Índice

1. Creación de una base de datos, usuario y tablas en MySQL

En la máquina virtual está ya instalado MySQL y al arrancar el sistema se inicia el servidor automáticamente. Por tanto lo que vamos a hacer es lanzar un cliente que se conecte a ese servidor para crear una base de datos, un usuario y asignarle permisos, crear tablas e insertar datos en esas tablas.

- Ejecución del cliente MySQL:

```
mysql -u root -p
```

  la contraseña que pide es la habitual.

- A continuación se crea una base de datos llamada "Libros":

```
mysql> create database Libros;
```

- Se crea un usuario (con una contraseña) y se le asignan permisos (en este caso todos) para el acceso a la base de datos creada

```
mysql>grant all on Libros.* to 'aplicacion'@'%' identified by 'apppasswd';
mysql>grant all on Libros.* to 'aplicacion'@'localhost' identified by 'apppasswd';
```

  En este caso se le permite el acceso desde la máquina "localhost" y desde cualquier otra máquina.

- En Aula Virtual hay un fichero (llamado `librosaw.sql`) con las sentencias necesarias para crear las tablas e introducir valores. Se descarga y se ejecuta:

```
mysql> use Libros;
mysql>source librosaw.sql;
```

  La primera instrucción indica que se va a usar la base de datos "Libros" y la segunda instrucción indica que se ejecuten las instrucciones que se encuentran en el fichero "librosaw.sql".

- Finalmente podemos comprobar que se ha creado el usuario, las tablas y que éstas tiene datos:

```
mysql> quit
mysql -u aplicacion -p
mysql>use Libros;
mysql>show tables;
```

```
mysql>describe authors;
mysql>select * from authors;
mysql>quit;
```

2. Configuración de un pool de conexiones y un recurso JNDI en Glassfish

Vamos a configurar Glassfish para que gestione un pool de conexiones a la base de datos que hemos creado.

1. Glassfish necesita las clases necesarias para conectarse a MySQL. Esas clases con la implementación del driver se pueden descargar de la dirección http://dev.mysql.com/downloads/connector/j, se extrae el fichero jar y se coloca en el directorio: `glassfishv3/glassfish/domains/domain1`

2. Iniciar el servidor Glassfish:

```
cd glassfishv3/glassfish/bin
./asadmin start-domain domain1
```

3. Entrar en la página de administración de Glassfish lanzando un navegador y escribiendo la dirección http://localhost:4848.

   Nota: El usuario es `admin` y la contraseña es la habitual.

4. En la página de administración ir a **Recursos** / **JDBC** / **Connection Pools** y pulsar sobre **New**

5. En la primera parte del asistente introducir los siguientes datos:

```
Name: librospool
Resource Type: javax.sql.ConnectionPoolDataSource
Database Vendor: MySQL
```

   A continuación pulsar sobre **Next**

6. En la segunda parte del asistente hay que introducir la información necesaria para realizar la conexión con la base de datos:

```
Database Name: Libros
User: aplicacion
Password: apppasswd
URL: jdbc:mysql://localhost:3306/Libros
Port Number: 3306
```

   Cuando tenemos los valores configurados pulsamos sobre **Finish**.

   Una vez configurado podemos comprobar si los valores son correctos pulsando sobre el botón **Ping**.

Una vez configurado el *pool* de conexiones vamos a hacer que se pueda acceder a él vía JNDI.

En la consola de administración de Glassfish vamos a **Resources** / **JDBC** / **JDBC Resources** y pulsamos sobre **New**

En la página de recursos JNDI hay que establecer la siguientes propiedades:

```
JNDI Name: jdbc/librospool
Pool Name: librospool
Status: Enabled
```

Los componentes de las aplicaciones que desarrollemos podrán solicitar al servidor una conexión usando el valor asignado a `JNDI Name`.

Finalmente pulsamos sobre **Finish**

La creación del pool y del recurso JNDI se puede hacer desde un terminal usando `asadmin`.

3. Uso de la base de datos en una aplicación web

Ya tenemos una base de datos creada, hemos configurado el servidor Glassfish para que gestione un pool de conexiones a esa base de datos y hemos puesto a disposición de las aplicaciones ese pool mediante JNDI.

Ahora vamos a realizar una aplicación que use ese recurso JNDI.

Creamos un proyecto del tipo "Dynamic Web Project".

3.1. Código del Java Bean

Creamos una clase (un JavaBean) `Libro` en el paquete `edu.uv.aw.taller6` con el siguiente código:

