PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ECEC-Escola de Ciências Exatas e da Computação
CMP1491-Desenvolvimento de Aplicações Web

CONEXÃO COM BANCOS DE DADOS

INTRODUÇÃO

O Modelo Cliente-Servidor (em inglês, Cliente/Server Model), em computação, segundo a Enciclopédia Livre Wikipédia, "é uma estrutura de aplicação distribuída que distribui as tarefas e cargas de trabalho entre fornecedores de um recurso ou serviço, designados como **servidores**, e os requerentes dos serviços, designados como **clientes**. Geralmente, clientes e servidores se comunicam através de uma rede de computadores, porém ambos podem estar em um mesmo computador. O servidor, que é um computador **host** (que hospeda e executa um ou mais serviços ou programas) oferece e compartilha recursos com os clientes, os quais solicitam conteúdo ou alguma função do servidor.

As aplicações Web são as mais conhecidas, as quais possuem o navegador como a base para apresentar as interfaces gráficas do usuário (cliente) e solicitam recursos de um computador host (servidor).

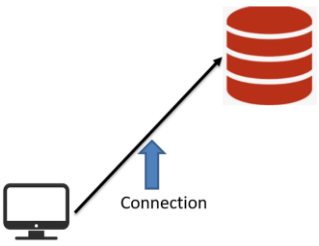
Na maioria dos casos empresariais, os serviços oferecidos por um servidor utilizam acesso a bancos de dados. Assim, torna-se importante conhecer como se efetua a conexão a um banco de dados usando a linguagem Java, na prática.

CONEXÃO COM BANCO DE DADOS USANDO A LINGUAGEM JAVA

Para se conectar com um banco de dados deve-se utilizar uma biblioteca da linguagem de programação Java, denominada de JDBC (Java DataBase Connection). Para isso deve-se utilizar três classes: Connection, ResultSet e Statement.

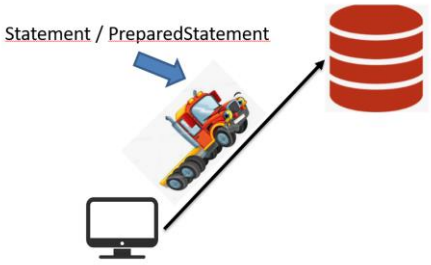
Uma Connection permite a conexão com o banco de dados.

Por analogia, considere uma instância (ou objeto) de Connection, como sendo a estrada que liga o computador do usuário e o servidor de banco de dados.



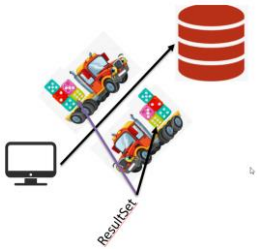
Um Statement/PreparedStatement permite preparar a estrutura que irá executar os scripts SQL.

Por analogia, considere um objeto de Statement/PreparedStatement como sendo o carro para transporte de informações na estrada "Connection".



Um ResultSet é o responsável por receber informações do banco de dados. Essas informações podem ser, por exemplo, um conjunto de linhas resultantes de um select ou apenas um retorno do banco de dados quando se efetua uma inclusão.

Por analogia, considere que uma instância (ou objeto) de um ResultSet seja a carga que o carrinho "Statement" esteja transportando na estrada "Connection".

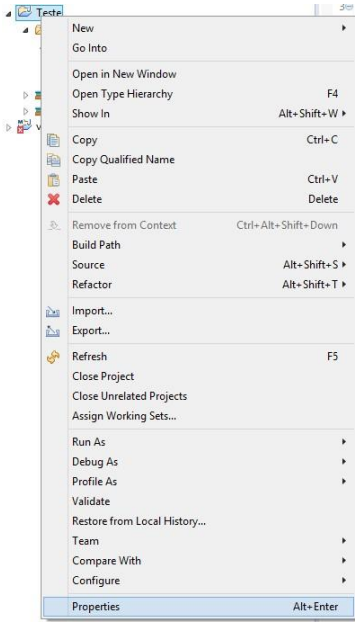


Para se conectar a qualquer banco de dados é necessário utilizar o conector do banco de dados que será utilizado. **Assim, cada modelo de banco de dados possui um arquivo de conexão específico. Dessa forma, se for utilizar o PostgreSQL, deve-se efetuar o download do conector equivalente a esse banco de dados. O conector é um arquivo com extensão .jar.**

