Home Contents

JSP & MySQL Database

In this part of the JEE 5 tutorials, we will work with MySQL database.

MySQL is a leading open source database management system. It is a multi-user, multithreaded database management system. MySQL is especially popular on the web. Recently, the company that developed MySQL database was bought by Sun which was later aquired by Oracle.

Starting and stopping MySQL

```
$ sudo /etc/init.d/mysql start
```

Starting the MySQL database server.

```
$ sudo /etc/init.d/mysql stop
```

Stopping the MySQL database server.

```
$ sudo mysqladmin -p ping
Password:
Enter password:
mysqld is alive
```

With the `sudo mysqladmin -p ping` command we check if the server is running. We typed two passwords. One for the `sudo` command and one for the `mysqladmin` command.

Creating a database

In this example, we will create an empty database. The example consists of two JSP files: `index.js` and `error.jsp`. The `index.jsp` creates a new database or reports an error in `error.jsp` file.

index.jsp

```
<%@ page contentType="text/html;charset=UTF-8" %>
<%@ page errorPage="error.jsp" %>
<%@ page import="java.sql.*" %>
```

```
<html>
<head>
<title>MySQL Database creation</title>
<style>
* { font-size: 12px; font-family: Verdana }
</style>
</head>
<body>

<h2>Creation of a books database</h2>

<jsp:declaration>

Statement stmt;
Connection con;
String url = "jdbc:mysql://localhost:3306/";

</jsp:declaration>

<jsp:scriptlet><![CDATA[

Class.forName("com.mysql.jdbc.Driver");
con = DriverManager.getConnection(url, "root", "");

stmt = con.createStatement();
stmt.executeUpdate("CREATE DATABASE books");
con.close();

]]></jsp:scriptlet>

</body>
</html>
```

This JSP page connects to a server and creates a books database. If we refresh the page again, we get an error message and an error.jsp page is loaded. We cannot create two databases with a same name.

```
<%@ page errorPage="error.jsp" %>
```

If an Exception occurs in page, the server loads an `error.jsp` page.

```
<jsp:declaration>

Statement stmt;
Connection con;
String url = "jdbc:mysql://localhost:3306/";

</jsp:declaration>
```

Here we declare a `Statement` a `Connection` and a connection URL. The connection URL consists of a protocol, host, and a port number. The default MySQL port number is 3306.

```
Class.forName("com.mysql.jdbc.Driver");
```

We register a MySQL JDBC driver.

```
con = DriverManager.getConnection(url, "root", "");
```

We get a connection to the database for user root with blank password. This is the default admin account on MySQL.

```
stmt = con.createStatement();
stmt.executeUpdate("CREATE DATABASE books");
```

We get a `Statement` object and execute an SQL query.

```
con.close();
```

Finally, we close the connection.

error.jsp

```
<%@ page contentType="text/html" pageEncoding="UTF-8"%>
<%@ page isErrorPage="true" %>

<html>
<head>
<title>Error page</title>
<style>
    * { font-size: 12px; font-family: Verdana }
</style>
</head>
<body>
<h2>Error occured!</h2>
<p>Message <jsp:expression> exception.getMessage() </jsp:expression>

</body>
</html>
```

This is our error page.

```
<%@ page isErrorPage="true" %>
```

This directive enables an `exception` object for a page.

```
<p>Message <jsp:expression> exception.getMessage() </jsp:expression>
```

Here we print the error message.

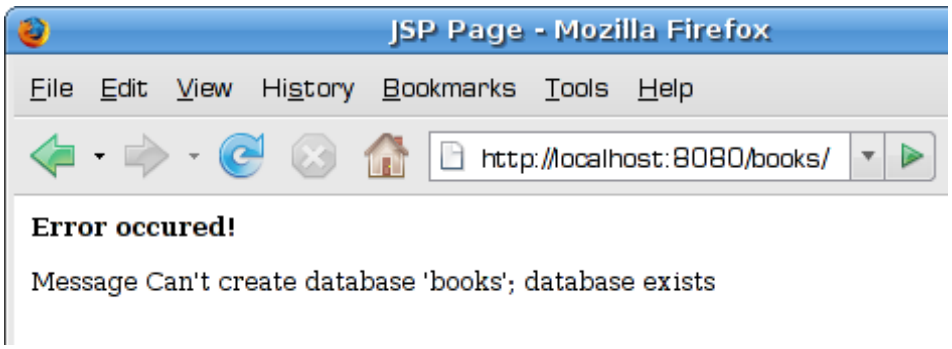

Figure: Error page

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| books              |
| mysql              |
| testdb             |
+--------------------+
4 rows in set (0.12 sec)
```

We can check from the `mysql` command line tool if the database was created.

Books

The following example will be a more complex application. We will be adding and deleting items.

