Introduction to JavaServer Pages (JSP) – Part 2

Instructor: Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cnt4714/sum2013

Department of Electrical Engineering and Computer Science
Computer Science Division
University of Central Florida
A JSPs Conversion To A Servlet

- As shown in the diagram of the lifecycle of a JSP shown in part 2 (page 2), a JSP is converted into a servlet during execution.

- While the converted servlet looks very similar in nature to those we have already seen, there are some differences.

- Within Tomcat, the servlet version of the JSP is stored in the work directory (see part 2, page 12).

- The exact directory within the work directory depends in part on your Tomcat set-up and in part on your web-application structure. The next slide illustrates the location of the servlet files that were generated for the ComputeLoan.jsp and ComputeLoan2.jsp applications that appeared in part 2 of the notes on pages 7 and 13 respectively.
Servlet Versions of JSPs in Tomcat

The servlet files corresponding to the JSPs from the loan web-application example are located in the directory:

work/Catalina/localhost/CNT4714/org/apache/jsp/jsp
Note that this package is reflected in the location shown in the previous slide.
try {
    response.setContentType("text/html");
    pageContext = _jspxFactory.getPageContext(this, request, response, 
    null, true, 8192, true);
    _jspx_page_context = pageContext;
    application = pageContext.getServletContext();
    config = pageContext.getServletConfig();
    session = pageContext.getSession();
    out = pageContext.getOut();
    _jspx_out = out;

    out.write("<!-- ComputeLoan.jsp -->\r\n");
    out.write("<html>\r\n");
    out.write("<head>\r\n");
    out.write("<title>ComputeLoan</title>\r\n");
    out.write("</head><body bgcolor=white background=images/background.jpg lang=EN-US link=blue vlink="
    out.write("style='tab-interval:5in'">\r\n");

    double loanAmount = Double.parseDouble (request.getParameter("loanAmount"));
    double annualInterestRate = Double.parseDouble (request.getParameter("annualInterestRate"));
    double numberOfYears = Integer.parseInt(request.getParameter("numberOfYears"));
    double monthlyInterestRate = annualInterestRate / 1200;
    double monthlyPayment = loanAmount * monthlyInterestRate / 
    (1 - 1 / Math.pow(1 + monthlyInterestRate, numberOfYears * 12));
    double totalPayment = monthlyPayment * numberOfYears * 12;

    out.write("\r\n");
    out.write("<b><font size = 7> Loan Details </b></font><br><br>\r\n");
    out.write("<font size = 5>\r\n");
    out.write("Loan Amount: \r\n");
    out.write("\r\n");
Import the Loan class in package code

```java
package org.apache.jsp.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import code.Loan;

public final class ComputeLoan2_jsp extends org.apache.jasper.runtime.HttpJspBase
 implements org.apache.jasper.runtime.JspSourceDependent {

private static final javax.servlet.jsp.JspFactory _jspxFactory =
 javax.servlet.jsp.JspFactory.getDefaultFactory();

private static java.util.Map<java.lang.String,java.lang.Long> _jspx_dependants;

private javax.el.ExpressionFactory _el_expressionfactory;
private org.apache.tomcat.InstanceManager _jsp_instancemanager;

public java.util.Map<java.lang.String,java.lang.Long> getDependants() {
 return _jspx_dependants;
}

public void jspInit() {
}
```
Begin the HTML content from the original ComputeLoan.jsp file now constructed from within the servlet (i.e., Java).
<jsp: setProperty> Action

• Action <jsp: setProperty> sets JavaBean property values and is most useful for mapping request parameter values to JavaBean properties.

• Request parameters can be used to set properties of primitive types boolean, byte, char, short, int, long, float and double as well as java.lang types String, Boolean, Byte, Character, Short, Integer, Long, Float, and Double.

