Trabalho Gerenciador de Banco de Dados

Bruno Paiva de Oliveira 11511BCC013
Leonardo Sergi Molina 11511BCC041
Vinícius Gonzaga 11511BCC019
Victor Pignataro 11511BCC023

Estruturação do projeto

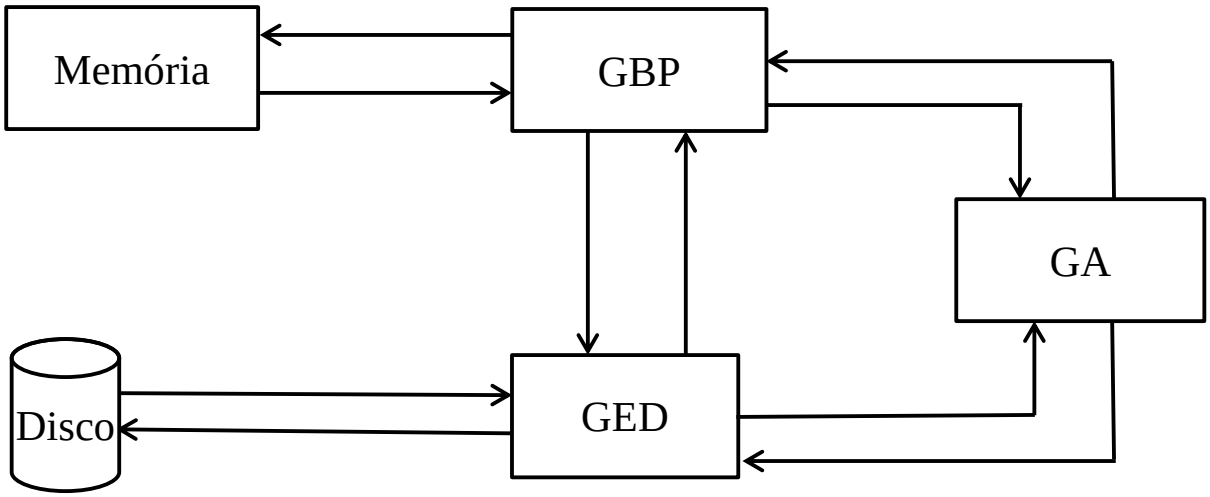
Nosso projeto é responsável por simular um gerenciador de banco de dados. E para representar o que ocorre na realidade, criamos as seguintes estruturas:

- **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, alocando e 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:

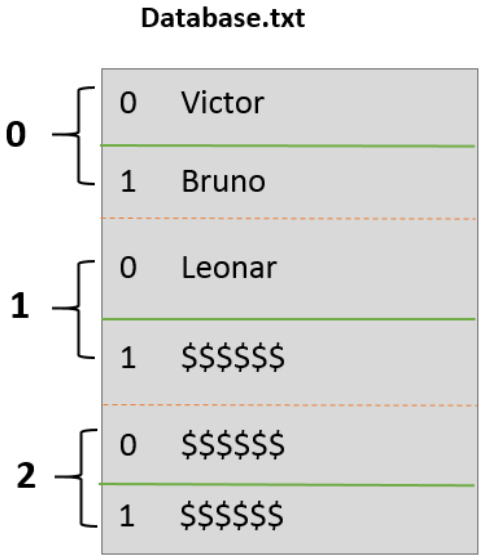
- Nosso disco será um arquivo .txt, como dito anteriormente.
- A memória será um conjunto de páginas.
- Cada bloco conterá 2 registros de tamanho 6 bytes cada.
- Cada frame conterá 2 slots.
- No total a memória conterá 5 frames.

