responsável por dizer se devemos incrementar ou decrementar o espaço livre de uma página (1 – decrementar; -1 – incrementar).

**Saída:** variável do tipo inteiro, sendo 0 caso a operação ocorreu com sucesso e 1 caso contrário.

*int busca\_pagina(int id\_arquivo, int id\_pagina);*

**Função:** Responsável por retornar o endereço de uma página a ser buscada.

**Entrada:**

*int id\_arquivo* - identificador do arquivo que contém a página.

*int id\_pagina* - identificador da página a ser buscada.

**Saída:** variável do tipo inteiro correspondente ao endereço da página.

**Operações**

**Inicializando o gerenciador de banco de dados**

Para inicializar todo o nosso sistema, a seguinte função, proveniente da estrutura GED, é chamada:

* int inicializa\_ged(int num\_blocos);

**Criar uma nova tabela**

Para criar uma nova tabela, é chamada a função do GA *criar\_tabela*. Posteriomente, essa chama a *cria\_diretorio.* E por fim, teremos um novo campo no nosso diretório de arquivos, constituído pelo identificador dessa nova tabela e o primeiro bloco correspondente.

**Inserindo um novo registro**

Para inserir um novo registro, o GBP chama a função *inserir\_registro*. Essa função comunica com o GA e chama *checar\_espaco\_blocos* para verificar em qual bloco poderemos inserir esse novo registro. Poderá acontecer dois casos:

1. Não há blocos livres: deveremos chamar a função *cria\_pagina* do GA, que será responsável por chamar *aloca\_bloco* do GED. Após isso, teremos criado um novo bloco e ainda no GED, será chamada a função *ler\_bloco,* que retornará para o GBP o endereço do bloco que foi alocado. O GBP, por sua vez, chamará a função *habilita\_transferencia* que ficará encarregada de efetivamente transferir o bloco do disco para a área de buffer pool*.* É chamada a função *verifica\_quant\_frames* para saber se existem frames disponíveis, e se não tiver, a função *gbp\_lru* é invocada para aplicar a política de substituição. Por fim carrega essa página para memória. Após a inserção, a função do GA *atualiza\_espaco* é chamada para atualizar o espaço livre dessa nova página.
2. Existe bloco com espaço livre para esse registro: será chamada a função chama *pagina\_na\_memoria* para verificar se a página correspondente ao bloco em que iremos inserir o registro está na memória. Se estiver, ele retorna o endereço da página correspondente. É chamada a função *verifica\_quant\_frames* para saber se existem frames disponíveis, e se não tiver, a função *gbp\_lru* é invocada para aplicar a política de substituição. Após a inserção, a função *frame\_modificado* é chamada para atualizar as informações do frame que contém a página que acabou de ter um novo registro, alterando assim, o campo *dirty\_bit* dela. Além disso, a função do GA *atualiza\_espaco* é chamada para atualizar o espaço livre dessa nova página.

**Removendo um registro**

Para se remover um registro, a função *remover\_registro* do GBP é chamada. Tal função chama a função *busca\_registro* do GBP, que retornará o número do slot desse registro. Com esse número, a função *remover\_registro* consegue remover o registro e posteriomente chama *atualiza\_espaco* do GA para atualizar o espaço.

**Atualizando um registro**

Para se atualizar um registro, a função *atualiza\_registro* do GBP é chamada. Tal função, semelhante a operação removendo, chama a função *busca\_registro,* que retornará o número do slot do registro que se deseja alterar. Com esse número, a função atualiza*\_registro* consegue atualizar o registro e posteriomente chama *atualiza\_espaco* do GA para atualizar o espaço.

**Escrever página no disco**

Para se escrever uma página no disco, é chamada a função *escrever\_página* no GBP. Tal função chama *escreve\_bloco* no GED, que finalmente realiza a persistência dos dados.