• The table on the following page summarizes the attributes of this action.
### `<jsp:setProperty>` Action

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>name</strong></td>
<td>The ID of the JavaBean for which a property (or properties) will be set.</td>
</tr>
<tr>
<td><strong>property</strong></td>
<td>The name of the property to set. Specifying “*” for this attribute specifies that the JSP should match the request parameters to the properties of the bean. For each request parameter that matches (i.e., the name of the request parameter is identical to the bean’s property name), the corresponding property in the bean is set to the value of the parameter. If the value of the request parameter is “”, the property value in the bean remains unchanged.</td>
</tr>
<tr>
<td><strong>param</strong></td>
<td>If the request parameter names do not match bean property names, this attribute can be used to specify which request parameter should be used to obtain the value for a specific bean property. This attribute is optional. If this attribute is omitted, the request parameter names must match the bean property names.</td>
</tr>
<tr>
<td><strong>value</strong></td>
<td>The value to assign to a bean property. The value typically is the result of a JSP expression. This attribute is particularly useful for setting bean properties that cannot be set using request parameters. This attribute is optional. If this attribute is omitted, the JavaBean property must be of a type that can be set using request parameters.</td>
</tr>
</tbody>
</table>
JSP Directives

- Directives are messages to the JSP container that enable the programmer to specify page settings, such as, the error page to invoke if an error occurs (page directive), including content from other resources (include directive), and to specify custom-tag libraries for use in a JSP (taglib directive).

- Directives are delimited by `<%@` and `%>` and are processed at translation time. As such, directives do not produce any immediate output, because they are processed before the JSP accepts any requests.

- For our purposes here, the most important of these is the page directive, which we will make use of in the final example JSP. Some of the attributes of the page directive are shown on the next page.
## JSP Page Directive Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>import</td>
<td>Specifies a comma-separated list of fully qualified type names and/or packages that will be used in the current JSP.</td>
</tr>
<tr>
<td>errorPage</td>
<td>Any exceptions in the current page that are not caught are sent to the error page for processing. The error-page implicit object <code>exception</code> references the original exception.</td>
</tr>
<tr>
<td>extends</td>
<td>Specifies the class from which the translated JSP can inherit. This attribute must be a fully qualified class name.</td>
</tr>
</tbody>
</table>
<jsp: useBean> **Action**

- Action `<jsp: useBean>` enables a JSP to manipulate a Java object. This action creates a Java object or locates an existing object for use in the JSP.

- The table on the following page summarizes the attributes of this action.

- If attributes `class` and `beanName` are not specified, the JSP container attempts to locate an existing object of the type specified in attribute `type`.

- Like JSP implicit objects, objects specified with this action have scope – page, request, session, or application – which indicates where they can be used in a web application. (Recall that objects with page scope are only accessible by the page in which they are defined. For example, all JSPs that process a single request can access an object in request scope.)
**<jsp: useBean> Action**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>The name used to manipulate the Java object with actions <code>&lt;jsp:setProperty&gt;</code> and <code>&lt;jsp:getProperty&gt;</code>. A variable of this name is also declared for use in JSP scripting elements. Case sensitive.</td>
</tr>
<tr>
<td>scope</td>
<td>The scope in which the Java object is accessible – <code>page</code>, <code>request</code>, <code>session</code>, or <code>application</code>. The default scope is <code>page</code>.</td>
</tr>
<tr>
<td>class</td>
<td>The fully qualified class name of the Java object.</td>
</tr>
<tr>
<td>beanName</td>
<td>The name of the JavaBean that can be used with method <code>instantiate</code> of class <code>java.beans.Beans</code> to load a JavaBean into memory.</td>
</tr>
<tr>
<td>type</td>
<td>The type of the JavaBean. This can be the same type as the class attribute, a superclass of that type, or an interface implemented by that type. The default value is the same as for attribute class. A <code>ClassCastException</code> occurs if the Java object is not of the type specified with attribute type.</td>
</tr>
</tbody>
</table>
A JSP Using `<jsp: useBean>` Action

- A common feature on many web sites is to place rotating advertisements on their webpages. Each visit to one of these pages results in a different advertisement being displayed in the user’s web browser. Typically, when you click on the advertisement (or picture of a product) you are redirected to the website of the company that placed the advertisement or to the page that more completely describes the product.

