Mini Simulador de Sistema de Gestão de Metadados

Objetivo

Construir um mini simulador de Sistema de Gestão de Metadados que suporte a criação, atualização, exclusão e recuperação eficientes de documentos JSON representando metadados que podem ser associados a dados de outros sistemas.

A implementação deve contemplar os seguintes elementos básicos:

1. *Datafile*: implementado como um arquivo binário, criado sobre o *file system* do sistema operacional, dividido em *datablocks* de mesmo tamanho;
2. *Buffers*: área de memória para acesso de leitura e escrita dos *datablocks*, implementando algoritmo de *cache write-back*, com política de descarte *clock*;
3. *B-Tree+*: índice tendo por chave de busca um ID serial atribuído a cada documento JSON de metadados;
4. Interface de usuário: contemplando as seguintes operações:
    a. Inserir um documento de metadados: digitar ou ler de um arquivo (abrir) um documento JSON, atribuir um ID serial e armazená-lo no Banco de Dados.
    b. Recuperar um documento de metadados pelo seu ID: digitar o ID serial de um documento JSON, recuperar seu *rowId* da *B-Tree+* e recuperar seu conteúdo dos *datablocks* do *datafile*.
    c. Recuperar um documento de metadados pelo valor de uma *tag* JSON: digitar o nome da tag e seu valor, varrendo sequencialmente todos os documentos JSON recuperando cada um que possuir aquela *tag* com aquele valor.
    d. Excluir um documento de metadados pelo seu ID: digitar o ID serial de um documento JSON e excluí-lo da *B-Tree+* e do *datafile*.
    e. Carregar um lote de documentos JSON: para simplificar os testes, deve ser possível digitar um número de 1 a 1000 documentos JSON a serem carregados em lote.
5. DESAFIO (valendo 1,0 ponto adicional): as operações devem poder ser executadas por meio de uma RESTful API.

Todo o desenvolvimento, desde a análise e o projeto até a implementação, deverá ser realizado empregando orientação a objetos.

Será tolerada a utilização de qualquer linguagem de programação baseada em objetos (Java, C++, C#, Python, etc.).

Entregáveis

Deverá ser entregue obrigatoriamente documentação digital composta por:

1. Documentação de análise e projeto em UML. Os diagramas a serem entregues são:
    a. Use Cases;
    b. Diagrama de Classes (incluindo a representação de atributos e métodos)
    c. Diagramas de Sequência.
2. Código fonte comentado.
3. Instruções detalhadas para instalação e testes.

Todos os integrantes do grupo deverão apresentar o trabalho no laboratório, em data a ser definida pelo professor.

Detalhamento de Requisitos

1. ***Datafile***
   1.1. Deve ser criado um *datafile* de 256 MB (Megabytes) = 262.144 KB (Kilobytes).
   1.2. O arquivo deve ser binário.
   1.3. O acesso deve ser randômico.
   1.4. Deve ser garantida a persistência dos dados entre as execuções do mini simulador.
   1.5. O(s) primeiro(s) *datablock*(*s*) do *datafile* deve(m) ser utilizado(s) para armazenamento de informações de controle e do dicionário de dados (estabeleçam o que é necessário armazenar conforme sua implementação).
2. ***Datablocks***
   2.1. O *datafile* deve estar dividido em *datablocks* de 4KB (4.096Bytes = 2^{12} Bytes).
   2.2. Logo, o *datafile* será formado por 65.536 *datablocks* (2^{16} *datablocks*).
   2.3. Cada *datablock* deve possuir um cabeçalho com diversas entradas, cada qual indicando o Byte inicial e o tamanho de cada documento JSON armazenado nele.
3. ***Rowid***
   3.1. O *rowid* de um documento JSON é formado por 16 + 16 bits:
      - Endereço do *datablock*: 16 bits (2 Bytes);
      - Entrada do documento na tabela de endereços do *datablock*: 16 bits (2 Bytes).
   3.2. Um documento JSON pode iniciar em um *datablock* e estender-se por vários outros, sendo que cada trecho armazenado em um *datablock* aponta para o *rowid* da próxima parte até a última, que não aponta para nenhum outro *datablock*.
   3.3. Um *datablock* pode armazenar trechos de diversos documentos JSON.
4. ***B-Tree+***
   4.1. Sobre esta tabela, deverá ser implementado um Índice Secundário B-Tree+.
   4.2. A chave de busca (k) é o ID do documento JSON, que ocupa 32bits (4 Bytes).
   4.3. Deve ser calculada a ordem **d** dos nós ramo e dos nós folha da árvore.
   4.4. O endereço do *datablock* raiz da árvore deve ser armazenado no dicionário de dados.
5. ***Buffer***
   5.1. O *buffer* deve ser composto por 256 *frames*.
   5.2. O algoritmo de seleção de *frames* para descarte deve se dar por meio do algoritmo de *clock*.
   5.3. Ao ser requisitado um *datablock*, deverá ser implementado um algoritmo de busca em memória para localização (ou não) do mesmo no *buffer*.

Observações

Em caso de plágio da web, de semestres anteriores ou de outra turma, cuja comprovação será realizada pelo professor junto ao grupo, também será atribuída nota ZERO!