Sistema Gerenciador de Estoque

O Sistema é uma aplicação desenvolvida em Java junto a uma interface e conexão a um banco de dados MySQL, consiste em ser um sistema de gerenciamento de estoque para armazenar informações, como: as movimentações do estoque, adição de produtos, categorias e consultas dos mesmos.

• Estrutura do Projeto

O projeto consiste em ser detalhado e organizado por funções, sendo cada um com suas responsabilidades.

Sistema de Estoque: O sistema implementa um **gerenciador de estoque** em Java, integrando o banco de dados MySQL usando **JDBC** Consiste em gerenciar estoques com interação em tempo real no terminal.

Configuração de Conexão JDBC

- **DriverManager**:
  - Garante a conexão com o banco MySQL, utilizando a URL, usuário e senha.
  - O método `getConnection` retorna um objeto `Connection`, encapsulando a sessão com o banco.
- **Configuração de Banco**:
  - Banco: `gerenciadordeestoque`.
  - Métodos de integração com **Procedures MySQL** (via `CallableStatement`) otimizam segurança e reutilização do banco.

Design Modular

Cada funcionalidade corresponde a um método estático, facilitando a manutenção e expansão. O design é estruturado de maneira procedural, focando no fluxo linear.

- **Menu Dinâmico (`exibirMenu`)**:
  - Exibe opções e recebe entrada do usuário via `Scanner`.
  - Evita interrupções do programa com controle de exceções.
- **Execução Baseada em Opções**:
  - Um `switch-case` organiza as chamadas para métodos de cadastro, consulta ou relatórios.

Uso de Procedures no MySQL

- Procedimentos armazenados eliminam lógica SQL redundante no código.

- Métodos usam `CallableStatement` para invocar **procedures** com parâmetro

Detalhamento das Funções

- **Cadastro**:
  - `cadastrarCategoria` e `cadastrarProduto`.
  - Aceitam entrada do usuário para criar categorias/produtos, com parâmetros validados antes de invocar procedures como `CadastroCategoria`.
- **Consulta**:
  - `consultarProdutos` e `consultarCategorias`.
  - Executam consultas no banco, filtrando por parâmetros opcionais (`null` em parâmetros ignorados).
  - Processam resultados com `ResultSet`.
- **Relatórios**:
  - `relatorioProdutosCadastrados`, `relatorioMovimentacoes`.
  - Exibem dados em formato tabular para fácil análise, iterando sobre os resultados.

Tratamento de Exceções

- **SQL Exceptions**:
  - O `try-with-resources` assegura o fechamento de conexões e statements.
  - Exceções são tratadas com `printStackTrace` para depuração.

Gerenciador de Estoque: É um código SQL que implementa o esquema de um sistema de gerenciamento de estoque com **tabelas, procedures armazenadas e triggers**

Criação do Banco de Dados

```
CREATE DATABASE GerenciadorDeEstoque;
USE GerenciadorDeEstoque;
```

Define o banco de dados chamado `GerenciadorDeEstoque` e passa a utilizá-lo para as operações subsequentes.

Tabela Categorias

```
CREATE TABLE Categorias (
    IDCategoria INT PRIMARY KEY AUTO_INCREMENT,
    NomeCategoria VARCHAR(60) NOT NULL,
    DescricaoCategoria TEXT NOT NULL
);
```

- **Colunas:**
    - `IDCategoria`: Chave primária, com incremento automático.
    - `NomeCategoria`: Nome da categoria, obrigatório.
    - `DescricaoCategoria`: Texto descrevendo a categoria.

Tabela Produtos

```
CREATE TABLE Produtos (
    IDProduto INT PRIMARY KEY AUTO_INCREMENT,
    NomeProduto VARCHAR(60) NOT NULL,
    DescricaoProduto TEXT,
    QuantidadeEstoque INT DEFAULT 0,
    PrecoCompra FLOAT NOT NULL,
    PrecoVenda FLOAT NOT NULL,
    CategoriaProduto INT,
    CONSTRAINT fk_categoria FOREIGN KEY (CategoriaProduto)
REFERENCES Categorias(IDCategoria)
);
```

- **Relacionamento:** `CategoriaProduto` é uma **chave estrangeira** que faz referência à `IDCategoria` de `Categorias`.

Tabela `MovimentacoesEstoque`

```
CREATE TABLE MovimentacoesEstoque (
    IDMovimentacao INT PRIMARY KEY AUTO_INCREMENT,
    IDProduto INT NOT NULL,
    TipoMovimentacao ENUM('Entrada', 'Saída') NOT NULL,
    Quantidade INT NOT NULL,
    DataMovimentacao TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (IDProduto) REFERENCES Produtos(IDProduto)
);
```

- Registra movimentações de estoque para produtos:
    - `TipoMovimentacao`: Entrada ou saída.

- `Quantidade`: Quantidade movimentada.
- `DataMovimentacao`: Registrada automaticamente.