- The next example illustrates a similar scenario, by rotating through a series of pictures (click the refresh button of your browser to simulate multiple logins or login from different browsers). In this example, I set it up to rotate through some pictures of some of my toys. If you click on a picture…you’ll be redirected to the manufacturer’s web page.
// Rotator.java
// A JavaBean that rotates pictures.
package com.cnt4174.jsp.beans;

public class Rotator {

    private String images[] = {
        "images/image1.jpg",
        "images/image2.jpg",
        "images/image3.jpg",
        "images/image4.jpg",
        "images/image5.jpg"};

    private String links[] = {
        "http://www.eddymerckx.be",
        "http://www.competitivecyclist.com",
        "http://www.bianchi-usa.com",
        "http://www.colnago.it",
        "http://www.cometkartsales.com"};

    private int selectedIndex = 0;

    // returns image file name for current ad
    public String getImage() {
        return images[selectedIndex];
    } // end method getImage

    // returns the URL for corresponding Web site
    public String getLink() {
        return links[selectedIndex];
    } // end method getLink

    // update selectedIndex so next calls to getImage and getLink return a different picture
    public void nextPic() {
        selectedIndex = (selectedIndex + 1) % images.length;
    } // end method nextPic

} // end class Rotator

//continue here -- returns the URL for corresponding Web site
<?xml version = "1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- picturerotator.jsp -->
<jsp:useBean id = "rotator" scope = "session"
class = "com.cnt4714.jsp.beans.Rotator" />
<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
 <title>PictureRotator Example</title>
 <style type = "text/css">
  .big { font-family: helvetica, arial, sans-serif; font-weight: bold; font-size: 2em }
 </style>
 <%%-- update picture --%%>
 <%% rotator.nextPic(); %>
 </head>
<body>
 <p class = "big">PictureRotator Example</p>
 <p>
  <a href = "<jsp:getProperty name = "rotator"
     property = "link" />">
   <img src = "<jsp:getProperty name = "rotator"
     property = "image" /
   alt = "picture" />
  </a>
 </p>
</body>
</html>
PictureRotator Example

First image seen in the rotation of five images.
PictureRotator Example

Second image seen in the rotation of five images.
PictureRotator Example

Another image seen in the rotation of five images.
PictureRotator Example

Fifth and final image seen in the rotation of five images.
Redirected to web site by clicking on the image
More Details On Using Beans

• The Rotator bean has three elements: getImage, getLink, and nextPic.
  – Method getImage returns the image file name for the picture to be displayed.
  – Method getLink returns the hyperlink to the manufacturer/supplier of the “toy”.
  – Method nextPic updates the Rotator so that the next calls to getImage and getLink will return information for a different picture.

• Methods getImage and getLink each represent a read-only JavaBean property – image and link, respectively. These are read-only properties because no set methods are provided to change their values.

• Rotator keeps track of the current picture with its selectedIndex variable, which is updated by invoking method nextPic.
More Details On Using Beans (cont.)

- JavaBeans were originally intended to be manipulated visually in visual development environments (often called builder tools or IDEs).

- Builder tools that support beans provide programmers with tremendous flexibility by allowing for the reuse and integration of existing disparate components that, in many cases, were never intended to be used together.

- When used in an IDE, JavaBeans adhere to the following coding conventions:
  1. Implements the Serializable interface.
  2. Provides a public no-argument (default) constructor.
  3. Provides `get` and/or `set` methods for properties (which are normally implemented as fields.)
More Details On Using Beans (cont.)

• When used on the server side, such as within a JSP or a servlet, JavaBeans are less restricted.
  – Notice for example, that the Rotator bean does not implement the Serializable interface because there is no need to save and load the Rotator bean as a file.

• The JSP picturerotator.jsp (see page 6) obtains a reference to an instance of class Rotator. The id for the bean is rotator. The JSP uses this name to manipulate the bean. The scope of the object is session, so that every client will see the same sequence of pictures during their browsing sessions.
More Details On Using Beans (cont.)

• When `picturerotator.jsp` receives a request from a new client, the JSP container creates the bean and stores it in that client’s `session` (an `HttpSession` object).

• In each request to this JSP, the `rotator` reference which is created is used to invoke the `Rotator` bean’s `nextPic` method. Therefore, each request will receive the next picture selected by the `Rotator` bean.

• Notice the two `<jsp:getProperty>` actions in the `picturerotator.jsp` file. One of these obtains the `link` property value from the bean, the other obtains the `image` property value.

  – Action `<jsp:getProperty>` has two attributes: `name` and `property`, which specify the bean object to manipulate and the property to get.
More Details On Using Beans (cont.)

