UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ADRIEL DE SOUZA
CLARA SCHONS THEISEN

Aplicação de B-Tree na Implementação de um Protótipo de Agendamentos para Clínicas

Relatório apresentado como requisito parcial para a obtenção de conceito na Disciplina de Classificação e Pesquisa de Dados.

Orientador: Prof. Dr. Leandro Krug Wives

Porto Alegre
2024

SUMÁRIO

1 INTRODUÇÃO

Este relatório pretende descrever a motivação, o desenvolvimento e a implementação de um protótipo de sistema de agendamentos para uma clínica médica fictícia. O projeto foi desenvolvido como parte do trabalho final da disciplina de Classificação e Pesquisa de Dados da Universidade Federal do Rio Grande do Sul. O propósito principal deste projeto é aplicar os conceitos estudados na disciplina mediante uma aplicação que simula o gerenciamento de agendamentos e pacientes, utilizando um banco de dados desenvolvido especificamente para armazenar e manipular tais informações.

2 DESCRIÇÃO DA APLICAÇÃO

A aplicação busca simular um sistema gerenciador de agendamentos para uma clínica médica, sendo capaz de importar, padronizar, armazenar e consultar dados relacionados a agendamentos e pacientes. Para a importação foi utilizada o dataset *"Medical Appointment No Shows"* da plataforma KAGGLE que contém cerca de 110.000 entradas com atributos de agendamentos, sendo eles os seguintes: *PatientId, AppointmentID, Gender, ScheduledDay, AppointmentDay, Age, Neighbourhood, Scholarship, Hipertension, Diabetes, Alcoholism, Handcap, SMS_received, No-show*.

Durante o pré-processamento dos dados, um algoritmo desenvolvido pelos autores na linguagem Python foi utilizado para criar um dataset adicional de pacientes. Além dos atributos originais fornecidos pelo dataset da plataforma Kaggle, foram atribuídos nomes aleatórios a cada paciente para enriquecer o dataset e torná-lo mais representativo.

Para o armazenamento e consulta dos dados, foi utilizada uma combinação de Arquivos de Índices e Arquivos Seriais. Os Arquivos de Índices - uma representação em disco de Árvores B, as quais foram adaptadas de Comer (1979) -, são utilizados para otimizar a busca e a recuperação rápida de informações, enquanto os Arquivos Seriais permitem a persistência dos dados em formato binário, facilitando o acesso e a manipulação dos registros. Essa abordagem garante um equilíbrio entre desempenho e capacidade de armazenamento, permitindo que o sistema gerencie grandes volumes de dados de forma eficaz e ágil.

Está à disposição do usuário final 5 módulos para a manipulação destes dados: *importação, criação, listagem, consulta* e *edição* e por meio destes é possível realizar um processamento incremental de informações e manter a integridade dos dados, permitindo a atualização contínua e precisa do banco de dados conforme novas informações são adicionadas ou alteradas.

3 GUIA DE COMPILAÇÃO E EXECUÇÃO

Para compilar a aplicação, é necessário ter o GNU Compiler Collection (GCC) instalado na máquina que realizará o processo. Cada módulo da palicação pode ser compilado separadamente através do terminal de comando. A seguinte instrução deve ser executada, a partir do diretório raiz do projeot, para compilar um módulo específico é o seguinte:

```
gcc -std=c11 -I include -L lib -o <nome_do_arquivo> lib/* src/<nome_do_arquivo>.c
```

Onde<*nome_do_arquivo*> deve ser substituído pelo nome do módulo que você deseja compilar. Certifique-se de que a estrutura de diretórios está correta e todos os arquivos necessários estão presentes antes de executar o comando. Alternativamente pode-se usar o arquivo batch *compilar-todos.bat* para realizar a compilação de todos os módulos da aplicação.

A execução dos módulos deve ser feita através do terminal de comando. Para executar um módulo específico, utilize a seguinte instrução:

```
.\<nome_do_modulo> [argumentos]
```

Onde <*nome_do_modulo*> é o nome do módulo que você deseja executar e [argumentos] são os parâmetros apropriados para o módulo em questão.

4 GUIA DE USO

A aplicação é estruturada em cinco módulos, cada um responsável por uma função específica relacionada ao gerenciamento de dados de pacientes e agendamentos. Esses módulos foram projetados para operar de forma independente, permitindo flexibilidade no uso.