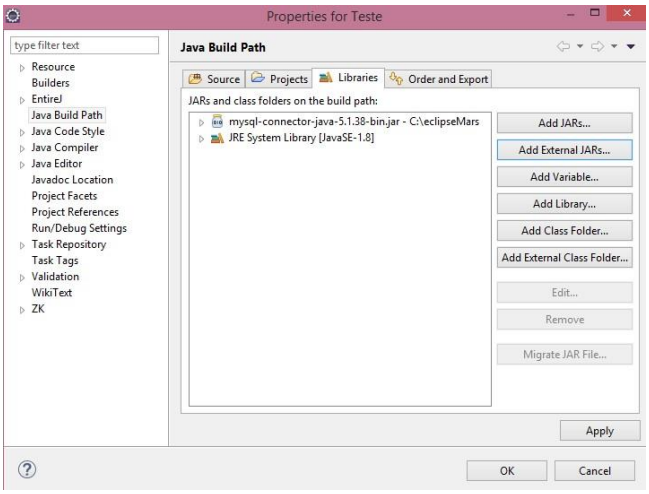
No nosso exemplo, o MySql é o modelo de banco de dados em uso. Cada SGDB possui um conector correspondente. Para isso, deve-se efetuar o download do referido arquivo no link http://dev.mysql.com/downloads/file/?id=462850

.

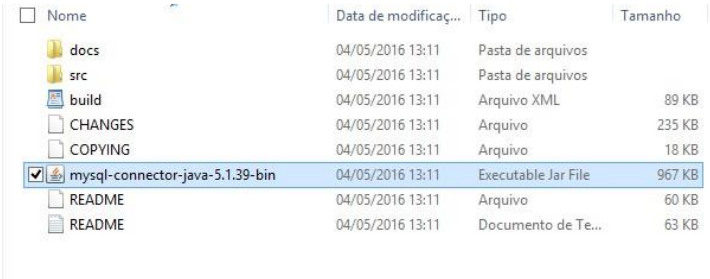
Após o download, descompacte o arquivo em uma pasta de sua preferência. Em seguida, crie um projeto no Eclipse. Em seguida, clique com o botão direito sobre o nome do seu novo projeto e selecione a opção **Properties**. Como é ilustrado seguir:



Uma nova janela será apresentada, como se segue:



Selecione o botão Add External JARS... Na nova janela, selecione a pasta onde o arquivo foi descompactado e selecione o arquivo indicado a seguir:



A partir desse momento a biblioteca do projeto permitirá a conexão com banco de dados MySQL e permitirá a execução de instruções SQL.

Assim, utilizaremos uma classe para efetuar a conexão com o banco de dados, que será denominada de Conexao. Ela usará um objeto da classe Connection para efetuar a conexão com o banco de dados e retornará um valor true caso

a conexão seja efetuada com sucesso ou um false caso a conexão não seja efetuada. O código da referida classe está indicado a seguir:

```
1
2
3   import java.sql.Connection;
4   import java.sql.DriverManager;
5   import java.sql.SQLException;
6
7   public class Conexao {
8
9       private String login="vicente";
10      private String senha="vicente";
11      private String host="localhost";
12      private String dbName="locadoradvd";
13      private String url="jdbc:mysql://"+host+"/"+dbName;
14
15      public Connection conexao = null;
16
17      public Conexao() {   }
18
19      public Connection getConnection()
20      {
21          try{
22              Class.forName("com.mysql.jdbc.Driver");
23          }
24          catch(ClassNotFoundException e)
25          {
26             return null;
27          }
28          try{
29              this.conexao = (Connection) DriverManager.getConnection(url,login,senha);
30          }
31          catch(SQLException ex)
32          {
33              return null;
34          }
35           return this.conexao;
36      }
37  }
38
```

As linhas 09 a 13 cria as variáveis para conexão; As variáveis login e senha possuem o nome do usuário e a senha de acesso ao banco de dados. A variável host indica o nome do servidor da rede ou o nome do seu computador local. No caso, é o servidor local (localhost). A variável dbName possui o nome do banco de dados (locadoradvd) e, finalmente, a url possui o nome da url de conexão com o banco de dados (“jdbc:mysql://”+host+”/”+dbName).

A linha 15 cria o objeto **conexao** da classe Connection com o valor null.