• Action `<jsp:getProperty>` has two attributes: name and property, which specify the bean object to manipulate and the property to get.
  – If the JavaBean object uses standard JavaBean naming conventions, the method used to obtain the `link` property value from the bean should be `getLink`.
  – Action `<jsp:getProperty>` invokes `getLink` on the bean referenced with `rotator`, converts the return value into a `String` and outputs the `String` as a part of the response to the client.
More Details On Using Beans (cont.)

- The link and image properties can also be obtained with JSP expressions.
  - The action `<jsp:getProperty>` (see page 6 for location, the line looks like: `<a href="<jsp:getProperty name="rotator" property="link"/>">`) could be replaced with the expression: `<%= rotator.getLink() %>`
  
  - Similarly, the action `<jsp:getProperty>` (see page 6 for location, the line looks like: `<img src="<jsp:getProperty name="rotator" property="image"/>" alt="picture"/>`) could be replaced with the expression:
    `<%= rotator.getImage() %>`

- However, the benefit of using actions is that someone who is unfamiliar with Java can be told the name of a property and the name of a bean, and it is the action’s responsibility to invoke the appropriate methods. The Java programmer’s job is to create a bean that supports the capabilities required by the page designer.
Final JSP Example - GuestBook

• Our final JSP example will illustrate many of the techniques that we’ve covered in dealing with JDBC, servlets, and JSPs.

• This example constructs a simple MySQL database to maintain a guest book that includes a guest’s first name, last name, and email address.
  – Once a guest enters their name into the guestbook, they will see a webpage containing all the guests in the guest book. Each email address is displayed as a hyperlink that makes it possible for guests to send email to another guest.

• This example illustrates the `<jsp: setProperty>` action, the JSP page directive, JSP error pages, and using JDBC from a JSP.
GuestBean.java

// GuestBean.java
// JavaBean to store data for a guest in the guest book.
package com.cnt4714.jsp.beans;

public class GuestBean
{
    private String firstName;
    private String lastName;
    private String email;

    // set the guest's first name
    public void setFirstName( String name )
    {
        firstName = name;
    } // end method setFirstName

    // get the guest's first name
    public String getFirstName()
    {
        return firstName;
    } // end method getFirstName

    // set the guest's last name
    public void setLastName( String name )
    {
        lastName = name;
    } // end method setLastName

    // get the guest's last name
    public String getLastName()
    {
        return lastName;
    } // end method getLastName

    // set the guest's email
    public void setEmail( String email )
    {
        this.email = email;
    } // end method setEmail

    // get the guest's email
    public String getEmail()
    {
        return this.email;
    } // end method getEmail

    // set the guest's name
    public void setName( String name )
    {
        this.firstName = name;
        this.lastName = name;
        this.email = name;
    } // end method setName

    // get the guest's name
    public String getName()
    {
        return this.firstName + ' ' + this.lastName + ' ' + this.email;
    } // end method getName

} // end class GuestBean

This JavaBean maintains information for one guest.
GuestBean.java (cont.)

// set the guest's last name
public void setLastName( String name )
{
    lastName = name;
} // end method setLastName

// get the guest's last name
public String getLastName()
{
    return lastName;
} // end method getLastName

// set the guest's email address
public void setEmail( String address )
{
    email = address;
} // end method setEmail

// get the guest's email address
public String getEmail()
{
    return email;
} // end method getEmail

} // end class GuestBean
// GuestDataBean.java
// Class GuestDataBean makes a database connection and supports
// inserting and retrieving data from the database.
package com.cnt4714.jsp.beans;

import java.sql.SQLException;
import javax.sql.rowset.CachedRowSet;
import java.util.ArrayList;
import com.sun.rowset.CachedRowSetImpl; // CachedRowSet implementation

public class GuestDataBean
{
    private CachedRowSet rowSet;

    // construct TitlesBean object
    public GuestDataBean() throws Exception
    {
        // load the MySQL driver
        Class.forName( "com.mysql.jdbc.Driver" );

        // specify properties of CachedRowSet
        rowSet = new CachedRowSetImpl();
        rowSet.setUrl( "jdbc:mysql://localhost:3310/guestbook" );
        rowSet.setUsername( "root" );
        rowSet.setPassword( "root" );
    } // constructor

