

Banco de Dados

Aula 19 - Utilizando SQL em Java

Apresentação

Depois de aprender como modelar, criar e manipular dados utilizando a linguagem SQL, você verá como utilizar uma linguagem de programação como Java para usar linguagem SQL, ou seja, você irá aprender como integrar a sua aplicação em Java com o seu banco de dados MySQL.



Vídeo 01 - Apresentação

Objetivos

- Estabelecer uma conexão com banco de dados em Java.
- Realizar consultas SQL através de uma aplicação Java.

Java: conceitos básicos

Antes de aprender como utilizar a linguagem SQL através da linguagem Java, você irá rever alguns conceitos básicos sobre Java.

Como você já viu na disciplina Programação Orientada a Objetos, Java é uma linguagem de programação orientada a objetos que utiliza uma máquina virtual para executar seus programas; ou seja, quando você compila um programa em Java, o código fonte é convertido para bytecodes, que é uma representação intermediária utilizada pela máquina virtual para executar seu programa.

JDBC

JDBC (*Java Database Connectivity*) é um conjunto de classes e interfaces escritas em Java que fazem o envio de instruções SQL para qualquer banco de dados relacional. Para cada banco de dados, há um driver JDBC.

Bem, agora que sabemos o que é JDBC, vamos prosseguir com nossa aula, começando a falar sobre os pacotes em Java que serão utilizados para conectar com o seu banco de dados.

O pacote java.sql

O pacote `java.sql` fornece os meios para conectar e utilizar bancos de dados relacionais em aplicações Web ou Desktop. Em geral, cada banco de dados fornece um driver diferente para permitir a conexão do banco de dados com as aplicações em Java. No nosso caso, como estamos trabalhando com o banco de dados MySQL, temos que baixar o seu driver da internet para podermos conectar nossa aplicação em Java com o banco de dados.

Para obter o driver do MySQL, primeiro acesse o site <<http://dev.mysql.com/downloads/connector/j/>>. Depois de acessar o site, estará disponível para download o Connector/J. Em **"Select Platform:"** você deverá optar por **"Platform Independent"**. Logo abaixo serão apresentadas duas opções de download. Você deve baixar o arquivo zip ou tar.gz, escolhendo o .zip para Windows, ou tar.gz para Linux e então clicar no botão **"Download"** ao lado da opção de sua escolha. Na próxima página, clique no link **'No thanks, just start my download.'**, localizado na parte inferior da tela.

Para instalar o driver, você deve descompactar e definir o arquivo mysql-connector-java.jar no seu classpath, ou seja, o arquivo mysql-connector-java-XXX-bin.jar (onde XXX é a sua versão) deve estar listado entre as bibliotecas que sua aplicação Java irá utilizar.

Conexão com banco de dados em Java

Depois de instalar o driver, você já está pronto para conectar seu banco de dados às aplicações escritas em Java. Antes de entendermos o código a seguir, podemos copiar e executar esse código em qualquer ferramenta Java. No entanto, você também pode utilizar um editor de texto simples. Tudo depende de você preferir uma maior facilidade ou não.

Para mostrar como conectar aplicações escritas em Java com banco de dados, vamos dividir o processo em passos apresentados a seguir.

Passo 1: criar o arquivo e importar os pacotes para conexão

Primeiro, você deve criar um arquivo ConexaoMySQL.java no seu editor preferido e importar três classes Java, como listado a seguir:

```
1 import java.sql.Connection;  
2 import java.sql.DriverManager;  
3 import java.sql.SQLException;
```

Passo 2: definir a estrutura da classe que fará a conexão

Agora você deve definir uma classe em Java para ficar responsável pela conexão com o banco de dados. No nosso exemplo a seguir, a classe criada é a "ConexaoMySQL".

```
1 //Início da classe de conexão//
2 public class ConexaoMySQL {
3     public static String status = "Não conectou...";
4     //Método Construtor da Classe//
5     public ConexaoMySQL() { }
6 }
```

Passo 3: definir o método de conexão

Agora iremos definir o método que de fato irá estabelecer a conexão com o banco de dados. No código a seguir, definimos o método `getConexaoMySQL`, que tem como valor de retorno um tipo `java.sql.Connection`, ou seja, ao estabelecer a conexão, esse método irá retornar um objeto `java.sql.Connection` que irá representar a conexão criada com o banco de dados.

```
1 public static java.sql.Connection getConexaoMySQL() {
2     Connection connection = null; //atributo do tipo Connection
3     return connection;
4 }
```

Passo 4: carregar o Driver do MySQL

O nosso próximo passo é carregar o driver do MySQL para que seja possível estabelecer uma conexão com o servidor MySQL. Para tanto, devemos definir uma variável "driverName" do tipo String e atribuir o valor referente ao driver. No nosso caso, `com.mysql.jdbc.Driver`. Depois disso, você deve chamar o método `Class.forName` para carregar o driver especificado. Veja o exemplo a seguir.

```
1 String driverName = "com.mysql.jdbc.Driver";
2 try {
3     Class.forName(driverName);
4 } catch (ClassNotFoundException e) {
5     e.printStackTrace();
6 }
```

Lembre-se de que como o método **Class.forName** pode levantar uma exceção, você deve definir o bloco try/catch para tratar tal exceção.