Módulo de Importação: Este módulo é utilizado para importar dados de arquivos CSV para o sistema, que são então indexados em árvores B para facilitar a busca e organização.

Exemplos de uso:

.*importar -reset -a Appointments.csv -p Patients.csv* Apaga o banco de dados atual e importa novos agendamentos e pacientes.

.*importar -p pacientesćlinica2.csv -a agendamentos_clinica2.csv* Importa incremental de pacientes e agendamentos

Módulo de Criação: Permite a criação de novos registros de pacientes ou agendamentos, após o uso da instrução inicial é solicitado ao usuário os dados da respectiva entidade.

Exemplos de uso:

.*novo -p* Cria um novo paciente

.*novo -a* Cria um novo agendamento

Módulo de Listagem (Serial e Paginada): Exibe uma lista paginada dos registros salvos no sistema, os itens são exibidos pela ordem de inserção no arquivo serial correspondente.

Exemplos de uso:

.*listar -p* Lista os pacientes

.*listar -a* Lista os agendamentos

Módulo de Consulta por ID: Realiza consultas específicas por ID de pacientes ou agendamentos, retornando informações detalhadas do registro solicitado, exibindo tanto as informações da entidade quanto das possíveis entidades relacionadas.

Exemplos de uso:

.*consulta -p 25* Exibe informações sobre o paciente de ID 25, se existir

.*consulta -a 1025* Exibe informações sobre o agendamento de ID 1025, se existir

Módulo de Edição por ID: Permite a edição de registros específicos, acessados por meio do ID, opera de forma semelhante ao módulo de criação - solicitando ao usuário os novos dados.

Exemplos de uso:

.*editar -p 25* Edita o paciente de ID 25, se existir

.*editar -a 1025* Edita o agendamento de ID 1025, se existir

5 ESTRUTURAS DE DADOS E ARQUIVOS

5.1 Árvores B

As chaves de cada nodo apresentam duas informações. A primeira é um inteiro key identificador para realizar as buscas e operações na árvore. A segunda é outro inteiro que representa o offset correspondente na estrutura de arquivo.

```
typedef struct {
   int key;
   int offset;
} keytype;
```

Cada nodo da árvore carrega as informações de número de chaves alocadas. Também são definidos um booleano indicando se o nodo é folha, um array de chaves, um array de filhos e um pai.

```
typedef struct nodo {
   int num;                // Representa o numero de chaves armazenadas no nodo
   bool flagfolha;         // Indicador de o nodo eh folha ou nao
   keytype* keys;          // Vetor de chaves
   struct nodo** filhos;   // Vetor de ponteiros para nodos filhos
   struct nodo* pai;       // Ponteiro do nodo pai
} nodoBTree;
```

A árvore em si tem um ponteiro para a raiz e um inteiro que informa sua ordem. Neste trabalho, estamos assumindo que a ordem é o valor mínimo de chaves que um nodo deve ter.

```
typedef struct {
   nodoBTree* raiz;        // Ponteiro para o nodo raiz da arvore
   int ordem;              // Representa a ordem da arvore (quantidade minima de chaves em um nodo)
} bTree;
```

Uma decisão importante foi tomada na implementação da árvore foi a alocação de um espaço extra no array de chaves para facilitar as operações de inserir, rotação e remoção. Tendo em vista as funcionalidades previstas para o sistema de agendamento de consultas e sua utilização em projetos que não demandam alto consumo de memória, foi considerado um trade-off positivo de definição na estrutura. Essa escolha de estrutura pode ser vista na criação de um nodo.

```
nodoBTree* criaNodo(int ordem, nodoBTree* pai){
      ...
   nodo->keys = (keytype*)malloc((ordem * 2 + 1) * sizeof(keytype));
   // note o tamanho do array de chaves passado para a funcao malloc
      ...
   nodo->filhos = (nodoBTree**)malloc((ordem * 2 + 2) * sizeof(nodoBTree*));
   // note o tamanho do array de filhos passado para a funcao malloc
      ...
   return nodo;
}
```

