

Documentação do TP2 - Banco de dados

Integrantes da equipe: Guilherme Kauê, Kaio Carvalho, Washington

Estrutura de cada arquivo de dados e índices

1. **Arquivo de dados:** cada linha contém os seguintes campos, nesta ordem: ID, Título, Ano, Autores, Citacoes, Data_Atualizacao, Snippet.
O acesso é feito por leitura sequencial (programa findrec) ou por offset (programas seek1 e seek2).
O tamanho do bloco lógico usado é de 4096 bytes.
Cada programa calcula blocos lidos e blocos totais chamando bytes_para_blocos() (definido em util.hpp).
2. **Índice primário:** Armazena pares (ID, offset) em binário, onde ID é um inteiro de 4 bytes e offset é a posição no arquivo de dados em 8 bytes, ocupando um total de 12 bytes.
A busca é realizada por varredura linear em disco (seek1.cpp).
3. **Índice secundário:** Armazena pares (Título, offset) em binário, com o comprimento do título contendo 4 bytes juntamente com o offset do registro.
A busca sequencial é feita pelo título exato.
4. **Arquivo de índice B+:** a classe ArvoreBMais representa um índice persistente simples.
Implementação reduzida, ou seja, grava pares (chave, posição) de forma sequencial em arquivo binário.

Quais fontes formam cada programa

1. **upload:** upload.cpp, registro.cpp, util.cpp
2. **findrec:** findrec.cpp, registro.cpp, util.cpp
3. **seek1:** seek1.cpp, registro.cpp, util.cpp
4. **seek2:** seek2.cpp, registro.cpp, util.cpp
5. **Estruturas auxiliares:** hashing_estatico.cpp, arvore_bmais.cpp

As funções que cada fonte contém e o papel de cada uma delas

1. **upload.cpp**

main(): executa o comando upload <arquivo.csv>, cria o diretório data/db + arquivos, lê cada linha do csv, converte em registro, escreve no arquivo de dados e atualiza os índices primário e secundário.

2. **findrec.cpp**

main(): busca direta no arquivo de dados, recebe id como argumento, lê sequencialmente dados.csv, converte linhas em registro e compara o id; calcula blocos lidos e totais.

3. **seek1.cpp**

main(): busca por id via índice primário, lê indice_primario.dat até encontrar o id correspondente, usa o offset armazenado para buscar, reconstrói o registro.

4. **seek2.cpp**

main(): busca por título via índice secundário, lê indice_secundario.dat, compara o título buscado e se for igual, lê o registro correspondente.

5. **hashing_estatico.cpp**

hashingEstatico(const std::string & arquivo, int qtd, int tam): Inicializa o hashing com arquivo, número de buckets e tamanho.

int funcao_hash(int chave): calcula chave % qtd_buckets.

void inserir(const Registro &r): Adiciona registro no arquivo binário.

Registro buscar(int chave): Percorre o arquivo e retorna o registro correspondente.

6. **arvore_bmais.cpp**

ArvoreBMais::ArvoreBMais(const std::string &arquivo, int o): Cria arquivo binário para a árvore.

void inserir(int chave, long posicao): adiciona par chave, posição no arquivo.

long buscar(int chave): busca linear no arquivo, retorna a posição ou -1.

7. **registro.cpp**

std::vector<char> Registro::serializar() const: Serializa o registro em bytes (com prefixos de tamanho e big endian).

Registro Registro::desserializar_from_buffer(const char* buf, size_t size): Reconstrói o registro a partir do buffer binário.

Funções auxiliares: write_u32, write_i64, read_u32, read_i64: manipulam tipos binários fixos.

8. **util.cpp**

std::vector<std::string> dividir_csv(const std::string &linha_inicial, std::istream *stream_restante): Divide uma linha CSV em campos, respeitando aspas duplas e quebras de linha.

`Registro campos_para_registro(const std::vector<std::string> &campos):` Converte um vetor de campos em struct Registro.

`std::string registro_para_csvline(const Registro &r):` Serializa o registro para formato CSV.

`long bytes_para_blocos(long bytes, long tam_bloco):` Calcula o número de blocos a partir de bytes.

Divisão de tarefas – o que cada um fez

Guilherme Kaue: fez os seguintes códigos: `upload.cpp`, `findrec.cpp`, `seek1.cpp`, `seek2.cpp`, fez toda a documentação.

Kaio Carvalho: fez os seguintes códigos: `hashing_estatico.cpp`, `arvore_bmais.cpp`, `registro.cpp`, `util.cpp`, fez todos os códigos hpp.

Washington: revisou o repositório e fez os testes dos programas, sugerindo algumas mudanças.