Passo 5: configurar conexão com o MySQL

Depois de carregar o Driver, você deve configurar os parâmetros para conexão com o banco de dados. Quando seu banco de dados MySQL está instalado na sua própria máquina, você deve definir uma String como "serverName = "localhost";). Depois disso, você deve definir o nome do banco de dados através de uma String como mydatabase="mysql". Finalmente, informamos a configuração do seu usuário através da String username = "root" e definimos senha através da String password = "123456".

Lembre-se de que a informação de usuário e senha foram definidas por você no momento da instalação do MySQL. Ao final, você deve invocar o método DriverManager.getConnection, passando como parâmetro todas essas informações definidas anteriormente. Veja no exemplo de código a seguir como ficará o código responsável pela conexão.

```
1 public static java.sql.Connection getConexaoMySQL() {
2     // Atributo do tipo Connection
3     Connection connection = null;
4
5     String driverName = "com.mysql.jdbc.Driver";
6     try {
7         Class.forName(driverName);
8     } catch (ClassNotFoundException e) {
9         e.printStackTrace();
10    }
11
12    // Endereço do servidor do BD
13    String serverName = "localhost";
14
15    // Nome do seu banco de dados
16    String mydatabase = "locadora";
17
18    // String de Conexão.
19    String url = "jdbc:mysql://" + serverName + "/" + mydatabase;
20
21    // Nome de um usuário de seu BD
22    String username = "root";
23
24    // A senha de acesso do usuário informado acima.
25    String password = "123456";
26
27    try {
28        connection = DriverManager.getConnection(url, username, password);
29    } catch (SQLException e) {
30        e.printStackTrace();
31    }
32
33    return connection;
34 }
```



Vídeo 02 - Classe Conexão BD

Atividade 01

1. Explique o que é JDBC.

2. Quais são os passos para conectar uma aplicação em Java a um banco de dados relacional? Explique.

Realizando consultas usando SQL em Java

Depois que você aprendeu como conectar sua aplicação escrita em Java a um Banco de Dados MySQL, agora você irá ver como usar consultas SQL para mostrar na sua aplicação Java os dados armazenados no seu banco de dados MySQL. Para mostrar como realizar consultas, vamos dividir o processo em passos listados a seguir.

Passo 1: criar uma instância da classe Statement

Antes de tudo, vamos criar um método para realizar a consulta ao banco de dados. Você pode criar um método **public void** chamado **consultar**, dentro da classe `ConexaoMySQL`. Depois disso, você deve declarar e inicializar uma variável do tipo `Statement`. Veja como fizemos no exemplo a seguir:

```
1 public void consultar(){  
2     Statement s = null;  
3 }
```

Para obter uma instância da classe `Statement`, você deve usar a sua conexão para invocar o método `createStatement`. No exemplo mostrado anteriormente, a conexão ao banco é representada pela classe `ConexaoMySQL`. Assim, obtemos uma conexão e criamos um objeto `Statement` a partir dela. Veja, a seguir, como deve ficar o seu método `consultar` até agora:

```
1 public void consultar(){  
2     Statement s = null;  
3     Connection connection = ConexaoMySQL.getConexaoMySQL();  
4     try {  
5         s = (Statement) connection.createStatement();  
6     } catch (SQLException e) {  
7         e.printStackTrace();  
8     }
```

Note que a invocação do método `createStatement()` é envolto por um bloco `try/catch`, porque tal invocação pode levantar a exceção `SQLException`. Para evitar que sua aplicação pare de funcionar devido à ocorrência dessa exceção, o melhor é usar o bloco `try/catch` para tratar tal exceção e imprimir a pilha de execução através do comando `printStackTrace`.

Passo 2: executar a instrução SQL

O próximo passo é definir e executar a instrução SQL desejada. Para tanto, você deve declarar e inicializar uma variável para armazenar o resultado da sua consulta. Tal variável deve ser do tipo `ResultSet`. Depois disso, você deve usar a instância da classe `Statement`, criada no passo anterior, para executar o método `executeQuery(SQL)`, onde `SQL` representa a consulta SQL que você deseja executar no seu banco de dados. Note que a consulta deve ser passada como uma `String` para o comando `executeQuery`.