5.2 Representação da Árvore B em disco

A descrição anterior apresenta a estrutura da árvore B na memória principal, porém para atingirmos a persistência de dados proposta foi necessário implementar um algoritmo para ser possível o armazenamento desta na memória secundária. O algoritmo se baseia na conversão dos ponteiros de memória para *offsets* no arquivo onde a estrutura será salva. O código abaixo é uma versão simplificada

```
bool salvar_arvore(FILE *arquivo, bTree *arvore__) {
    fseek(arquivo, sizeof(bTree), SEEK_SET); // Pula o espa o dedicado para o descritor da arvore
    size_t offset_raiz = salvar_nodo(arquivo, arvore->raiz, arvore->ordem); // Salva o nodo raiz
    arvore->raiz = (void *)offset_raiz; // Atualiza o ponteiro com o offset no arquivo
    fseek(arquivo, 0, SEEK_SET); // Volta para o inicio do arquivo
    fwrite(arvore, sizeof(bTree), 1, arquivo) // Salva o descritor no arquivo
}

size_t salvar_nodo(FILE *arq, nodoBTree *n_, int ordem) {
    for (int i = 0; i < max_filhos; i++) { // Salva os filhos recursivamente
        n->filhos[i] = (void *)salvar_nodo(arq, n->filhos[i], ordem);
    }
    offset_nodo fwrite(n, sizeof(nodoBTree), 1, arq) // Salva o nodo
    fwrite(n->keys, sizeof(keytype), max_chaves, arq) // Salva as chaves
    fwrite(n->filhos, sizeof(nodoBTree *), max_filhos, arq) // Salva os offsets dos filhos
    return offset_nodo;
}
```

5.3 Entidades

As entidades utilizadas pela aplicação são abstraídas por meio de *structs* da Lingaugem C - linguagem na qual a aplicação principal foi desenvolvida. A escolha dessa representação se deu devido à facilidade para a conversão da estrutura para sua sequência binária correspondente, agilizando e simplificando o processo de salvamento e leitura das entidades do arquivo. O trecho de código abaixo apresenta as estruturas utilizadas pelo programa.

```
typedef struct {
    size_t id;
    size_t id_paciente; // Chave relacional
    size_t id_medico; // Chave relacional
    size_t id_relatorio; // Chave relacional
    Timestamp data_agendamento;  // Data e hora do agendamento
    Timestamp data_consulta; // Data e hora da consulta
    bool paciente_compareceu; // Se o paciente compareceu
} Agendamento;

typedef struct {
    size_t id;
    char nome[50]; // Nome do paciente
    char sobrenome[50]; // Sobrenome do paciente
    char bairro[50]; // Sobrenome do paciente
    char genero; // Genero do paciente (M ou F)
    Timestamp data_nascimento; // Data de nascimento do paciente
    bool hipertensao; // Se o paciente tem hipertensao
    bool diabetes; // Se o paciente tem diabetes
    bool alcoolismo; // Se o paciente e alcoolatra
} Paciente;
```

5.4 Arquivos

A aplicação utiliza dois tipos de arquivos, localizados no diretório data, para gerenciar e acessar os dados de forma eficiente:

