Database Web Programming

COSC 304 – Introduction to Database Systems

Database Web Programming Overview

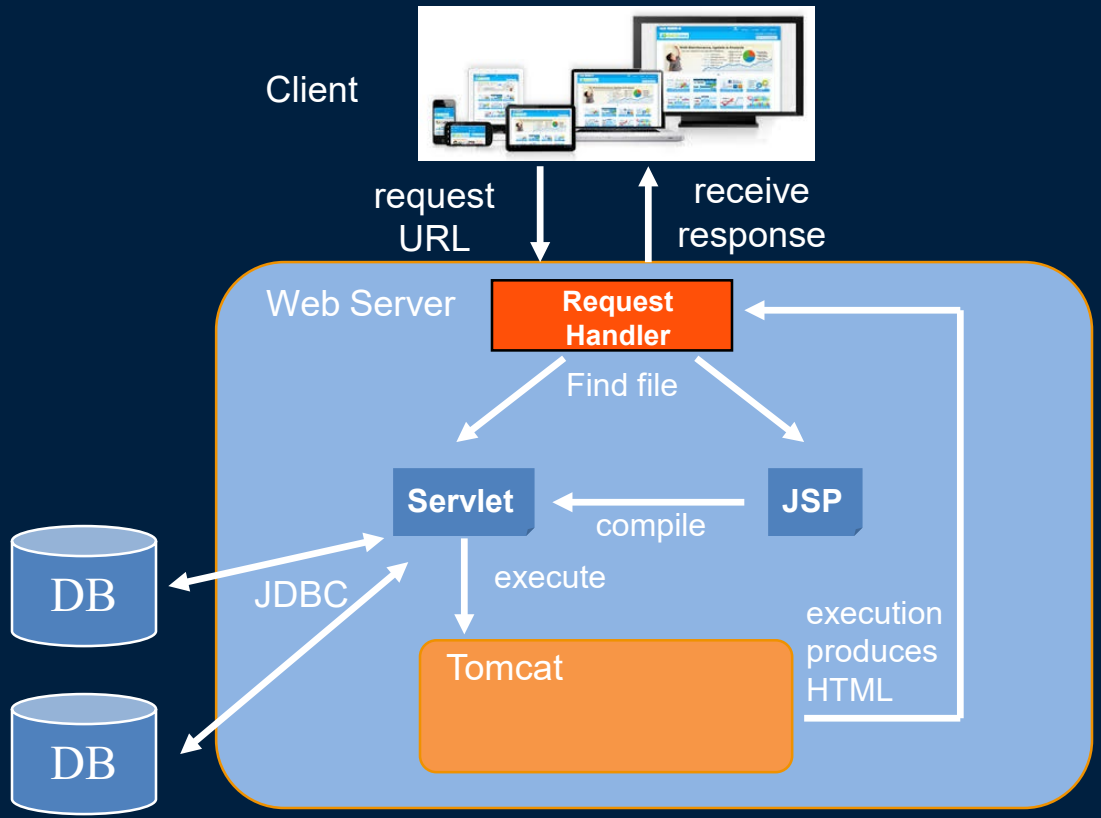
Web applications consist of a client interface and a server component.

The client interface does not access the database directly. The server code accesses the database and provides data as needed to the client.

- The server code may be JSP/Servlets, PHP, Python, etc.
- The client code is HTML/JavaScript.

HTTP is a stateless protocol which requires special handling to remember client state.

Dynamic Web Server Architecture



Hypertext Markup Language (HTML)

Hypertext Markup Language (***HTML***) describes the layout of a web page including structure, style, images, and links.

An HTML document has ***tags*** in angle brackets **<** and **>** as markup.

- In HTML 5, tags are not case-sensitive. We will use lower case as convention.

Example:

```
<html>
  <head>
    <title>Example HTML with Hyperlink</title>
  </head>
  <body>  <!-- This is a comment-->
    <p>Link: <a href="https://www.apple.com">Apple</a></p>
  </body>
</html>
```

Tables in HTML

Data can be displayed in tables using the **<table></table>** tags.

- Rows are enclosed in table row **<tr></tr>** tags, and each cell is denoted using table data **<td></td>** tags. A header cell uses the **<th></th>** tags.
- A table may have a caption centered at the top of the table by using the **<caption></caption>** tags.

```
<table border="1" style="background-color:green">
  <caption>This is my table caption.</caption>
  <tr><th>Tag</th><th>Purpose</th></tr>
  <tr><td>table</td><td>Start and end entire table</td></tr>
  <tr><td>tr</td><td>Start and end each row</td></tr>
  <tr><td>th</td><td>Special column header formatting</td></tr>
  <tr><td>td</td><td>Start and end each data cell</td></tr>
  <tr><td colspan="2">COLSPAN has cell span multiple columns</td></tr>
   <tr><td rowspan="2">ROWSPAN</td><td>ROWSPAN spans rows</td></tr>
</table>
```

Dynamic HTML and JDBC

Dynamic HTML involves constructing HTML output dynamically on the server then sending it to the client.

