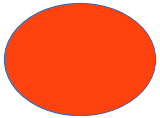
Aplicações de BD

JDBC

Aula 21
Prof. André Britto



Introdução

- Uso de bibliotecas de funções de acesso ao banco de dados (APIs).
- Biblioteca de funções (API) disponibilizam funções para o acesso e manipulação do banco de dados.
- As consultas são inseridas como parâmetros dessas funções.
- A biblioteca JDBC é disponibilizada para a linguagem Java.

Introdução

- O API JDBC é a mesma para qualquer aplicação que acesse um SGBD.
- Não há necessidade de se desenvolver aplicações voltadas para um BD específico.
- A aplicação roda em qualquer sistema operacional e pode acessar qualquer BD.

JDBC

- Características:
  - fácil mapeamento objeto para relacional;
  - independência de banco de dados;
  - independente de plataforma;
- Podemos executar comandos de manipulação de dados, execução de transações e chamada de *Stored Procedures*.

JDBC

- Para conseguir essa independência do banco de dados, a biblioteca JDBC disponibiliza métodos de acesso e manipulação do banco de dados.
  - Pacote java.sql.*
- Porém, cada fabricante de um SGBD deve disponibilizar um driver que implementa esses métodos.

Driver JDBC

- Cada fabricante disponibiliza um driver específico.
  - Arquivo .jar.
- Driver do PostgreSQL
  - http://jdbc.postgresql.org/download.html

Driver JDBC

- Essa biblioteca deve ser específica para a aplicação Java que está sendo construída:
  - Utilização de bibliotecas externas.
  - Configuração do classpath.
  - Especificação do uso da biblioteca no arquivo MANIFEST.mf

Driver JDBC

- Vários drivers podem ser carregados com o método
  - Class.forName.
- Por exemplo:
  - Class.forName ("oracle.jdbc.driver.OracleDriver");
  - Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
  - Class.forName("org.postgresql.Driver");

Exemplo

```
public class TesteJDBC {
   public TesteJDBC() {
      try{
         //Carregar driver JDBC do postgress
         Class.forName("org.postgresql.Driver");
         System.out.println("Carregou");
      } catch (ClassNotFoundException ex){ex.printStackTrace();}
   }

   public static void main(String[] args) {
      TesteJDBC teste = new TesteJDBC();
   }
}
```

Conexão - JDBC

- Um conexão com o banco de dados é especifica através da classe Connection.
  - import java.sql.Connection;
- A conexão é obtida através do método getConnection
  - DriverManager.getConnection

Conexão - JDBC

- DriverManager.getConnection
  - URL
    - Protocolo
    - Subprotocolo
    - Informação do SGBD
    - String url = "jdbc:postgresql://10.27.159.214:5432/bd_aula;
  - Usuário
  - Senha

Conexão - JDBC

```
import java.sql.*;
public Connection setConnection() throws SQLException{

String host = "10.27.159.214:5432";
String database = "bd_aula";
String url = "jdbc:postgresql://" +  host + "/" + database;
String user = "aluno";
String password = "aluno";
Connection conexao = DriverManager.getConnection(url, user, password);
System.out.println("Criou a conexao");
return conexao;
}
```

Conexão - JDBC

```
public void setConnection() throws SQLException{
    Properties info = new Properties();
   info.setProperty("user","aluno");
   info.setProperty("password","aluno");
   conexao = DriverManager.getConnection(url, info);
}
```

Conexão - JDBC

```
public void setConnection() throws SQLException{
    String url = "jdbc:postgresql://localhost/bd_aula?
user=aluno&password=aluno";
    conexao = DriverManager.getConnection(url);
}
```

Comandos - SQL

- O envio de um comando SQL é feito por meio de um Statement
- Um statement pode ser criado a partir de três classes:
  - Statement
    - é utilizado para enviar comandos SQL simples
  - PreparedStatement
    - o comando SQL é pré-compilado e utilizado posteriormente, sendo mais eficiente nos casos onde o mesmo comando é utilizado várias vezes
  - CallableStatement
    - utilizado para chamar procedimentos SQL armazenados no BD

Comandos - SQL

- Um objeto dessa classe é criado a partir do objeto connection:
  - CreatStatment();
  - prepareStatement(String sql);
  - prepareCall(String sql)

Comandos - SQL

```
public void exemploConsultaStatment(Connection conexao) throws SQLException{
   String sql = "Select cpf from universidade.professor where departamento =
'DCOMP'";
   Statement comando = conexao.createStatement();
   System.out.println("Executar consulta: " + sql);
   ResultSet resultado = comando.executeQuery(sql);
   while(resultado.next()){
      String cpf = resultado.getString("cpf");
      String cpf2 = resultado.getString(1);
     //String nome3 = resultado.getString(2);
     System.out.println(cpf);
   }
   comando.close();
}
```

Comandos - SQL

- Comandos executados
  - executeQuery(String sql)
  - executeUpdate(String sql)
- SQL é uma consulta SQL
  - Se houver algum erro, sintático ou semântico, só será checado em tempo de execução.