Comunicação dos nossos 3 gerenciadores

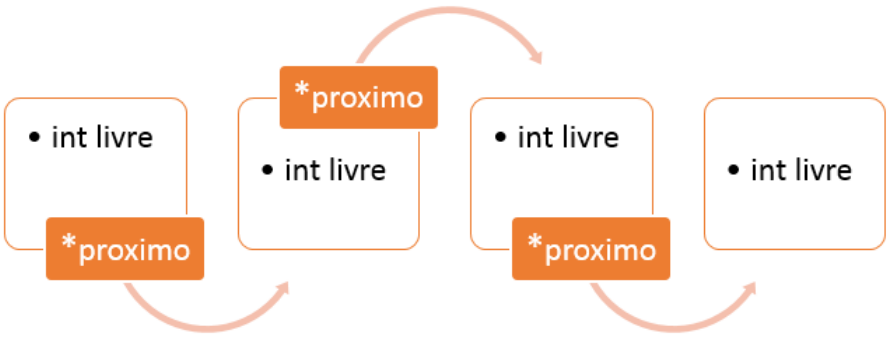


Diagramação das estruturas

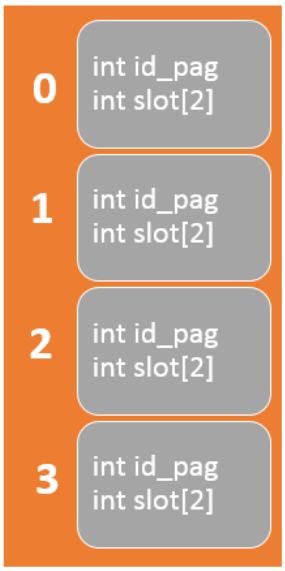
Disco: que no projeto é um arquivo .txt



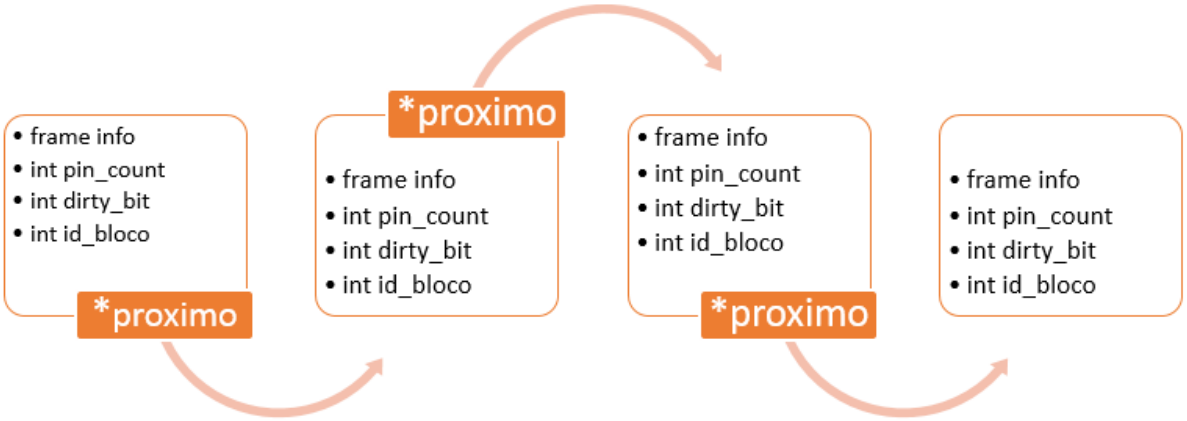
Gerenciador de espaço em disco:



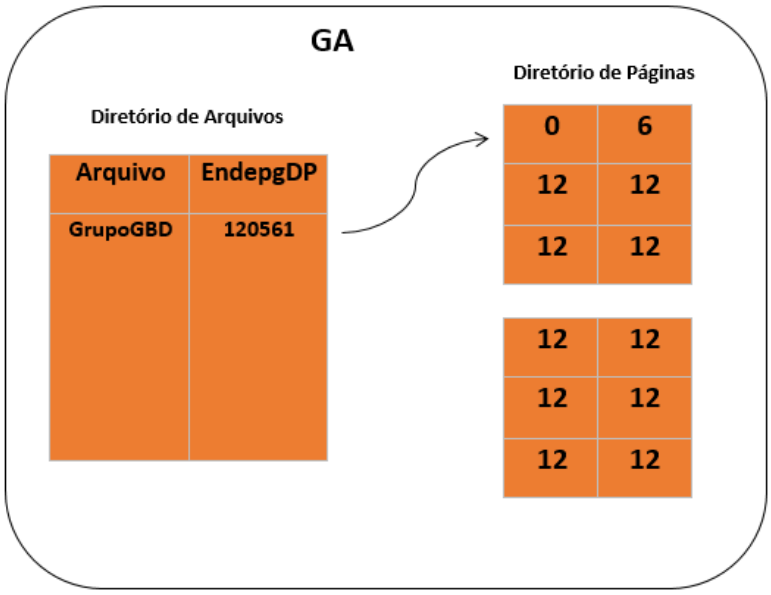
Buffer pool:



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.