A linha 17 cria um método, que é um bloco de código que pode possuir diversos comandos entre duas chaves. No caso, o método, denominado de **Conexao**, não possui nenhum comando entre as chaves.

As linhas 19 a 36 representam a definição do método **getConnection**. Esse método possui algumas instruções específicas. Na linha 22 utiliza-se **forName** para referenciar o driver do MySQL, que será refernciado pelo Connector do MySQL. A linha 29 faz a conexão com o banco de dados utilizando a **url**, o **login** e a **senha** que foram definidos anteriormente. Note o comando **return** existente nas linhas **26**,**33** e **35**. Esse comando retorna o valor da conexão do banco de dados, podendo ser **null** (linha 26 e linha 33) ou **um valor não nulo** (linha 35), indicando que a conexão foi realizada.

O programa (ou classe) **Principal.java** é o programa que será executado no projeto, seu código inicial é o seguinte:

```
1  import java.sql.Connection;
2
3  public class Principal {
4
5      public static void main(String[] args) {
6
7          Conexao cx = new Conexao();
8          Connection conn = null;
9
10         conn = cx.getConnection();
11         if (conn == null)
12             System.out.println("A conexão não ocorreu");
13         else
14             System.out.println("O banco de dados está conectado");
15     }
16
17 }
```

Dentro do método **main** criou-se um objeto **cx** da classe **Conexao** (linha 7). Em seguida, criou-se um objeto **conn** da classe **Connection** (linha 8). A linha 10 acessa o método **getConnection** da classe **Conexao** através de **cx**. Neste caso, caso **conn** seja nulo (linha 11) será executada a linha 12, que mostrará a mensagem **A conexão não ocorreu**. Caso contrário, ou seja, **conn** não seja **nulo**, a mensagem **O banco de dados está conectado** (linha 14).

Com uma conexão disponível (**conn** diferente de **null**) será possível executar scripts SQL via programação.

Insira os trechos de código a seguir no programa Principal.java do trecho indicado anteriormente.

Cada parte do trecho a seguir possui um comentário sobre a sua respectiva ação.

**** REVISE SEUS CONHECIMENTOS SOBRE SQL**

```
//INCLUSÃO
System.out.print("Nome do Cliente:");
nome=sc.next();
System.out.print("Fone do Cliente:");
fone=sc.next();


sql = "INSERT INTO cliente (nome,fone) values ('"+nome+"','"+fone+"')";
System.out.println(sql);

ResultSet rs = null;
Statement st = conn.createStatement();

int intRs=st.executeUpdate(sql);
if (intRs>0)
    System.out.println("Inclusão realizada com sucesso");
else
    System.out.println("Inclusão não realizada");
```

O exemplo anterior faz a inclusão de informações no banco de dados utilizando o script SQL que está indicado na String **sql**:

```
sql = "INSERT INTO cliente (nome,fone) values ('"+nome+"','"+fone+"')";
```

Note que é utilizada a classe Statement que referencia a objeto **conn** de Connection em:

```
Statement st = conn.createStatement();
```

A instrução anterior deve ser executada para relacionar o objeto **st** de Statement com o objeto **conn** de Connection.

Em seguida, executa o script SQL com a seguinte instrução:

```
int intRs=st.executeUpdate(sql);
if (intRs>0)
    System.out.println("Inclusão realizada com sucesso");
else
    System.out.println("Inclusão não realizada");
```

Observe que se a variável **intRs** for maior que zero (0), implica que a inclusão é realizada com sucesso. Caso contrário, ocorreu algum erro ao tentar executar a inclusão.

Ressalta-se que o trecho de código da inclusão faz referência ao objeto **rs (da classe ResultSet)**, mas esse objeto não é utilizado. O uso de ResultSet só é utilizado quando se efetua uma consulta no banco de dados com o comando SELECT**.** Os demais comandos **UPDATE, INSERT** e **DELETE** NÃO UTILIZAM O RESULTSET**, POIS RETORNAM UM VALOR PARA INDICAR QUE A OPERAÇÃO FOI SUCEDIDA OU NÃO.**

