Trabalho Prático - Fase 2

Gerenciamento de Estoque com Índices em Arquivos

Integrantes:

- Arthur Braga de Campos Tinoco
- Rafael Lima Mendonça Garcia

PUC Minas - Unidade Praça da Liberdade

AEDS 3

---

1. Documentação do Projeto (Fase 1 - Refinada)

1.1. Descrição do Problema

O sistema proposto visa permitir o cadastro e gerenciamento de Itens em Estoque, Fornecedores e Categorias. A solução deve possibilitar o acompanhamento da quantidade de itens comprados e vendidos, com todos os dados sendo armazenados em arquivos binários que incluem um cabeçalho para controle de metadados e um sistema de exclusão lógica por lápide.

1.2. Objetivo do Trabalho

- Desenvolver um sistema que permita as operações de CRUD (Create, Read, Update, Delete) para as entidades de estoque, fornecedores e categorias.
- Garantir a persistência dos dados em arquivos binários com controle de exclusão lógica.
- Fornecer documentação técnica completa, contendo Diagrama de Caso de Uso (DCU), Diagrama Entidade-Relacionamento (DER) e a Arquitetura Proposta.

1.3. Requisitos Funcionais

- **RF01:** Gerenciar Categorias (CRUD completo).
- **RF02:** Gerenciar Fornecedores (CRUD completo).
- **RF03:** Gerenciar Itens em Estoque (CRUD completo).

- **RF04:** Registrar Movimentações de Estoque (entradas e saídas) - *escopo para fases futuras*.

1.4. Requisitos Não Funcionais

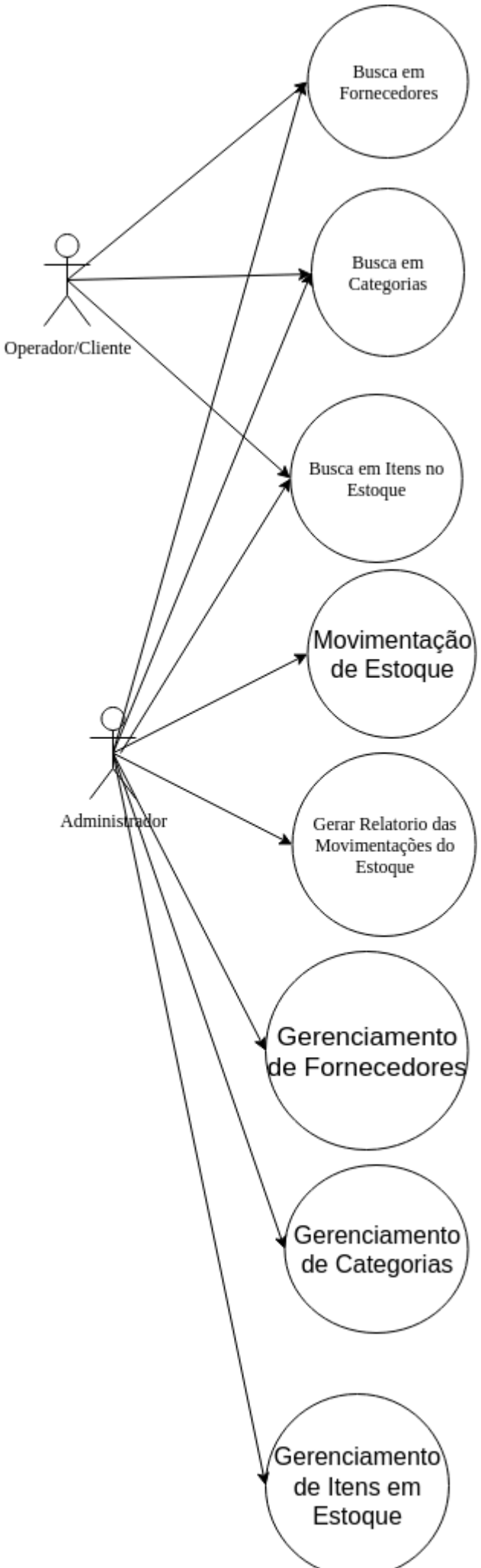
- **RNF01:** A persistência de dados deve ser realizada diretamente em arquivos binários, sem o uso de um SGBD.
- **RNF02:** Os arquivos de dados devem possuir um cabeçalho para armazenamento de metadados (ex: último ID utilizado).
- **RNF03:** A exclusão de registros deve ser implementada de forma lógica (lápide), sem remoção física dos dados.
- **RNF04:** O sistema deve seguir a arquitetura MVC + DAO.
- **RNF05:** A interface com o usuário para esta fase do projeto será via console.