    // insert data
    public void insertData(String name, String message)
    {
        // database query
        String query = "INSERT INTO guestbook (name, message) VALUES (?, ?)";

        // execute query
        rowSet.execute( query, name, message );
    }

    // retrieve data
    public ArrayList<String> retrieveData()
    {
        ArrayList<String> data = new ArrayList();

        // database query
        String query = "SELECT name, message FROM guestbook";

        // execute query
        rowSet.execute( query );

        // get result set
        CachedRowSet rs = rowSet.getResultSet();

        // iterate through result set
        while (rs.next())
        {
            String name = rs.getString( "name" );
            String message = rs.getString( "message" );

            data.add( name + " - " + message );
        }

        return data;
    }

} // class
// obtain list of titles
    rowSet.setCommand(
        "SELECT firstName, lastName, email FROM guests" );
    rowSet.execute();
} // end GuestDataBean constructor

// return an ArrayList of GuestBeans
public ArrayList< GuestBean > getGuestList() throws SQLException
{
    ArrayList< GuestBean > guestList = new ArrayList< GuestBean >();

    rowSet.beforeFirst(); // move cursor before the first row

    // get row data
    while ( rowSet.next() )
    {
        GuestBean guest = new GuestBean();

        guest.setFirstName( rowSet.getString( 1 ) );
        guest.setLastName( rowSet.getString( 2 ) );
        guest.setEmail( rowSet.getString( 3 ) );

        guestList.add( guest );
    } // end while
return guestList;
} // end method getGuestList

// insert a guest in guestbook database
public void addGuest( GuestBean guest ) throws SQLException
{
    rowSet.moveToInsertRow(); // move cursor to the insert row

    // update the three columns of the insert row
    rowSet.updateString( 1, guest.getFirstName() );
    rowSet.updateString( 2, guest.getLastName() );
    rowSet.updateString( 3, guest.getEmail() );
    rowSet.insertRow(); // insert row to rowSet
    rowSet.moveToCurrentRow(); // move cursor to the current row
    //rowSet.acceptChanges(); // force propagation of changes to database
} // end method addGuest
} // end class GuestDataBean

Note that the acceptChanges() method forces MySQL to perform a commit operation to make the changes permanent in the database. If your MySQL installation had autocommit set on (forcing automatic commits, you'll want to comment out this line, or in the reverse, if your MySQL has autocommit set off, you'll need to include this line of code.
GuestBookLogin.jsp

<?xml version = "1.0"?><!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- guestBookLogin.jsp -->

<%@ page errorPage = "guestBookErrorPage.jsp" %>

<%@ include file = "guestBookErrorPage.jsp" %>

<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
<title>Guest Book Login</title>
<style type = "text/css">
body
{
  font-family: tahoma, helvetica, arial, sans-serif;
}

All uncaught exceptions are forwarded to guestBookErrorPage.jsp for processing.

GuestBookLogin.jsp is a modified version of our welcome1.jsp and welcome1 servlet that we’ve already seen. It displays a form that the guest uses to enter their information. When the form is submitted, GuestBookLogin.jsp is requested again so that it can ensure that all of the data is entered. If not, the form is regenerated until the guest enters all information. If all information is entered, then this JSP forwards the request to guestBookView.jsp to display the contents of the guest book.
GuestBookLogin.jsp (cont.)

table, tr, td {
    font-size: 1.4em;
    border: 3px groove;
    padding: 5px;
    background-color: #dddddd;
}
</style>
</head>
<body>
<jsp:setProperty name = "guest" property = "*" />
<% // start scriptlet
    if ( guest.getFirstName() == null ||
        guest.getLastName() == null ||
        guest.getEmail() == null )
    {
    %>
    <!-- end scriptlet to insert fixed template data -->

    <form method = "post" action = "guestBookLogin.jsp">
        <p>Enter your first name, last name and email address to register in our guest book.</p>
        <table>
            <tr>
                <td>First name</td>
                <td>
                    <input type = "text" name = "firstName" />
                </td>
            </tr>
        </table>
    </form>
    </%
}
Once the guest has entered their information into the database, the guestBookView is generated via the \(<\$\text{jsp: forward} \>$\) action which invokes the guestBookView JSP.
<?xml version = "1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- guestBookView.jsp -->