Assim, as operações de alteração e exclusão de registros do banco de dados funcionam de forma semelhante ao INSERT do trecho anterior. Assim, são trechos auto explicativos.

```
//ALTERAÇAO
System.out.print("Código do Cliente:");
codCli=sc.nextInt();
System.out.print("Novo nome do Cliente:");
novoNome=sc.next();


 sql = "UPDATE cliente SET nome='"+novoNome+"' WHERE codCli="+codCli;
System.out.println(sql);

ResultSet rs = null;
Statement st = conn.createStatement();

int intRs=st.executeUpdate(sql);
if (intRs>0)
    System.out.println("Alteração realizada com sucesso");
else
    System.out.println("Alteração não realizada");



//EXCLUSÃO
System.out.print("Código do Cliente:");
codCli=sc.nextInt();

sql = "DELETE FROM  cliente WHERE codCli="+codCli;
System.out.println(sql);

ResultSet rs = null;
Statement st = conn.createStatement();

int intRs=st.executeUpdate(sql);
if (intRs>0)
    System.out.println("Exclusão realizada com sucesso");
else
    System.out.println("Exclusão não realizada");
```

```
//SELECT
System.out.print("Mostra todos os clientes\n");

ResultSet rs = null;
Statement st = conn.createStatement();
sql="SELECT * FROM cliente";
rs=st.executeQuery(sql);
while (rs.next())
{
    System.out.println("NOME :"+ rs.getString("nome")+" FONE :"+ rs.getString("fone")+"\n");
}
```

Observe que o trecho de código anterior (com o uso do SELECT) utiliza o objeto **rs** (da classe **ResultSet**), pois esse objeto armazena todo o conjunto de registros que são filtrados pelo uso do script indicado na variável **sql**:

```
sql="SELECT * FROM cliente";
```

Assim, ao executar a instrução `rs=st.executeQuery(sql);` a variável **rs** retorna todos os registros existentes na tabela **cliente**.

O loop com o **while(rs.next())** é efetuado enquanto existirem registros disponíveis na variável **rs**. Ou seja, a cada iteração do loop se obtem um registro em **rs**. Para obter o conteúdo de um determinado campo utiliza-se o nome da variável da classe ResultSet seguida de **getTipoCampo("nomeDoCampo")**, onde TipoCampo pode ser String, Int, Date, Double,etc e **nomeDoCampo** é o nome do campo da tabela do qual se deseja obter o seu conteúdo. No exemplo anterior, obteve-se os conteúdos dos campos **nome** e **fone** ambos do tipo String. Assim, obteve-se o seguinte:

`rs.getString("nome")` => obtém o conteúdo do campo **nome** que é do tipo **varchar**

`rs.getString("fone")` => obtém o conteúdo do campo **fone** que é do tipo **varchar**

Os exemplos anteriores utilizam a classe Statement para estabelecer a conexão com o banco de dados.

Mas, é interessante também utilizar a classe **PreparedStatement**. Essa classe oferece mais segurança ao acesso a bancos de dados, evitando, por exemplo, SQL INJECTION.

Exemplo de uso de PreparedStatement com consulta usando SELECT:

```
51  public List<Estoque> getEstoquesViaNome(String nome) {
52
53      List<Estoque> lista = new ArrayList<>();
54      Estoque estoque = new Estoque();
55      PreparedStatement pstm = null;
56      Connection conn=null;
57      try
58      {
59         conn =   this.getConnection();
60         String sql="select * from estoque where lower(descricao) like ? order by nome";
61         pstm = (PreparedStatement) conn.prepareStatement(sql);
62         pstm.setString(1,"%"+nome.toLowerCase()+"%");
63         ResultSet rs = pstm.executeQuery();
64         while (rs.next())
65         {
66             estoque = criaEstoque(rs);
67             lista.add(estoque);
68         }
69
70      }
71      catch(SQLException e)
72      {
73          //
74      }
75      return lista;
76
77  }
```

O trecho de código anterior ilustra um método para consulta de dados, o qual recebe uma String **nome** e retorna uma coleção de objetos **estoque** da classe **Estoque** (return **estoques**, na linha 93). O loop das linhas 75 a 79 permite carregar cada objeto no ArrayList **estoques** (que foi definido na linha 64).

