Sistema de Cadastro de Automóveis AutoPrime

Guilherme de Souza Teixeira

Ciência da Computação - Universidade Estácio de Sá (UNESA)CEP 24020340 - Niterói - RJ – Brasil

Trabalho Final - Programação Orientada a Objetos em Java

202202868991@alunos.estacio.br

Abstract. This article describes the operation of a Java language project that makes use of a connection to a database in PosgreSQL in order to help the inventory control in a fictitious candy store named "Doceria Beijo de Mel".

Resumo. Este artigo descreve o funcionamento de um projeto em linguagem Java que faz uso de uma conexão com um banco de dados no PostgreSQL visando auxiliar o controle de estoque em uma loja de doces fitícia de nome "Doceria Beijo de Mel".

1. Nome do sistema

Banco de dados Doceria Beijo de Mel.

2. Objetivo do sistema

O objetivo do sitema é auxiliar os funcionários da loja "Beijo de Mel" a manter controle sobre o estoque.

3. Requisitos do sistema / o que o sistema irá fazer

Esse sistema em Java requer uma conexão com um banco de dados no programa PosgreSQL. Ele possui um menu principal, com 8 opções selecionáveis. Nele é possível realizar todos as operações CRUD, além de algumas funções mais específicas, como apagar a lista por completo e separar apenas os doces com menos de 3,5kg disponível em estoque.

4. Caso de uso do sistema

Esse sistema poderia ser usado, exemplificando, por um trabalhador da loja de doces que avalia diariamente o estoque, ajudando o mesmo a saber os produtos que estão próximos de acabar, para serem, então, encomendados com antecedência.

5. Explicação do que foi implementado

5.1. BeijoDeMel.java

Inicialmente, é criada a classe principal do programa, a "BeijoDeMel". Em seguida ocorre a tentativa de conexão com o driver do PostgreSQL, com uma mensagem de erro em caso de falha. Cria-se, com o comando *System.out.println* a tela principal do programa, com as 8 opções, juntamente com um comando *while*, para repetir a impressão do menu principal até a decisão do usuário de fechar o sistema. Os comandos *switch* e *case* irão iniciar diferentes funções de acordo com o número selecionado pelo usuário.

5.2. CriarTabela.java

Antes da definição da classe "CriarTabela", são aplicadas as 4 bibliotecas "java.SQL.". "Connection", "DriverManager", "SQLException" e "Statement".
É criada a *String* "SQLcriarTabela", que construirá os tipos de dados e os nomes de cada variável de dados nos códigos. São criados os dados "nome", que se refere ao nome e sabor do doce, "tipo," que se refere ao grupo que esse doce se encontra (jujuba, chiclete, bala...), "precokg", e "qtdkg" que se referem ao preço e quantidade disponível por/em quilograma de doce, respectivamente. É feita a tentativa de conexão, e após isso a tabela é criada fazendo o uso da *String* "SQLcriarTabela". Por último, a conexão com o driver do PostgreSQL é fechada.

```
System.out.println(x: "Criando dados da lista...");
st = cn.createStatement();
st.executeUpdate(sql:SQLcriarTabela);
System.out.println(x: "A lista foi criada com sucesso!");
st.close();
cn.close();
```

Lista "doces" sendo criada na Database.

5.3. InserirDados.java

Essa classe será a responsável por inserir os novos dados desejados pelo usuário na tabela "doces". A *String* "SQLinserirDados" será usada para adicionar na lista os 4 valores definidos por linha (nome, tipo, precokg e qtdkg), com os valores sendo decididos pelo usuário. Pra isso, após o teste de conexão de driver, é pedido para o usuário informar os dados sobre a nova linha da lista, um por um, com *scanner.nextLine* para os dados em *String* e *scanner.nextFloat* para os dados em *float*. Eles logo serão atribuídos a novas variáveis, das quais serão inseridas na tabela por meio da variável *PreparedStatement* "pstmt".

```
System.out.print(s: "\nInforme os dados do doce: \n");
Scanner scanner = new Scanner(source: System.in);
System.out.print(s: "Nome: ");
String nome = scanner.nextLine();
System.out.print(s: "Tipo: ");
String tipo = scanner.nextLine();
System.out.print(s: "PrecoKg: ");
float precokg = scanner.nextFloat();
System.out.print(s: "QuantidadeKg: ");
float qtdkg = scanner.nextFloat();
scanner.nextLine();

PreparedStatement pstmt = cn.prepareStatement(sql:SQLinserirDados);
pstmt.setString(parameterIndex: 1, x: nome);
pstmt.setString(parameterIndex: 2, x: tipo);
pstmt.setFloat(parameterIndex: 3, x: precokg);
pstmt.setFloat(parameterIndex: 4, x: qtdkg);
```

Inserção de nova linha na lista "doces".