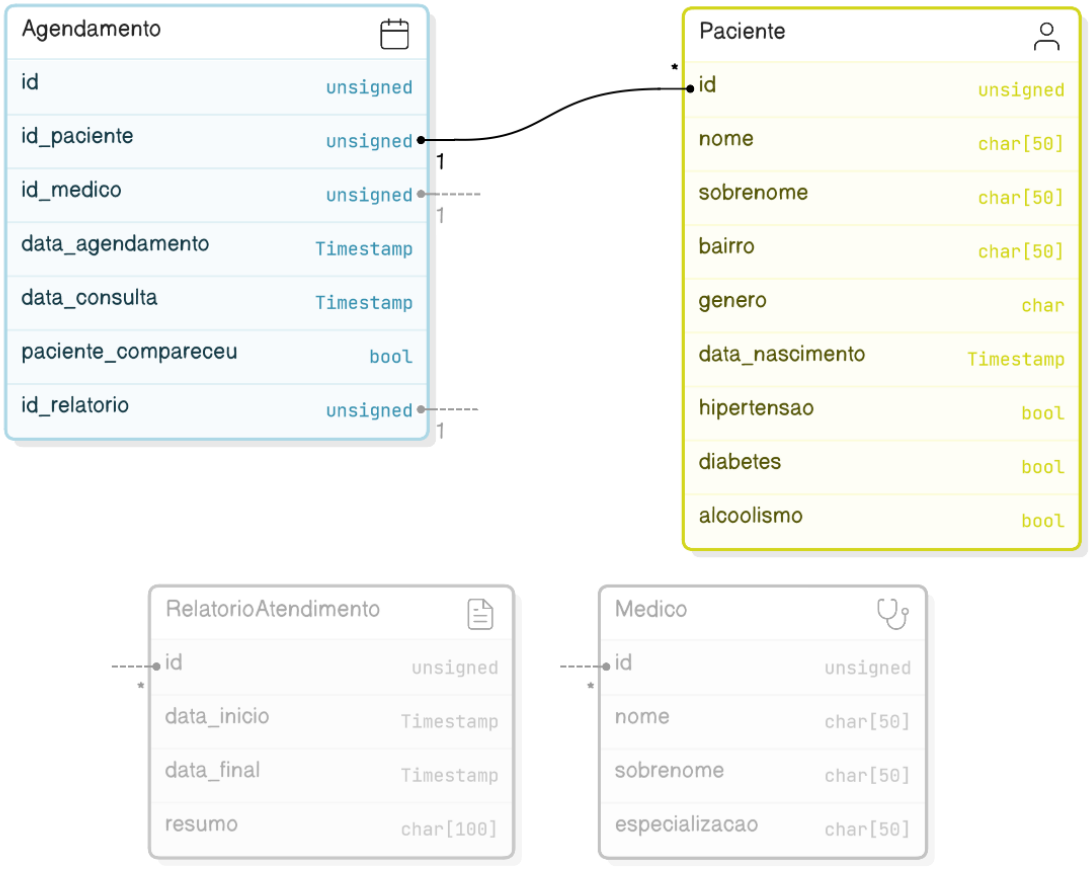
Arquivos seriais, identificados pela extensão .dat: armazenam os dados em formato binário. Estes contem os registros das entidades, salvos forma serial, O formato binário foi escolhido para garantir uma leitura e escrita rápidas e eficientes, de maneira que os registros possam ser interpretados por meio de uma conversão direta para a estrutura conhecida pelo programa.

Arquivos de índices, identificados pela extensão .idx: facilitam o acesso rápido e eficiente os registros dos arquivos seriais. Estes arquivos armazenam um descritor de uma Árvore B e seus respectivos nodos e através da reconstrução desta árvore é possível fazer consultas rápidas utilizando a chave identificadora do registro.

A combinação destes dois tipos de arquivos garantem que o sistema possa armazenar grandes volumes de dados de forma eficiente e recuperar as informações rapidamente quando necessário.

5.5 Diagramas

Figura 5.1 – Diagrama ER (Entity Relationship) da aplicação



Fonte: Elaborado pelos autores, 2024

As entidades *RelatorioAtendimento* e *Medico* não foram implementadas de maneira relacional, constam apenas para fins de demonstração.

6 CONSIDERAÇÕES FINAIS

Este projeto teve como objetivo principal a implementação de um protótipo - ou prova de conceito - de um sistema de gestão de agendamentos e pacientes de uma clínica médica fictícia, utilizando estruturas de dados e técnicas para persistência de dados. Através do desenvolvimento desta aplicação, tivemos a oportunidade de aplicar conceitos vistos na disciplina de Classificação de Pesquisa de Dados, em especial, Árvores B, Arquivos Seriais e Normalização de Tabelas Relacionais.

O trabalho apresentou desafios interessantes, que foram acentuados devido à escolha de uma linguagem de baixo nível. As implementações da estrutura de dados e algoritmos requereram conhecimentos adquiridos não somente por meio desta disciplina, mas também em disciplinas anteriores a ela.

Infelizmente, devido a limitações de tempo, não foi possível implementar as estruturas de Arquivos Invertidos e Árvores TRIE/PATRICIA, que permitiriam realizar buscas mais eficientes nos registros a partir de atributos secundários, como o nome do paciente ou data do agendamento. Estas funcionalidades teriam complementado o sistema oferecendo uma maior flexibilidade e agilidade na busca de registros específicos ou de consultas de forma mais genérica.

Considerando outras possíveis melhorias, uma futura adição de uma interface gráfica ao projeto facilitaria a interação dos usuários com o sistema, dado que o uso da aplicação através da linha de comando pode ser menos intuitivo e requerer um conhecimento prévio do funcionamento de uso de um terminal de comandos.

REFERÊNCIAS

COMER, D. Ubiquitous b-tree. **ACM Computing Surveys**, v. 11, p. 121–137, 06 1979.

KAGGLE. **Medical Appointment No Shows**. 2017. Disponível em: <https://www.kaggle.com/datasets/joniarroba/noshowappointments>.