Note que foi criado o objeto **pstmt** (da classe PreparedStatement), linha 66. Esse objeto foi alimentado com a instrução da linha 72, na qual há um script SQL de pesquisa com SELECT. Observe que esse script possui um símbolo de ? no seu final. Esse elemento informa ao PreparedStatement que se deve informar um parâmetro. No caso, isso ocorre na instrução da linha 73. Essa instrução utiliza o objeto **pstmt** informando que será indicado um parâmetro do tipo String (.setString). O 1 na instrução indica que é o primeiro parâmetro o qual é a String `"%"+nome.toLowerCase()+"%"`. Assim, o SELECT será executado de acordo com o esse parâmetro. Se existir mais de um símbolo de ? no script SQL, então deve-se criar mais de uma instrução semelhante à indicada na figura 73. Em seguida, a instrução da linha 74 será executada, retornando os registros dessa busca para a variável RecordSet **rs**.

O trecho de código a seguir retrata um exemplo de um método para inclusão ou alteração de dados, usando o INSERT/UPDATE.

```
79    public boolean salvarEstoque(Estoque estoque) {
80        boolean retorno=false;
81        String sql="";
82        PreparedStatement pstm = null;
83        Connection conn=null;
84        try
85        {
86           conn =   this.getConnection();
87           if (estoque.getId() == 0)
88           {
89               sql = "insert into estoque (descricao, precounit, quantidade) values ";
90               sql+=" (?, ?, ?)";
91               pstm = (PreparedStatement) conn.prepareStatement(sql,Statement.RETURN_GENERATED_KEYS);
92           }
93           else
94           {
95               sql = "update estoque set descricao=?, precounit=?, quantidade=?";
96               sql+=" where id=?";
97               pstm = (PreparedStatement) conn.prepareStatement(sql);
98           }
99           pstm.setString(1, estoque.getDescricao());
100          pstm.setDouble(2, estoque.getPrecounit());
101          pstm.setDouble(3,estoque.getQuantidade());
102          if (estoque.getId() !=0)
103          {
104              //update
105              pstm.setInt(4, estoque.getId());
106          }
107          int idAux=pstm.executeUpdate();
108          if (idAux==0)
109              retorno=false;
110          if (estoque.getId()==0)
111          {
112              int idInserir =  getGeneratedId(pstm);
113              estoque.setId(idInserir);
114          }
115          retorno = true;
116       }
117       catch(SQLException e)
118       {   retorno = false;}
119
120       return retorno;
121   }
```

Esse método permite salvar ou atualizar um registro, recebendo um objeto **estoque** da classe **Estoque** (linha 131). A linha 144 será executada caso o id do objeto **estoque** for zero(linha 142), indicando que esse objeto não tem chave e, por isso, é um objeto que deve ser incluído como um novo registro. Neste caso, estabelece o script SQL para a variável String **sql** usando INSERT. Essa String possui três símbolos de ?, indicando que se deve passar três parâmetros para o PreparedStatement. A linha 145 se destaca pelo fato de, além de relacionar o objeto **pstmt** com o objeto **conn** do tipo **Connection**, permite retornar a chave primária que é criada quando o registro for incluído, visto que a chave é numérica e auto incremental. Isso é realizado por **Statement.RETURN_GENERATED_KEYS**. Por outro lado, se o id do objeto **estoque** não for zero indica que esse objeto possui informação no banco de dados, assim será efetuada uma atualização. Neste caso, a linha 149 será executada usando o UPDATE**.**

Independentemente se for inclusão ou alteração, as linhas 153 a 155 serão executadas para receber os parâmetros que são indicados (ou na linha 144 ou na linha 149). A linha 153 faz referência ao primeiro parâmetro, indicado por 1, que é do tipo String. A linha 154 faz referência ao segundo parâmetro, indicado pelo 2, que é do tipo Double. O mesmo ocorre com a instrução da linha 155, que faz referência ao terceiro parâmetro, que é Double. Assim, a instrução do PreparedStatement recebe 3 parâmetros, os quais devem ser indicados corretamente pela ordem 1,2 e 3. A linha 159 executa o script, retornando um valor. Se esse valor for maior que zero ocorreu a inclusão ou a atualização.

As linhas 165 a 169 destacam o bloco do teste condicional que verifica se o id do estoque é igual a zero. Caso for verdade, ocorreu uma inclusão na linha 159. Assim, a linha 167 obtém o valor da chave primária e a armazena no id do objeto **estoque** na linha 168. Como esse método é do tipo void, como exemplo, ele não retorna o objeto **estoque**. Portanto, torna-se interessante que esse método deve ser ajustado para retornar o objeto **estoque** para o código que efetuou a chamada desse método e efetuar o tratamento adequado com o objeto de retorno.