5.4. RemoverDados.java

Nesta classe, a String "SQLapagarDados" removerá uma ou mais linhas da lista de acordo com o nome do item que será inserido pelo usuário. Para isso, essa String recebe o valor de "*DELETE FROM doces WHERE nome = ?*", e é criada a variável "nomeDoceRemover". Por último, após a inserção do nome, ocorre uma verificação com a variável "linhasAfetadas" para garantir de que ao menos 1 linha foi apagada. Caso contrário, uma mensagem será mostrada na tela, informando que o nome fornecido deve ser verificado novamente.

```
System.out.print(s: "\nInforme o nome do doce a ser removido: ");
String nomeDoceRemover = new Scanner(source: System.in).nextLine();

PreparedStatement pstmt = cn.prepareStatement(sql:SQLapagarDados);
pstmt.setString(parameterIndex: 1, x: nomeDoceRemover);

System.out.println(x: "\nRemovendo o item da tabela...");
    int linhasAfetadas = pstmt.executeUpdate();
    if (linhasAfetadas > 0){
        System.out.println(x: "\nItem removido com sucesso!");
    }else{
        System.out.println(x: "\nNenhum item removido. Verifique o nome fornecido.");
    }
```

Remoção do item da tabela com base no nome do(s) item(ns).

5.5. ModificarDados.java

Essa classe é uma união de duas classes anteriores: a "RemoverDados" e a "InserirDados", uma seguida da outra nessa ordem. Com a remoção de um dado

seguida pela inserção de um novo, a classe desempanha o mesmo que uma modificação de um item específico.

5.6. LimparLista.java

Essa classe é bem simples, apenas fazendo a conexão com o driver e usando a *String* "SQLapagarDados" com o valor "DELETE from doces". Com o *statement* "st", todos os dados da lista são apagados.

```
System.out.println(x: "\nApagando dados...");
st = cn.createStatement();
st.executeUpdate(sql:SQLapagarDados);
System.out.println(x: "\nDados Apagados com sucesso!");
st.close();
cn.close();
```

Apagando todos os dados da lista "doces".

5.7. ListarDoces.java

Nessa classe, serão listados todos os atuais itens listados em "doces". Para isso, serão usados comandos simples. Com a *String* "SQLlerDados" possuindo o valor de "*SELECT * FROM doces*", é utilizado o comando *while(result.next())* para passar por todos os dados na lista.

```
public static void main(String[] args){
    String SQLlerDados = "SELECT * FROM doces";
    String driver = "jdbc:postgresql://localhost/DadosBeijoDeMel";
    Statement st = null;
    ResultSet result = null;
```

Definição da String "SQLlerDados"

Conforme o while(result.next()) passa pelos dados, os mesmos são formatados com 'System.out.println", "getString" e "getFloat" para a visualização do usuário.

```
System.out.println(x: "\nListando dados da tabela...\n");
st = cn.createStatement();
result = st.executeQuery(sql:SQLlerDados);

while(result.next()){
    System.out.println(x: "-----------------------------------");
    System.out.println("Nome: " + result.getString(columnIndex:1));
    System.out.println("Tipo: " + result.getString(columnIndex:2));
    System.out.println("PrecoKg: " + result.getFloat(columnIndex:3));
    System.out.println("QuantidadeKg: " + result.getFloat(columnIndex:4));
    System.out.println();
}
```

Código percorre por todos os dados na tabela, imprimindo-os.

5.8. ListarDocesAcabando.java

Para verificar na lista "doces" apenas os itens em estoque que acabaram ou estão perto de acabar, usa-se essa classe. É o mesmo que a anterior ("ListarDoces"), porém com um condicional na *String* "SQL" inicial de que apenas serão impressos os itens com a quantidade em kg disponível abaixo de 3,5. Caso contrário, o item não aparecerá na lista. No fim, ficamos com este resultado:

```
public static void main(String[] args) {
    String SQLconsultarDados = "SELECT * FROM doces WHERE qtdkg < 3.5";
```

Definição da String "SQLconsultarDados", que apenas encontra dados com quantidade em kg disponível abaixo de 3,5.

6. Conclusão

Em suma, esse sistema funciona como o intencionado e proposto pelo trabalho. Faz uma conexão sucedida com o banco de dados no PostgreSQL realiza as funções base do CRUD e um pouco além disso. Todavia, por conta de se tratar apenas de texto, e não de um aplicativo com otimização gráfica da tabela, ela acaba se tornando datada.

Para finalizar, há sim, um uso para ela, e para o uso básico de CRUD, pode ser razoavelmente eficiente.