<%@ page errorPage = "guestBookErrorPage.jsp" %>
<%@ page import = "java.util.*" %>
<%@ page import = "com.cnt4714.jsp.beans.*" %>

<%-- GuestDataBean to obtain guest list --%>
jsp:useBean id = "guestData" scope = "request"
class = "com.cnt4714.jsp.beans.GuestDataBean" />

<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
<title>Guest List</title>
<style type = "text/css">
body
{
  font-family: tahoma, helvetica, arial, sans-serif;
}
</style>
</head>
<body>
These three page directives specify that the error page for this JSP is guestBookErrorPage.jsp, that classes from package java.util are used in this JSP, and classes from the package com.cnt4714.jsp.beans are also used.
GuestBookView.jsp (cont.)

table, tr, td, th
{
  text-align: center;
  font-size: 1.4em;
  border: 3px groove;
  padding: 5px;
  background-color: #dddddd;
}
</style>
</head>
<body>
<p style = "font-size: 2em;">Guest List</p>
<table>
<thead>
<tr>
  <th style = "width: 100px;">Last name</th>
  <th style = "width: 100px;">First name</th>
  <th style = "width: 200px;">Email</th>
</tr>
</thead>
<tbody>
<% // start scriptlet
  List guestList = guestData.getGuestList();
  Iterator guestListIterator = guestList.iterator();
  GuestBean guest;
%>
while ( guestListIterator.hasNext() )
{
    guest = (GuestBean) guestListIterator.next();
    %> <%- end scriptlet; insert fixed template data --%>
    <tr>
        <td><%= guest.getLastName() %></td>
        <td><%= guest.getFirstName() %></td>
        <td>
            <a href = "mailto:<%= guest.getEmail() %>">
                <%= guest.getEmail() %></a>
        </td>
    </tr>
    <% // continue scriptlet
    } // end while
    %> <%- end scriptlet --%>
</tbody>
</table>
</body>
</html>
guestBookErrorPage.jsp

<?xml version = "1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- guestBookErrorPage.jsp -->

<%@ page isErrorPage = "true" %>
<%@ page import = "java.util.*" %>
<%@ page import = "java.sql.*" %>

<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
<title>Error!</title>
<style type = "text/css">
.bigRed {   font-size: 2em;   color: red;     font-weight: bold;      }
</style>
</head>
<body>
<p class = "bigRed">
<% // scriptlet to determine exception type
    // and output beginning of error message
    if ( exception instanceof SQLException )
    {
    %>

A SQLException
<%}
} // end if
else if ( exception instanceof ClassNotFoundException )
{
%
A ClassNotFoundException
<%}
} // end else if
else
{
%
A general exception
<%}
} // end else
%
<%-- end scriptlet to insert fixed template data --%>
<%-- continue error message output --%>
ocurred while interacting with the guestbook database.
</p>
<p class = "bigRed"> The error message was:<br />
<%= exception.getMessage() %></p>
</p>
<p class = "bigRed">Please try again later</p>
</body>
</html>
To run this example you will need to create the database named guestbook and the table named guests with the schema shown above.

The script named “guestbookscript.sql” is on the course code page for you to use.
Welcome to the CNT 4714 JSP Driven Guest Book

Enter your first name, last name and email address to register in our guest book.

First name: Mark
Last name: Llewellyn
Email: markl@cs.ucf.edu

Initial screen for client to enter information to be sent to the database.
User’s screen after they click the submit button after entering their information into the form.

Once they click the submit button the guestBookLogin JSP forwards to the GuestBookView JSP to display the contents of the guest book (the database) at that point in time.
After a number of people have entered information into the database.

Once information is entered into the database, the guestBookLogin JSP forwards to the GuestBookView JSP to display the contents of the guest book.
```
select * from guests
```
Causing An Error From GuestBookLogin JSP

Welcome to the CNT 4714 JSP Driven Guest Book

Enter your first name, last name and email address to register in our guest book.

First name
Eva

Last name
Mendes

Email
markl@cs.ucf.edu

Email address is the primary key and this one will be a duplicate value when the user clicks the submit button. Next page illustrates the results.
A SQLException occurred while interacting with the guestbook database.

The error message was:
7 conflicts while synchronizing

Please try again later