2. Procedures Armazenadas

Cadastro de Categoria e Produto

Procedures como `CadastroCategoria` e `CadastroProduto` permitem inserir novos registros nas tabelas correspondentes.

```
CREATE PROCEDURE CadastroProduto(
    IN NomeProduto VARCHAR(60),
    IN DescricaoProduto TEXT,
    IN QuantidadeEstoque INT,
    IN PrecoCompra FLOAT,
    IN PrecoVenda FLOAT,
    IN CategoriaProduto INT
)
BEGIN
    INSERT INTO Produtos
    (NomeProduto, DescricaoProduto, QuantidadeEstoque, PrecoCompra,
PrecoVenda, CategoriaProduto)
    VALUES
    (NomeProduto, DescricaoProduto, QuantidadeEstoque, PrecoCompra,
PrecoVenda, CategoriaProduto);
END //
```

- Insere um novo produto com os detalhes fornecidos como parâmetros.

Editar e Deletar

Procedures como `EditarProduto` e `DeletarProduto` permitem atualizar ou excluir registros.

```
UPDATE Produtos
SET
    NomeProduto = COALESCE(NovoNomeProduto, NomeProduto),
    QuantidadeEstoque = COALESCE(NovaQuantidadeEstoque,
QuantidadeEstoque)
WHERE IDProduto = IDProduto;
```

- **Uso de COALESCE:** Garante que, caso um parâmetro seja NULL, o valor atual não seja alterado.

Movimentação de Estoque

```
CREATE PROCEDURE RegistrarMovimentacaoEstoque(
    IN ProdutoID INT,
    IN TipoMovimentacao ENUM('Entrada', 'Saída'),
    IN Quantidade INT
)
BEGIN
    IF TipoMovimentacao = 'Entrada' THEN
        UPDATE Produtos
        SET QuantidadeEstoque = QuantidadeEstoque + Quantidade
        WHERE IDProduto = ProdutoID;
    ELSEIF TipoMovimentacao = 'Saída' THEN
        IF (SELECT QuantidadeEstoque FROM Produtos WHERE IDProduto =
ProdutoID) < Quantidade THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Estoque insuficiente para a saída!';
        ELSE
            UPDATE Produtos
            SET QuantidadeEstoque = QuantidadeEstoque - Quantidade
            WHERE IDProduto = ProdutoID;
        END IF;
    END IF;
END //
```

- Garante que saídas de estoque não excedam a quantidade disponível.
- Usa SIGNAL para lançar erros.

3. Triggers

Verificação de Estoque Baixo

```
CREATE TRIGGER VerificarEstoqueBaixo
AFTER UPDATE ON Produtos
FOR EACH ROW
BEGIN
    IF NEW.QuantidadeEstoque < 5 THEN
        SIGNAL SQLSTATE '45000'
```

```
        SET MESSAGE_TEXT = 'Estoque abaixo do nível mínimo!';
    END IF;
END //
```

- Verifica se a quantidade de um produto cai abaixo de 5 após atualização e lança um erro.

4. Relatórios

As procedures de relatório agregam informações úteis. **Exemplo:**

Produtos com Estoque Baixo

```
CREATE PROCEDURE RelatorioProdutosBaixoEstoque(IN EstoqueMinimo INT)
BEGIN
    SELECT
        p.IDProduto,
        p.NomeProduto,
        p.QuantidadeEstoque,
        c.NomeCategoria
    FROM Produtos p
    LEFT JOIN Categorias c ON p.CategoriaProduto = c.IDCategoria
    WHERE p.QuantidadeEstoque < EstoqueMinimo;
END //
```

- Lista produtos com estoque abaixo de um valor mínimo.

Relatório de Vendas e Lucro

```
CREATE PROCEDURE RelatorioVendasLucro()
BEGIN
    SELECT
        p.NomeProduto,
        SUM(m.Quantidade * p.PrecoVenda) AS TotalVendas,
        SUM(m.Quantidade * (p.PrecoVenda - p.PrecoCompra)) AS Lucro
    FROM MovimentacoesEstoque m
    JOIN Produtos p ON m.IDProduto = p.IDProduto
    WHERE m.TipoMovimentacao = 'Saída'
    GROUP BY p.IDProduto;
END //
```

- Calcula o total de vendas e lucro baseado nas movimentações de saída.

Dependências

```
<orderEntry type="inheritedJdk" />
<orderEntry type="sourceFolder" forTests="false" />
<orderEntry type="library" name="mysql-connector-j-9.1"
level="project" />
```

- **`type="inheritedJdk"`:** Especifica que o módulo usará o JDK herdado do projeto global.
- **`type="sourceFolder"`:** Configura pastas de código-fonte para o módulo.
- **`type="library"`:**
    - Adiciona a biblioteca `mysql-connector-j-9.1` como dependência.
    - **`level="project"`:** A biblioteca está configurada no nível do projeto, compartilhada por todos os módulos.