A seguir, você pode ver como deve ter ficado seu método consultar depois deste passo. No nosso caso, estamos usando a `String` `"Select * from clientes"` para selecionar todos os campos da tabela `Clientes`.

```
1 ResultSet r = null;
2 try {
3     r = s.executeQuery("Select * from clientes");
4 } catch (SQLException e) {
5     e.printStackTrace();
6 }
```

Note que a invocação do método `executeQuery()` é envolto por um bloco `try/catch`, porque tal invocação pode levantar a exceção `SQLException`. Para evitar que sua aplicação pare de funcionar devido à ocorrência dessa exceção, o melhor é usar o bloco `try/catch` para tratar tal exceção e imprimir a pilha de execução através do comando `printStackTrace`.

Passo 3: obter dados retornados pelo MySQL

Depois de executar a consulta, o próximo passo é percorrer o `ResultSet` para obter os dados do banco de dados. Para tanto, você deve definir uma estrutura de repetição *enquanto* (`while`) que tenha como condição de parada o método `next` do `ResultSet`. Esse método retorna `true` quando existe dado retornado pela consulta. Veja a seguir como definir um `while` para mostrar os valores retornados pela consulta à tabela `clientes`.

```
1 try {  
2     while (r.next()){  
3         System.out.println(r.getInt("cli_codigo") + " " + r.getString("cli_nome"));  
4     }  
5 } catch (SQLException e) {  
6     e.printStackTrace();  
7 }
```

Note que para obter o valor individual de cada coluna na tabela `Clientes`, você deve usar os métodos `getInt`, `getString`, `getDate`, `getDouble` etc. Esses métodos devem ser usados de acordo com o tipo da coluna no banco de dados. Note que a coluna `cli_codigo` na tabela `Clientes` é do tipo inteiro. Por esse motivo, usamos o método `getInt`. Por outro lado, a coluna `cli_nome` é do tipo `Varchar`; por isso usamos `getString`. Como parâmetro, esses métodos devem receber o nome do atributo que você deseja utilizar. Novamente, note que a invocação do método `next()` é envolto por um bloco `try/catch`, porque tal invocação pode levantar a exceção `SQLException`. Para evitar que sua aplicação pare de funcionar devido à ocorrência dessa exceção, o melhor é usar o bloco `try/catch` para tratar tal exceção e imprimir a pilha de execução através do comando `printStackTrace`.

Passo 4: fechar o `ResultSet` e a conexão

Depois de executar e percorrer os dados retornados, você sempre deve fechar a sua instância da classe `ResultSet`. Para tanto, você deve invocar o método `close`. Ao final desses passos, seu método consultar deve estar parecido com o mostrado a seguir:

```
1 public void consultar() {
2     Statement s = null;
3     Connection connection = ConexaoMySQL.getConexaoMySQL();
4
5     try {
6         s = (Statement) connection.createStatement();
7         ResultSet r = null;
8         r = s.executeQuery("Select * from clientes");
9
10        System.out.println("ID    NOME");
11        while (r.next()) {
12            System.out.println(r.getInt("cli_codigo") + " " + r.getString("cli_nome"));
13        }
14
15        r.close();
16    } catch (SQLException e) {
17        e.printStackTrace();
18    }
19 }
20 }
```



Vídeo 03 - Classe Curso



Vídeo 04 - Classe Testa Classes

Atividade 02

1. Qual a diferença entre a classe Statement e ResultSet?
2. Descreva os passos para exibir os resultados de uma consulta SQL.

Resumo

Nesta aula, aprendemos que o JDBC serve para fazer a conexão entre aplicações Java e banco de dados. Você agora já deve saber como conectamos o nosso banco de dados MySQL à nossa aplicação Java e como podemos fazer consultas e mostrar o resultado na nossa aplicação.

Autoavaliação

1. Para que serve o JDBC?
2. Por que devemos importar as classes relativas ao pacote `java.sql`? Explique.
3. Escreva uma aplicação em Java que realize as seguintes consultas ao seu banco de dados `CursoX`, criado na Aula 10:
 - a. `Select * from Alunos`
 - b. `Select * from disciplina where dis_cod = 20`

Referências

CHAN, Mark C.; GRIFFITH, Steven W.; IASI, Anthony F. **Java**: 1001 dicas de programação. São Paulo: Makron Books, 1999.

DATE, Christopher J. **Introduction to Database Systems**. 7th ed. 1999.

_____. **Introdução a sistemas de banco de dados**. Rio de Janeiro: Campus, 2000.

DEITEL, H. M.; DEITEL, P. J.; FURMANKIEWICZ, Edson. **Java, como programar**. 3. ed. Porto Alegre: Bookman, 2007.

ELMASRI, R. E.; NAVATHE, S. B. **Sistemas de banco de dados**. 4. ed. Rio de Janeiro: Addison-Wesley, 2005.

GOODRICH, Michael T.; TAMASSIA, Roberto. **Estruturas de dados e algoritmos em Java**. 2. ed. Porto Alegre: Bookman, 2001.

HEUSER, C. A. **Projeto de banco de dados**. 5. ed. Porto Alegre: Sagra-Luzzatto, 2004.

_____. **Projeto de banco de dados**. 6. ed. Porto Alegre: Editora Bookman, 2009.