By building the content dynamically, it can have different content for each user. Typically, this dynamic content is retrieved from databases and then inserted into the HTML.

The two standard ways for generating dynamic content using Java are ***Java Server Pages*** (***JSPs***) and ***Servlets***. Since both use Java, they can use JDBC to retrieve dynamic, database content.

JSP/Servlet Question

Question: **True or False:** Java JSP/Servlet code runs in the browser.

A) true

B) false

JSP/Servlet Question #2

Question: **True or False:** JavaScript running in the browser can connect to a database directly using JDBC.

A) true

B) false

Java Server Pages (JSPs)

Java Server Pages (***JSPs***) provide a layer above `Servlets` that allow you to mix HTML and calls to Java code.

JSPs are converted into `Servlets`, and are easier to work with for those familiar with HTML.

Java Server Pages (JSPs) (2)

How to create JSPs:

- 1) Create a file with a `.jsp` extension that contains HTML.
- 2) Include JSP Scripting Elements in the file such as:
  - Expressions `<%= expression %>`
    - Note that an expression returns a string into the HTML.
  - Scriptlets `<% code %>`
  - Declarations `<%! Code %>`
  - Directives `<%@ variable directive %>`
  - Comments `<%-- JSP Comment -->`
- 3) Can use pre-defined variables:
  - `request` `HttpServletRequest`
  - `response` `HttpServletResponse`
  - `session` `HttpSession`
  - `out` `PrintWriter`
    - Prints to HTML document.

JSP "Hello World!"

```
<html>
<head>
<title>Hello World in JSP</title>
</head>
<body>

<% out.println("Hello World!"); %>

</body>
</html>
```

← Java code embedded in HTML file.

Useful JSP Code

Include another file:

```
<%@ include file="commontop.html" %>
```

Import from the Java library:

```
<%@ page import="java.util.*" %>
```

Declare and use a variable:

```
<%! private int accessCount = 0 %>
<h2>Accesses to page since server reboot:
<%= ++accessCount %></h2>
```

Access session and request information:

```
<h2>Current time: <%= new java.util.Date() %></h2>
<h2>Remote Host: <%= request.getRemoteHost()%></h2>
<h2>Session ID: <%= session.getID() %></h2>
```

JSP and JDBC

```
<%@ page import="java.sql.*" %>
<html><head><title>Query Results Using JSP</title></head><body>
<% // Note: Need to define url, uid, pw
  try
  ( Connection con = DriverManager.getConnection(url, uid, pw);
    Statement stmt = con.createStatement(); )
  {
    String sql = "SELECT ename, salary FROM emp";
    ResultSet rst = stmt.executeQuery(sql);
    out.println("<table><tr><th>Name</th><th>Salary</th></tr>");
    while (rst.next()) {
    {  out.println("<tr><td>"+rst.getString(1)+"</td>"
                  +"<td>"+rst.getDouble(2)+"</td></tr>"); }
    out.println("</table>");
  } catch (SQLException ex)       { out.println(ex); }
%></body></html>
```

Answering Queries using HTML Forms

One of the common uses of dynamic web pages is to construct answers to user queries expressed using HTML forms.

The HTML code will contain a `FORM` and the `FORM ACTION` will indicate the server code to call to process the request.

In our example, the JSP or Servlet gets the HTML request (and parameters in the URL), queries the database, and returns the answers in a table.

HTTP GET and POST Methods

GET and POST are two ways a HTTP client can communicate with a web server. GET is used for getting data from the server, and POST is used for sending data there.

- GET appends the form data (called a query string) to an URL, in the form of key/value pairs, for example, name=John.
  - In the query string, key/value pairs are separated by & characters, spaces are converted to + characters, and special characters are converted to their hexadecimal equivalents.
  - Since the query string is in the URL, the page can be bookmarked. The query string is usually limited to a relatively small amount of data.
- POST passes data of unlimited length as an HTTP request body to the server. The user working in the client Web browser cannot see the data that is being sent, so POST requests are ideal for sending confidential or large amounts of data to the server.

GET vs. POST Question

Question: Select a **true** statement.

A) Data sent using `POST` is passed as part of the URL.

B) A `GET` request with all parameters cannot be bookmarked.

C) Your JSP/servlet can do the same action for both `GET` and `POST` requests.

D) A `GET` request is used to send a large amount of data.

