AT04 - Processado comando SQL em JDBC

Layza Nauane – BD2 – 12211BSI251

1.1. Arquivo MyQueries.java: Função getMyData() Modificada

A função *getMyData* foi modificada com o objetivo de listar o nome dos fornecedores de café e a quantidade de tipos de café vendidos para a loja. Para isso precisei integrar dados de duas tabelas do banco de dados, chamadas *COFFEES* e *SUPPLIERS*. A função executa uma consulta SQL que usa junções e agregação para buscar essas informações.

1.2. Lógica

A função recebe um objeto de conexão JDBC, que é o que nos permite falar com o banco de dados.

a. Primeiro um objeto Statement é inicializado com null, após isso ajustei a consulta SQL para atender ao exercício, que é listar os fornecedores e a quantidade de café vendidos.

```
public static void getMyData(Connection con) throws SQLException {
  Statement stmt = null;
  String query =
    "SELECT S.SUP_NAME, COUNT(C.COF_NAME) AS totalSalesOfCoffees " +
    "FROM COFFEES C JOIN SUPPLIERS S ON C.SUP_ID = S.SUP_ID " +
    "GROUP BY S.SUP_NAME";
```

Alguns detalhes da consulta:

SELECT S.SUP_NAME, COUNT(C.COF_NAME) AS totalSalesOfCoffees:

- Utilizei o *SELECT* para selecionar as informações que eu quero, que no caso é o nome do fornecedor *(SUP_NAME)* e a quantidade de vendas de tipos de café para a loja *(COUNT(C.COF_NAME))*.

FROM COFFEES C JOIN SUPPLIERS S ON C.SUP_ID = S.SUP_ID:

- Fiz a junção das tabelas *SUPPLIES* e *COFFEE* e verifiquei se o campo *SUP_ID* da tabela SUPPLIERS é igual ao campo *SUP_ID* da tabela *COFFEE*

GROUP BY S.SUP_NAME:

- Agrupei os resultados pelo nome do fornecedor, para conseguir contar os cafés vendidos de cada um.

b. Depois de montar a consulta, usei um `Statement` do JDBC para executar e obter os resultados.

c. No loop, cada linha do *ResultSet* é processada e para cada linha pego o nome do fornecedor e a quantidade de cafés vendidos. Depois faço a exibição dos dados no console, onde exibo o nome do fornecedor e a quantidade de cafés que ele vendeu.

d. Envolvendo o código acima, temos um bloco *try-catch* para pegar possíveis erros de SQL que podem rolar na execução da consulta. Se algo der errado, uso uma função para imprimir o erro.

```
    try {
      stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery(query);
      System.out.println("\nLista de Fornecedores e a quantidade de tipos de cafés vendidos:\n");
      while (rs.next()) {
        String supplierName = rs.getString("SUP_NAME");
        int totalSalesOfCoffees = rs.getInt("totalSalesOfCoffees");
        System.out.println("Fornecedor: " + supplierName + "\nTotal de tipos de café vendidos: " + totalSalesOfCoffees + " \n");
      }
    } catch (SQLException e) {
      JDBCUtilities.printSQLException(e);
    } finally {
      if (stmt != null) { stmt.close(); }
    }
  }
```

2. Script comp

O scrip comp serve para automatizar a compilação e execução de programas java que interage com o banco de dados Derby.

Foi necessário apenas modificar uma linha no script, que seria o caminho especificado em *mypath,* definindo o caminho correto em que o JDBCTutorial está em meu computador.

```
derby=/usr/share/java/derby.jar
mypath=/home/user/Documentos/JDBCTutorial
mypackage=com/oracle/tutorial/jdbc
```

3. Arquivo build.xml

Foi necessário adicionar uma sessão de configuração chamada *mycode* para executar a classe MyQueries no build.xml.

```
<target name="runmyqueries">
  <java classname="com.oracle.tutorial.jdbc.MyQueries" classpathref="CLASSPATH" fork="true" dir=".">
    <arg value="${PROPERTIESFILE}"/>
  </java>
</target>
<target name="mycode">
  <java classname="com.oracle.tutorial.jdbc.MyQueries" classpathref="CLASSPATH" fork="true" dir=".">
    <arg value="${PROPERTIESFILE}"/>
  </java>
</target>
<target name="zip" description="Package source in zip file">
  <delete file="./JDBCTutorial.zip"/>
  <zip destfile="./JDBCTutorial.zip">
    <zipfileset dir="sql" prefix="sql"/>
    <zipfileset dir="src" prefix="src"/>
    <zipfileset dir="properties" prefix="properties"/>
    <zipfileset dir="." includes="build.xml"/>
  </zip>
```

4. Arquivo JDBCUtilities.java

Nesse arquivo não foi necessário fazer nenhuma modificação, mas ele é importante para trabalhar com JDBC, permitindo conectar-se a diferentes bancos de dados e executar consultas SQL. Ele inclui várias funcionalidades, como captura de exceções SQL, manipulação de conexões, e gerenciamento de propriedades de conexão.

5. Execução