1.5. Atores do Sistema

- **Administrador:** Responsável por cadastrar, editar, excluir e consultar categorias, fornecedores e itens em estoque.
- **Cliente/Operador:** Pode apenas consultar os itens disponíveis no sistema.

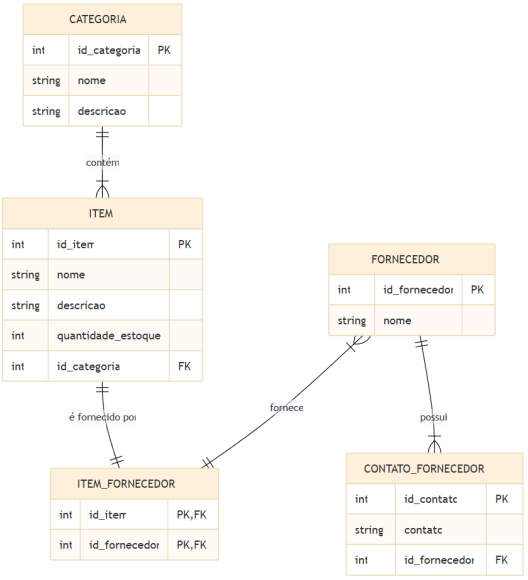
1.6. Diagrama de Caso de Uso

O diagrama a seguir ilustra as funcionalidades do sistema e as interações dos atores. Os casos de uso foram nomeados com verbos no infinitivo, conforme as boas práticas de modelagem.


Busca em Fornecedores
Busca em Categorias
Busca em Itens no Estoque
Movimentação de Estoque
Gerar Relatorio das Movimentações do Estoque
Gerenciamento de Fornecedores
Gerenciamento de Categorias
Gerenciamento de Itens em Estoque
Operador/Cliente
Administrador

1.7. Diagrama Entidade-Relacionamento (DER)

O DER conceitual abaixo modela as entidades principais do sistema e seus relacionamentos.



1.8. Arquitetura Proposta

O sistema foi estruturado seguindo o padrão arquitetural **MVC + DAO**:

- **Model:** Contém as classes de domínio (`Categoria`, `Fornecedor`, `ItemEstoque`).
- **View:** Contém a classe `MainView`, responsável pela interface com o usuário via console.
- **DAO (Data Access Object):** Camada de acesso aos dados, responsável pela manipulação dos arquivos binários e pela interação com as estruturas de índice.

[PRÓXIMA PÁGINA]

2. Formulário de Projeto – Decisões Técnicas (Fase 2)

a) Qual a estrutura usada para representar os registros?

A persistência dos dados foi implementada utilizando arquivos binários de acesso aleatório (`RandomAccessFile`). Cada arquivo de entidade (ex: `categorias.db`) possui um **cabeçalho** de 4 bytes que armazena o último ID utilizado, garantindo o controle sequencial dos identificadores. Cada registro no arquivo segue a estrutura: `[ Lápide (1 byte) ] [ Tamanho do Registro (int - 4 bytes) ] [ Dados Serializados (N bytes) ]`. A lápide (' ') indica um registro ativo e ('*') um registro logicamente excluído. O tamanho do registro é gravado antes dos dados para permitir pular registros de tamanho variável de forma eficiente.

b) Como atributos multivalorados do tipo string foram tratados?

O atributo multivalorado de telefones na entidade Fornecedor foi tratado convertendo a lista de Strings (List<String>) em uma única String concatenada, utilizando um caractere delimitador (;). No método toByteArray, String.join(";") é usado para criar a string única antes da gravação. No método fromByteArray, o método split(";") é utilizado para reconstruir a ArrayList de telefones a partir da string lida do arquivo.

c) Como foi implementada a exclusão lógica?