HTML Form

```
<html>
<head><title>Querying Using JSP and Forms</title></head>
<body>

<h1>Enter the name and/or department to search for:</h1>

<form method="get"
   action="https://cosc304.ok.ubc.ca/tomcat/EmpQuery.jsp">
Name:<input type="text" name="empname" size="25">
Dept:<input type="text" name="deptnum" size="5">
<input type="submit" value="Submit">
<input type="reset" value="Reset">
</form>

</body></html>
```

FORM action and type

FORM buttons and input fields

Reading Parameters

Parameters passed into the JSP are read using the `request` object:

```
String empName = request.getParameter("empname");
String deptNum = request.getParameter("deptnum");
```

Reading Parameters
Badly Building a Query – SQL Injection!

```
String empName = request.getParameter("empname");
String deptNum = request.getParameter("deptnum");
try
( Connection con = DriverManager.getConnection(url, uid, pw);
  Statement stmt = con.createStatement(); )
{
  String SQLSt="SELECT ename, salary, dno FROM emp WHERE 1=1";
  if (empName != null && !empName.equals(""))
    SQLSt = SQLSt + " and ename LIKE '%"+empName+"%'";

  if (deptNum != null && !deptNum.equals(""))
    SQLSt = SQLSt + " and dno ='"+deptNum+"'";

  ResultSet rst = stmt.executeQuery(SQLSt);
  ...
```

Building a Query using a PreparedStatement

```
String empName = request.getParameter("empname");
String deptNum = request.getParameter("deptnum");
try
( Connection con = DriverManager.getConnection(url, uid, pw);)
{
  String sql = "SELECT ename, salary, dno FROM emp";
  boolean hasEmp = empName != null && !empName.equals("");
  boolean hasDept= deptNum != null && !deptNum.equals("");
  PreparedStatement pstmt=null;
  ResultSet rst = null;
  if (!hasEmp && !hasDept)
  {
    pstmt = con.prepareStatement(sql);
    rst = pstmt.executeQuery();
  }
```

Building a Query using a PreparedStatement

```
else if (hasEmp)
{ empName = "%"+empName+"%";
  sql += " WHERE ename LIKE ?";
  if (hasDept)  sql += " AND dno = ?";
  pstmt = con.prepareStatement(sql);
  pstmt.setString(1, empName);
  if (hasDept) pstmt.setString(2, deptNum);
  rst = pstmt.executeQuery();
}
else if (hasDept)
{ sql += " WHERE dno = ?";
  pstmt = con.prepareStatement(sql);
  pstmt.setString(1, deptNum);
  rst = pstmt.executeQuery();
}
```

JSP Examples
Implementing Login and Security

How do you password protect files in your web site?

The basic idea is to:

- Create a login page like `login.jsp`.
- Create a page to validate the user login (often by connecting to the database to determine if given valid user id/password).
- Create a file containing JSP code that is included in every one of your protected pages. This code:
  - Examines if a session variable flag is set indicating if logged in.
  - If user is logged in, show page. Otherwise redirect to login page.

Implementing Login and Security

login.jsp

```
<html>
<head><title>Login Screen</title></head>
<body>
<center>
<h3>Please Login to System</h3>

<%
// Print prior error login message if present
if  (session.getAttribute("loginMessage") != null)
    out.println("<p>"+
        session.getAttribute("loginMessage").toString()+"</p>");
%>

<br>
```

Implementing Login and Security

login.jsp (2)

```
<form name="MyForm" method=post action="validateLogin.jsp">
<table width="40%" border="0" cellspacing="0" cellpadding="0">
<tr>
<td><div align="right">Username:</div></td>
<td><input type="text" name="username" size=8 maxlength=8></td>
</tr>
<tr>
<td><div align="right">Password:</div></td>
<td><input type="password" name="password" size=8
maxlength=8"></td>
</tr>
</table>
<input class="submit" type="submit" name="Sub" value="Log In">
</form>
```

Implementing Login and Security

validateLogin.jsp

```
<%@ page language="java" import="java.io.*" %>
<%
    String authenticatedUser = null;
    session = request.getSession(true);// May create new session
    try
    {
        authenticatedUser = validateLogin(out,request,session);
    }
    catch(IOException e) { out.println(e); }
    if (authenticatedUser != null)
        response.sendRedirect("protectedPage.jsp"); // Success
    else
        response.sendRedirect("login.jsp"); // Failed login
        // Redirect back to login page with a message
%>
```

Implementing Login and Security

validateLogin.jsp (2)

```
<%!
String validateLogin(JspWriter out,HttpServletRequest request,
                     HttpSession session) throws IOException
{
    String username = request.getParameter("username");
    String password = request.getParameter("password");
    String retStr = null;
    if(username == null || password == null)
        return null;
    if((username.length() == 0) || (password.length() == 0))
        return null;
    // Should make a database connection here and check password
    // Here just hard-coding password
    if (username.equals("test") && password.equals("test"))
        retStr = username;
```