```
mysql> create table books(id int not null primary key auto_increment,
              author varchar(30), title varchar(40),
              year int, remark varchar(100));
Query OK, 0 rows affected (0.13 sec)
```

We create a books table first. The table will have five columns. The id of the row, author name, tit of the book, year of publishing, and a small remark about the book.

```
mysql> describe books;
+--------+--------------+------+-----+---------+----------------+
| Field  | Type         | Null | Key | Default | Extra          |
+--------+--------------+------+-----+---------+----------------+
| id     | int(11)      | NO   | PRI | NULL    | auto_increment |
| author | varchar(30)  | YES  |     | NULL    |                |
| title  | varchar(40)  | YES  |     | NULL    |                |
| year   | int(11)      | YES  |     | NULL    |                |
| remark | varchar(100) | YES  |     | NULL    |                |
+--------+--------------+------+-----+---------+----------------+
5 rows in set (0.22 sec)
```

Here we see, how the table looks like using the `mysql` command line tool.

The example consists of four files: `style.css`, `add.jsp`, `delete.jsp`, and `BooksWorker.java`.

style.css

```
* { font-size: 12px; font-family: Verdana }

input { border: 1px solid #ccc }

a#currentTab {
    border-bottom:1px solid #fff;
}

a { color: black; text-decoration:none;
    padding:5px; border: 1px solid #aaa;
}

a:hover { background: #ccc }

td { border: 1px solid #ccc; padding: 3px }
th { border: 1px solid #ccc; padding: 3px;
    background: #009999; color: white }

.navigator { border-bottom:1px solid #aaa; width:300px; padding:5px }
```

The `style.css` file defines the style of our application. It defines the look and feel of input boxes, buttons, tables, and anchors. It also creates a navigation toolbar.

index.jsp

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

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Books database</title>
<link rel="stylesheet" href="style.css" type="text/css">
</head>
<body>

<br>

<div class="navigator">
<a id="currenttab" href="index.jsp">Add</a>
<a href="delete.jsp">Delete</a>
</div>
<%
  String author = request.getParameter("author");
  String title = request.getParameter("title");
  String year = request.getParameter("year");
  String remark = request.getParameter("remark");
  if (author != null && title != null
          && year != null && remark!= null) {
      com.zetcode.BooksWorker.Insert(author, title, year, remark);
  }
%>
```

```
<br> <br> <br>

<form method="post" action="index.jsp">
<table>
<tr>
<td>Author</td><td><input type="text" name="author"></td>
</tr>
<tr>
<td>Title</td><td><input type="text" name="title"></td>
</tr>
<tr>
<td>Year</td><td> <input type="text" name="year"></td>
</tr>
<tr>
<td>Remark</td><td> <input type="text" name="remark"></td>
</tr>
</table>

<br>
<input type="submit" value="submit">
</form>
</body>
</html>
```

The `index.jsp` file is used to add new books into the database. It defines the form, where we input data. Upon clicking submit button, it calles itself.

```
<%
  String author = request.getParameter("author");
  String title = request.getParameter("title");
  String year = request.getParameter("year");
  String remark = request.getParameter("remark");
  if (author != null && title != null
          && year != null && remark!= null) {
      com.zetcode.BooksWorker.Insert(author, title, year, remark);
  }
%>
```

This scriptlet will get the parameters from the request. If none of them is null, we call the Insert method of the `BooksWorker` class. We put all database related coding into a new class `BooksWorker`. One of the adopted principles in creating web applications is to divide the business logic from its presentation layer as much as possible.

delete.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="java.util.*" %>


<html>
<head>
<title>Delete</title>
<link rel="stylesheet" href="style.css" type="text/css">
```

```
</head>
<body>
<%
  Enumeration names = request.getParameterNames();
  while (names.hasMoreElements()) {
      String name = (String) names.nextElement();
      StringBuffer sb = new StringBuffer(name);
      sb.deleteCharAt(0);
      com.zetcode.BooksWorker.Delete(sb.toString());
  }

%>
<br>

<div class="navigator">
<a href="index.jsp">Add</a>
<a id="currenttab" href="delete.jsp">Delete</a>
</div>

<br> <br> <br>

<form action="delete.jsp" method="post">
<table>
<tr>
<th>Author</th>
<th>Title</th>
<th>Year</th>
<th>Remark</th>
</tr>
<%

  List list = com.zetcode.BooksWorker.GetBooks();
  int id = 0;
  String box = null;

  Iterator<String> it = list.iterator();

  while (it.hasNext()) {
      id = Integer.parseInt(it.next());
      out.print("<tr>");
      for (int i = 0; i < 4; i++) {
          out.print("<td>");
          out.print(it.next());
          out.print("</td>");
  }
  out.print("<td>");
  box = "<input name=r" + id + " type='checkbox'>";
  out.print(box);
  out.print("</td>");
  out.print("</tr>");
 }
%>

</table>

<br>
<input type="submit" value="Delete">
```

```
</form>

</body>
</html>
```

The `delete.jsp` file does two things. It deletes selected items from the database table and shows a list of available books.