A exclusão lógica foi implementada através de uma "lápide", que é o primeiro byte de cada registro no arquivo de dados. Quando um registro é criado, este byte é gravado com o valor de espaço (' '). Quando o método delete de um DAO é chamado, ele localiza o registro (usando o índice primário) e sobrescreve apenas o byte da lápide com um asterisco ('*'). As operações de leitura são programadas para ignorar qualquer registro que comece com o caractere de lápide. Além disso, a chave do registro excluído é removida do índice primário (Hash Extensível) para que não seja mais encontrada em buscas diretas.

d) Além das PKs, quais outras chaves foram utilizadas nesta etapa?

Além das chaves primárias (PKs) de todas as tabelas, foi utilizada a chave estrangeira (FK) idCategoria na tabela ItemEstoque. Esta chave estabelece o relacionamento 1:N, onde uma Categoria pode ter N Itens de Estoque.

e) Quais tipos de estruturas foram utilizadas para cada chave de pesquisa?

- **Para todas as Chaves Primárias (PKs):** Foi implementado um índice de **Hash Extensível**. Esta estrutura foi escolhida por sua alta eficiência em buscas diretas por chave (complexidade O(1) em média), ideal para operações de read, update e delete baseadas em um ID específico.
- **Para a Chave Estrangeira idCategoria:** Foi implementado um índice secundário utilizando uma **Árvore B+**. Esta estrutura foi escolhida por ser extremamente eficiente em buscas por faixa e por agrupar chaves iguais, permitindo recuperar rapidamente todos os registros ('N') associados a uma chave específica ('1'), o que é a exata definição da busca no relacionamento 1:N.

f) Como foi implementado o relacionamento 1:N?

O relacionamento 1:N entre Categoria e ItemEstoque foi implementado através de um índice secundário de **Árvore B+** sobre a chave estrangeira idCategoria na tabela de ItemEstoque. A navegação funciona da seguinte forma: para listar todos os itens de uma categoria, o sistema consulta a Árvore B+ com o idCategoria desejado. A árvore retorna uma lista de todos os endereços de disco (ponteiros) para os registros de ItemEstoque que possuem aquele idCategoria. O DAO então percorre essa lista de endereços, acessando diretamente cada registro no arquivo de dados (itens_estoque.db) sem a necessidade de uma varredura sequencial. A **integridade referencial** é mantida no nível da aplicação: antes de criar um ItemEstoque, a MainView utiliza o método read do CategoriaDAO e FornecedorDAO para verificar se os IDs da categoria e do fornecedor informados são válidos.

g) Como os índices são persistidos em disco?

Cada estrutura de índice gerencia seus próprios arquivos binários.

- O **Hash Extensível** utiliza dois arquivos: um para o diretório (_dir.db), que armazena a profundidade global e a lista de ponteiros para os cestos; e outro para os cestos (_cestos.db), que armazena os próprios cestos (profundidade local e os pares chave/endereço).
- A **Árvore B+** utiliza um único arquivo (_bplus.db) que contém um cabeçalho com o ponteiro para a página raiz, seguido pelas páginas (nós) da árvore serializadas.

A **sincronização** é imediata: a cada operação de create, update ou delete no DAO, a estrutura do índice em memória é modificada e, em seguida, as alterações são gravadas diretamente nos arquivos de índice correspondentes no disco. Isso garante que os índices e os dados nunca fiquem dessincronizados.

h) Como está estruturado o projeto no GitHub?

O projeto está estruturado seguindo o padrão arquitetural **MVC + DAO**, organizado em pacotes Java distintos dentro da pasta src/:

- /src/model: Contém as classes de domínio (Categoria, Fornecedor, ItemEstoque).
- /src/dao: Contém as classes de acesso a dados, responsáveis pela manipulação dos arquivos binários e pela interação com os índices.
- /src/view: Contém a classe de interface com o usuário via console (MainView).
- /src/app: Contém a classe Main, que é o ponto de entrada da aplicação.
- /src/indices: Contém as implementações das estruturas de dados de indexação (HashExtensível e Árvore B+).
- /data: Diretório na raiz do projeto onde todos os arquivos de dados (.db) e de índices são criados e mantidos.

---

3. Link para o Código-Fonte

O código-fonte completo do projeto, incluindo as instruções de compilação e execução no arquivo README.md, está disponível no seguinte repositório GitHub:

Braga451/AEDSIII-Trabalho at TP2