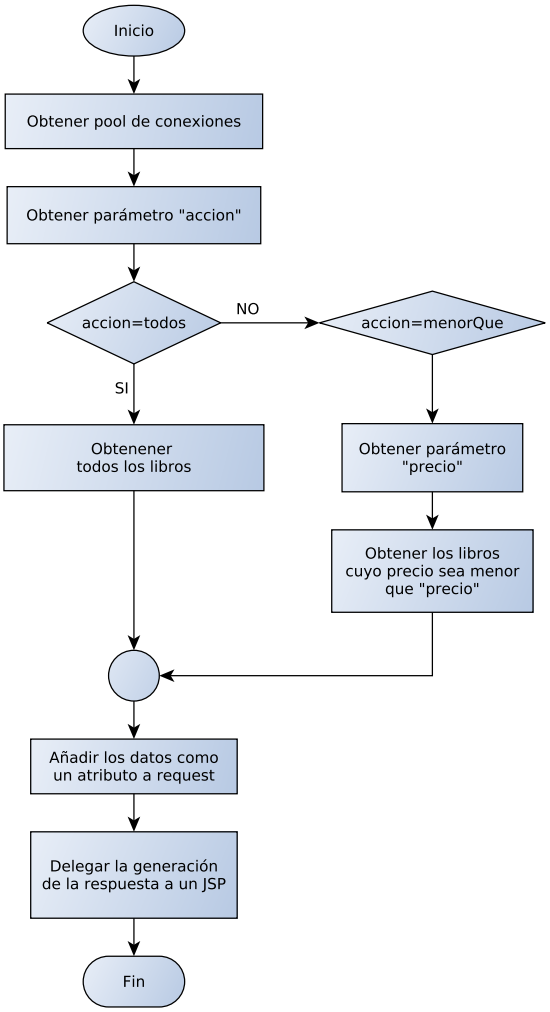
```
package edu.uv.aw.taller6;

public class Libro {
    private String id;
    private String titulo;
    private double precio;

    public Libro(){
    }

    public Libro(String id, String titulo, double precio){
      this.id = id;
      this.titulo = titulo;
      this.precio = precio;
    }

    public String getId() {
      return id;
    }

    public void setId(String id) {
      this.id = id;
    }

    public String getTitulo() {
      return titulo;
    }

    public void setTitulo(String titulo) {
      this.titulo = titulo;
    }

    public double getPrecio() {
      return precio;
    }

    public void setPrecio(double precio) {
      this.precio = precio;
    }
}
```

3.2. Código de la clase que accede a la base de datos

```
package edu.uv.aw.taller6;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Vector;

public class AccesoDatos {
   private Vector<Libro> libros;

   public AccesoDatos(Connection con){
      libros = new Vector<Libro>();
      try{
         Statement st = con.createStatement();
         ResultSet rs = st.executeQuery("select TITLE_ID, TITLE, PRICE from titles");
         rs.beforeFirst();
         while (rs.next()){
            libros.add(new Libro(rs.getString(1),rs.getString(2),rs.getDouble(3)));
         }
         rs.close();
         st.close();
      }catch(Exception ex){

      }
   }

   public Vector<Libro> getLibros(){
      return libros;
   }

   public Vector<Libro> getLibrosPrecioMenorQue(double precio){
      Vector<Libro> librosPrecio = new Vector<Libro>();
      for (Libro l:libros)
         if (l.getPrecio()<=precio)
            librosPrecio.add(l);
      return librosPrecio;
   }
}
```

3.3. Código del Servlet

En el diagrama de flujo del Servlet se muestra en la siguiente figura:



Para obtener el recurso JNDI que se ha definido se usará la anotación `@Resource` a la que se pasa el nombre JNDI.

```
package edu.uv.aw.taller6;

import java.io.IOException;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Vector;
import javax.annotation.PostConstruct;
import javax.annotation.Resource;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;

@WebServlet("/VerLibros")
public class VerLibros extends HttpServlet {
   private static final long serialVersionUID = 1L;

   @Resource(name="jdbc/librospool")
   private DataSource ds;

   private AccesoDatos ad;

    public VerLibros() {
        super();
    }
```

```
    @PostConstruct
    public void inicia() throws SQLException{
      Connection con = ds.getConnection();
      ad = new AccesoDatos(con);
    }

   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
       ServletException, IOException {
      String accion = request.getParameter("accion");

      Vector<Libro> l = new Vector<Libro>();

      if (accion.equals("todos")){
         l = ad.getLibros();
      }else
         if (accion.equals("menorQue")){
            double precio = Double.parseDouble(request.getParameter("precio"));
            l = ad.getLibrosPrecioMenorQue(precio);
         }

      request.setAttribute("libros", l);

      getServletContext().getRequestDispatcher("/mostrarLibros.jsp").forward(request,
          response);
   }

   protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
       ServletException, IOException {
      doGet(request,response);
   }

}
```

3.4. Código del JSP

En el JSP mostraremos un formulario y usaremos JSTL para los condicionales y los bucles y EL para mostrar la información del JavaBean.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core"  prefix="c"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/
    loose.dtd">
<html>
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
   <title>Mostrar libros</title>
</head>

<body>

<form action="VerLibros" method="post">
   Ver todos los libros <br/>
   <input type="hidden" name="accion" value="todos"/>
   <input type="submit" value="Buscar"/>
</form>

<form action="VerLibros" method="post">
   Ver los libros con precio menor que el dado<br/>
   <input type="hidden" name="accion" value="menorQue" />
   <input type="text" name="precio"/>
   <input type="submit" value="Buscar" />
</form>

<h1> Resultado </h1>

<c:if test="${not empty libros}">
   <table>
   <tr>
```

```
      <td>Titulo</td> <td> Precio </td>
   </tr>
   <c:forEach items="${libros}" var="libro">
      <tr>
          <td> ${libro.titulo} </td>
          <td> ${libro.precio} </td>
       </tr>
   </c:forEach>
   </table>
</c:if>

</body>
</html>
```

3.5. Instrucción para reinstalar una aplicación en Glassfish

Para reinstalar en Glassfish una aplicación que ya tenemos instalada se puede ejecutar la siguiente instrucción (asumo que se está en el directorio `glassfish3/glassfish/bin`):

```
./asadmin redeploy --name taller6 ~/ruta/taller6.war
```

donde "ruta" es la ruta donde se encuentra el fichero `war` exportado desde Eclipse.