```
<%
  Enumeration names = request.getParameterNames();
  while (names.hasMoreElements()) {
      String name = (String) names.nextElement();
      StringBuffer sb = new StringBuffer(name);
      sb.deleteCharAt(0);
      com.zetcode.BooksWorker.Delete(sb.toString());
  }
%>
```

This scriplet retrieves all parameters from the request. From the selected check boxes we receive values r1, r2 ... rn. Where the r indicates a row and the number is the id of the row in the database table. We get the number by calling the `deleteCharAt()` method and call the `Delete()` method of the BooksWorker class.

```
<%

  List list = com.zetcode.BooksWorker.GetBooks();
  int id = 0;
  String box = null;

  Iterator<String> it = list.iterator();

  while (it.hasNext()) {
      id = Integer.parseInt(it.next());
      out.print("<tr>");
      for (int i = 0; i < 4; i++) {
          out.print("<td>");
          out.print(it.next());
          out.print("</td>");
  }
  out.print("<td>");
  box = "<input name=r" + id + " type='checkbox'>";
  out.print(box);
  out.print("</td>");
  out.print("</tr>");
 }
%>
```

The second scriptlet builds the table displays all items. We get all the data by calling the `GetBooks(` method. Next we parse the data and build a table.

BooksWorker.java

```
package com.zetcode;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;


public class BooksWorker {

  static final String url = "jdbc:mysql://localhost:3306/books";

  public static void Insert(String author, String title,
          String year, String remark) {
      try {

          String insert = "INSERT INTO books(author, title, year, remark)" +
                  "VALUES (?, ?, ?, ?)";

          Class.forName("com.mysql.jdbc.Driver");
          Connection con = DriverManager.getConnection(url, "root", "");

          PreparedStatement ps = con.prepareStatement(insert);

          ps.setString(1, author);
          ps.setString(2, title);
          ps.setString(3, year);
          ps.setString(4, remark);
          ps.executeUpdate();
          con.close();

      } catch (Exception ex) {
          Logger.getLogger(BooksWorker.class.getName()).log(
                          Level.SEVERE, null, ex);
      }
  }

  public static List GetBooks() {

      List<String> list = new ArrayList<String>();

      try {

          Class.forName("com.mysql.jdbc.Driver");
          Connection con = DriverManager.getConnection(url, "root", "");

          Statement stmt = con.createStatement();

          ResultSet result = stmt.executeQuery("SELECT * FROM books");

          while(result.next())
          {
             list.add(result.getString("id"));
             list.add(result.getString("author"));
```

```
            list.add(result.getString("title"));
            list.add(result.getString("year"));
            list.add(result.getString("remark"));
        }

        con.close();

    } catch (Exception ex) {
        Logger.getLogger(BooksWorker.class.getName()).log(
                        Level.SEVERE, null, ex);
    }
        return list;
  }

  public static void Delete(String id) {
    try {
        String delete = "DELETE from books WHERE id = ?";

        Class.forName("com.mysql.jdbc.Driver");
        Connection con = DriverManager.getConnection(url, "root", "");
        PreparedStatement ps = con.prepareStatement(delete);

        ps.setString(1, id);
        ps.executeUpdate();
        con.close();

    } catch (Exception ex) {
        Logger.getLogger(BooksWorker.class.getName()).log(
            Level.SEVERE, null, ex);
    }
  }
}
```

We put all the database coding into the `BooksWorker` class, thus separating model from the view. Here we define methods for inserting data to the database, deleting data and getting all data from the table.

```
String insert = "INSERT INTO books(author, title, year, remark)" +
    "VALUES (?, ?, ?, ?)";
...
PreparedStatement ps = con.prepareStatement(insert);
ps.setString(1, author);
ps.setString(2, title);
ps.setString(3, year);
ps.setString(4, remark);
ps.executeUpdate();
```

We use prepared statements. These are `parameterized SQL statements`. The statements use question mark `?` as a marker character for passing the parameters. There are two major reasons to use parameterized statements. Securing the code against the SQL injections and easier coding due to difficulties with double and single quotes.