Comandos - SQL

```
public void exemploConsultaStatment(Connection conexao) throws SQLException{
   String sql = "Select cpf from universidade.professor where departamento =
'DCOMP'";
   Statement comando = conexao.createStatement();
   System.out.println("Executar consulta: " + sql);
   ResultSet resultado = comando.executeQuery(sql);
   while(resultado.next()){
      String cpf = resultado.getString("cpf");
      String cpf2 = resultado.getString(1);
     System.out.println(cpf);
   }
   comando.close();
}
```

Comandos - SQL

Statement st = con.createStatement();

st.executeUpdate("create table Professor (mat_professor varchar[8], nome varchar (30), salario real);");

st.executeUpdate("insert into Plunos values ('100','Alan',2000);");

st.executeUpdate(”update Professor set salario = 3000 where mat_professor = '100'");

ResultSet - JDBC

- A classe ResultSet encapsula uma tabela resultante de um Select
  - Mantém um cursor posicionado em uma linha da tabela.
  - Inicialmente este cursor está antes da primeira linha.
  - O método next() movimenta o cursos para frente.
  - Permite à aplicação pegar os dados das colunas da linha corrente através de mensagem getXXX(<coluna>)
    - XXX é o tipo da coluna
    - <coluna> é o nome da coluna ou sua posição (a partir de 1)

Comandos - SQL

```
public void exemploConsultaStatment(Connection conexao) throws SQLException{
   String sql = "Select nome from professor where departamento = 'DCOMP'";
   Statement comando = conexao.createStatement();
   System.out.println("Executar consulta: " + sql);
   ResultSet resultado = comando.executeQuery(sql);
   while(resultado.next()){
      String cpf = resultado.getString("cpf");
      String cpf2 = resultado.getString(1);

     System.out.println(cpf);
   }
   comando.close();
}
```

PreparedStament

- Podemos construir uma consulta onde os tipos dos parâmetro são especificados antes da execução da consulta.
- O tipo do parâmetro é checado antes da execução da busca.

PreparedStament

```
public void exemploConsultaPreparedStatment(Connection conexao) throws
SQLException{
   String sql = "Select cpf from universidade.professor where departamento =
?";
   PreparedStatement comando = conexao.prepareStatement(sql);
   System.out.println("Executar consulta: " + sql);
   comando.setString(1, "DCOMP");
   ResultSet resultado = comando.executeQuery();
   while(resultado.next()){
      String cpf = resultado.getString("cpf");
      System.out.println(cpf);
   }
}
```

Liberando recursos

- Após o uso precisamos fechar o statement e a conexão usando os métodos:
  - comando.close();
  - conexao.close();

CallableStatment

- Comando utilizado para fazer chamadas a stored procedures.
- Devem ser especificado os parâmetros de saída e de entrada.
- Sintaxe semelhante ao PreparedStatment.

CallableStatment

```
public void exemploCallableStatement() throws SQLException{
String sql = "{? = call universidade.calcular_aumento( ? )}";
CallableStatement call = conexao.prepareCall(sql);
call.registerOutParameter(1, Types.REAL);
call.setString(2, "P100");
call.execute();
double retorno = call.getFloat(1);
System.out.println(retorno);
call.close();

}
```

Metadados

- Permite obter informações sobre o tipo de tabela que resultou o select.
  - Quais os nomes e os tipos das colunas.
  - Número de itens da coluna.
  - Se o tipo da coluna é um tipo específico.
  - O tamanho de colunas da tabela, etc.
- Classe ResultSetMetaData

Metadados

```
public void exemploMetaDados() throws SQLException{
   Statement comando = conexao.createStatement();
   ResultSet result = comando.executeQuery("SELECT * FROM
professor");
   ResultSetMetaData meta = result.getMetaData();
   int columns = meta.getColumnCount();
   for (int i=1;i<=columns;i++) {
      System.out.println (meta.getColumnLabel(i) + "\t"
      + meta.getColumnTypeName(i));
   }
}
```

Leitura recomendada

ELMASRI, R; NAVATHE, S.B. **Sistemas de Banco de Dados**, Addison Wesley, 6ª Edição.

- Capítulo 28 e 29

Silberschatz, A; Korth H.F.; Sudarshan S. **Sistemas de Banco de Dados**, Editora Campus, 6ª Edição.

- Capítulo 5

http://jdbc.postgresql.org/documentation/head/index.html