DOCUMENTAÇÃO DO TP2 DE BANCO DE DADOS 1

Equipe:

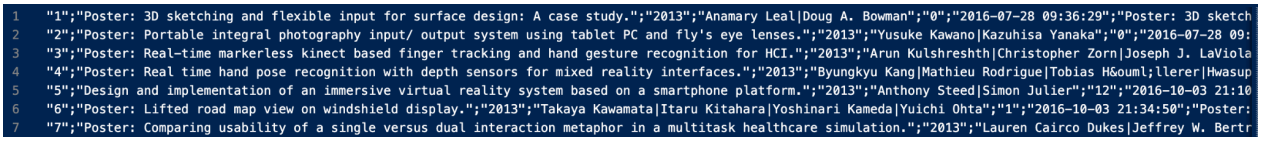
- Alexandre da Silva Tupinambá [21952463]
- Marcos André Araújo Costa [22050949]

Com o objetivo de registrar as decisões de projeto tomadas no desenvolvimento do Trabalho Prático 2 da disciplina de Banco de Dados 1, essa documentação foi elaborada.

A estrutura interna do arquivo *artigos.csv* é a seguinte:

Campo	Tipo	Descrição
ID	inteiro	Código identificador do artigo
Título	alfa 300	Título de artigo
Ano	inteiro	Ano de publicação do artigo
Autores	alfa 150	Lista dos autores do artigo
Citações	inteiro	Número de vezes que o artigo foi citado
Atualização	data e hora	Data e hora da última atualização dos dados
Snippet	alfa entre 100 e 1024	Resumo textual do artigo

Cada linha do arquivo armazena um registro em que os campos são separados por ";", como mostra a imagem a seguir:



```
"1";"Poster: 3D sketching and flexible input for surface design: A case study.";"2013";"Anamary Leal|Doug A. Bowman";"0";"2016-07-28 09:36:29";"Poster: 3D sketch
"2";"Poster: Portable integral photography input/ output system using tablet PC and fly's eye lenses.";"2013";"Yusuke Kawano|Kazuhisa Yanaka";"0";"2016-07-28 09:
"3";"Poster: Real-time markerless kinect based finger tracking and hand gesture recognition for HCI.";"2013";"Arun Kulshreshth|Christopher Zorn|Joseph J. LaViola
"4";"Poster: Real time hand pose recognition with depth sensors for mixed reality interfaces.";"2013";"Byungkyu Kang|Mathieu Rodrigue|Tobias H&ouml;llerer|Hwasup
"5";"Design and implementation of an immersive virtual reality system based on a smartphone platform.";"2013";"Anthony Steed|Simon Julier";"12";"2016-10-03 21:10
"6";"Poster: Lifted road map view on windshield display.";"2013";"Takaya Kawamata|Itaru Kitahara|Yoshinari Kameda|Yuichi Ohta";"1";"2016-10-03 21:34:50";"Poster:
"7";"Poster: Comparing usability of a single versus dual interaction metaphor in a multitask healthcare simulation.";"2013";"Lauren Cairco Dukes|Jeffrey W. Bertr
```

1. **upload.cpp(Alexandre e marcos):** O programa cria arquivos de dados, índice primário e índice secundário em formato binário.

Includes:

- **hash.hpp**
- **prim.hpp**
- **sec.hpp**
- **stdio.h**
- **stdlib.h**
- **string**
- **fstream**
- **iostream**
- **cstring**

Funções e suas descrições:

- **processaLinhaRegistro**: seu objetivo é analisar uma linha de um arquivo de entrada e armazenar as informações em uma estrutura de artigo (Article).
- **main**: inicia a execução do programa.

2. hash.cpp(Marcos):

Esse código possui dois arquivo de cabeçalho (header.hpp e hash.hpp), descrito a seguir:

header.hpp(Marcos):

Defines:

- Tamanho do tipo alfa para o título (a): 300
- Tamanho do tipo alfa para os autores (b): 150
- Tamanho do tipo alfa para a atualização (c): 20
- Tamanho do tipo alfa para o snippet (d): 1024
- Número de buckets: 270973
- Número de registros: 10
- Tamanho do corpo do bloco: número de registros * (12 + a + b + c + d)
- Nome do arquivo hashing: arquivoDados

Includes:

- fstream

Estruturas:

- Article: id (inteiro não sinalizado), title (string), year (inteiro não sinalizado), author (string), citations (inteiro não sinalizado), update (string), snippet (string).
- Block: nRegisters (inteiro não sinalizado), body (string).

Cabeçalhos de funções:

- void imprimirRegistroArt (fstream *f)

hash.hpp(Marcos):

Includes:
- hearder.hpp

Cabeçalhos de funções:

- void **inicializaArquivoDeSaida** (fstream *f)
- bool **insereArquivoHash** (fstream *f, Article article)
- Article **buscaRegistroPorId** (fstream *f,int id)

- Article **buscaBucketPorTitulo** (fstream *f, int posicao, char title[TITLE_SIZE])

Focando agora no código em si, ele tem a responsabilidade de implementar as funções para construir o arquivo de dados, organizado por hashing. Aqui estão as variáveis globais e as descrições das funções:

Variáveis Globais:

- int collision

Funções e suas descrições:

- atualizaCabecalhoIndPrimario: Inicializa um arquivo de saída preenchendo-o com blocos vazios.
- hashing: Implementa um algoritmo de hashing simples (modular).
- consultaBucketPorId: Consulta e retorna o conteúdo de um bucket específico no arquivo de dados com base no ID fornecido.
- imprimirRegistroArt: Imprime os valores dos campos de um registro de um artigo (Article).
- insereArquivoHash: Insere um registro de um artigo (Article) em um arquivo de dados utilizando o método de hashing.
- buscaRegistroPorId: Busca um registro no arquivo de dados com base no ID fornecido.
- buscaBucketPorPosicao: Busca um bucket (que representa um bloco) no arquivo de dados com base na posição fornecida.
- buscaBucketPorTitulo: Busca um registro no arquivo de dados com base no título fornecido, realizando busca sequencial dentro de um bucket específico.

3. findrec.hpp(Alexandre)

O programa busca diretamente no arquivo de dados por um registro com o ID fornecido. Se encontrado, os campos do registro, a quantidade de blocos lidos para encontrá-lo e a quantidade total de blocos do arquivo de dados são retornados. Esse programa é compilado utilizando dois arquivos fonte: hash.cpp (previamente descrito) e findrec.cpp (desenvolvido por Gabriel). Além de criar a interface para a execução do programa, o findrec.cpp utiliza a função buscaRegistroPorId, implementada no arquivo hash.cpp, para realizar a busca.

O findrec.cpp tem duas dependências: a biblioteca iostream e o cabeçalho hash.hpp. Consiste apenas na função main, responsável por executar o programa.

Execução:

para execução do upload devemos colocar o seguinte comando no terminal:

- g++ hash/hash.cpp data/upload.cpp -o upload
  ./upload artigo.csv

para executar o findrec temos que colocar o seguinte comando no terminal:

- g++ findrec.cpp hash/hash.cpp -o findrec
  ./findrec <idRegistro>