```
} catch (Exception ex) {
```

```
        Logger.getLogger(BooksWorker.class.getName()).log(
                        Level.SEVERE, null, ex);
    }
```

If there is some exception, we log a message about it. The messages can be viewed in a log file, which is located in domains/domain1/logs/serve.log file of the glassfish root directory.

We have put most of the business code of the application outside the JSP files. But there is still some code left, which would be better to put elsewhere. We will manage this later, when we will talk about *servlets* and *custom JSP tags*.

Using Ant to build and deploy the web application.

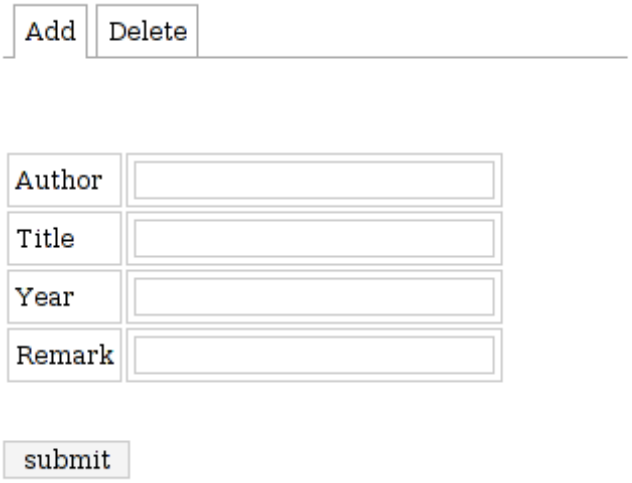

Figure: index.jsp

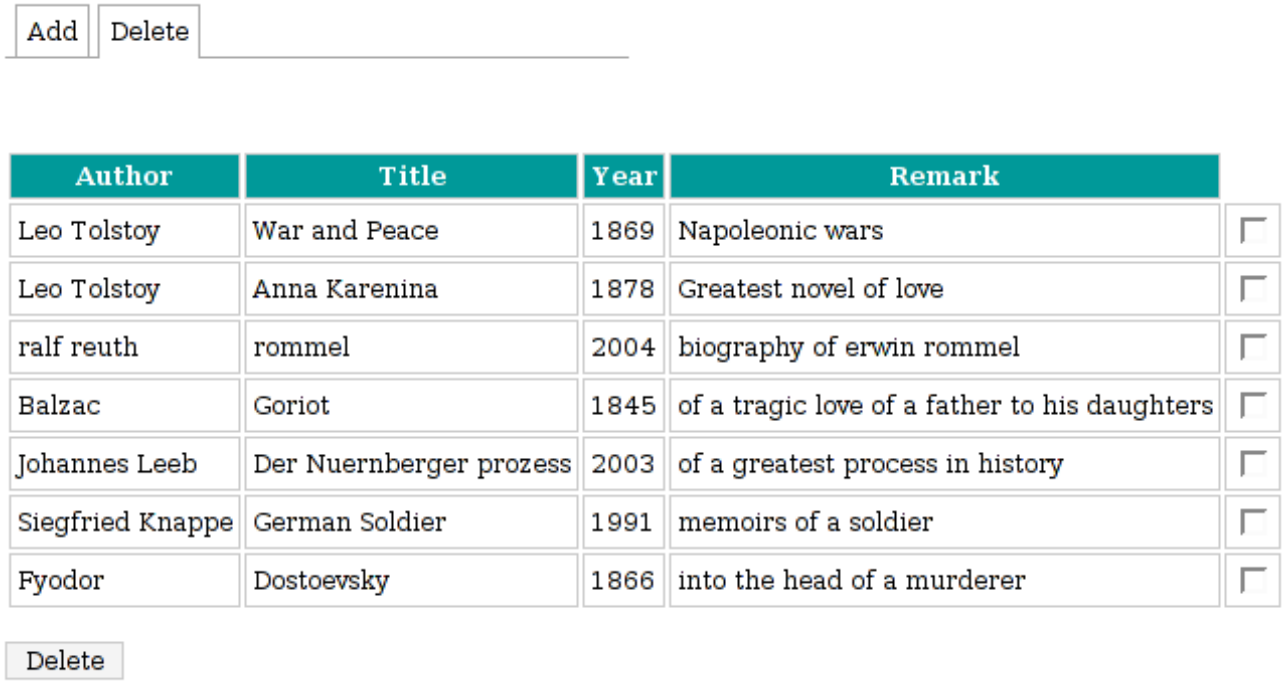

Figure: delete.jsp

In this chapter, we worked with MySQL from JSP pages.

Home ‡ Contents ‡ Top of Page

ZetCode last modified January 8, 2008