Implementing Login and Security

validateLogin.jsp (3)

```
    if(retStr != null)
    {   session.removeAttribute("loginMessage");
        session.setAttribute("authenticatedUser",username);
    }
    else
        session.setAttribute("loginMessage","Failed login.");
    return retStr;
}
%>
```

Implementing Login and Security

protectedPage.jsp

```
<html>
<head><title>Password Protected Page</title></head>
<body>

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

<%
String user = (String)session.getAttribute("authenticatedUser");
out.println("<h1>You have access to this page: "+user+"</h1>");
%>

</body>
</html>
```

Implementing Login and Security

auth.jsp

```
<%
    Object authUser = session.getAttribute("authenticatedUser");
    boolean authenticated = authUser == null ? false : true;

    if (!authenticated)
    {
        String loginMessage = "You are not authorized to "+
            "access the URL "+request.getRequestURL().toString();
        session.setAttribute("loginMessage",loginMessage);
        response.sendRedirect("login.jsp");
        return;
    }
%>
```

JSP Examples
Passing Objects Between Pages

How do you pass information between pages?

One way is to encode that information in the URL as parameters. These parameters can be extracted from the `request` object.

If you have a lot of information, it is better to use the `session` object. The session object allows you to store any number of objects that are later looked up by name. Since they remain on the server, performance/security is improved.

Passing Objects Between Pages

sendingPage.jsp

```
<%@ page import="java.util.ArrayList,java.util.Random" %>
<html><head><title>Sending Data Page</title></head><body>
<%
    // Generate and print array
    ArrayList<Integer> ar = new ArrayList<Integer>(20);
    Random generator = new Random();
    out.println("<h2>Created the following array:</h2>");
    for (int i = 0; i < 20; i++)
    {   ar.add(new Integer(generator.nextInt(10)));
        out.println(ar.get(i)+"<br>");
    }
    // Store arraylist in a session variable
    session.setAttribute("arraydata",ar);
%>
</body></html>
```

Passing Objects Between Pages

receivingPage.jsp

```
<%@ page import="java.util.ArrayList" %>
<html><head><title>Receiving Data Page</title> </head><body>

<%
    ArrayList ar = (ArrayList)session.getAttribute("arraydata");
    if (ar == null)
        out.println("<h2>No data sent to page.</h2>");
    else
    {   out.println("<h2>Received the following array:</h2>");
        for (int i = 0; i < 20; i++)
            out.println(ar.get(i)+"<br>");
    }
%>
</body>
</html>
```

Session Question

Question: **True or False:** Data associated with a session remains on the server.

A) true

B) false

Servlets

`Servlets` are Java programs that run inside a web server that can perform server-side processing such as interacting with a database or another application. `Servlets` can generate dynamic html and return this to the client.

How to create a `Servlet`:

- 1) create an HTML file that invokes a `Servlet` (usually through the `FORM ACTION=…`).
- 2) create a Java program that does the following:
  - `import javax.servlet.*;`
  - `import javax.servlet.http.*;`
  - inherit from `HttpServlet`
  - override the `doGet` and `doPost` methods
  - write the response HTML file using `java.io.Printwriter`

Servlets Notes

There is one instance of `Servlet` for each `Servlet` name, and the same instance serves all requests to that name.

Instance members persist across all requests to that name.

Local variables in `doPost` and `doGet` are unique to each request.

Servlets "Hello World!"

```
import javax.servlet.*;
import javax.servlet.http.*;
```

Import Servlet API

```
public class HelloServlet extends HttpServlet {
     public void init(ServletConfig cfg) throws ServletException {
          super.init(cfg);  // First time Servlet is invoked
     }
     public void doGet(HttpServletRequest request, HttpServletResponse response)
               throws ServletException, java.io.IOException {
        doHello(request, response);     }

     public void doPost(HttpServletRequest request, HttpServletResponse response)
               throws ServletException, java.io.IOException {
        doHello(request, response); }

     private void doHello(HttpServletRequest request,HttpServletResponse response)
               throws ServletException, java.io.IOException {
          response.setContentType("text/html");
          java.io.PrintWriter out = response.getWriter();
          out.println("<html><head><title>Hello World</title></head>");
          out.println("<body><h1>Hello World</h1></body></html>");
          out.close();
} }
```

Get & Post

Write out HTML for client

Servlets and JDBC

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;

public class JdbcServlet extends HttpServlet {

    private Connection con;

    public void init(ServletConfig cfg) throws ServletException {
        super.init(cfg);
        String url = "<fill-in>";
        con = null;
        try
        {
            con = DriverManager.getConnection(url);
        }
        catch (SQLException e)
        {   throw new ServletException("SQLException: "+e); }
    }
```

Servlets and JDBC (2)

```
public void destroy() {
    try {
        if (con != null)
            con.close();
    }
    catch (SQLException e)
    {   System.err.println("SQLException: "+e); }
}

public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, java.io.IOException {
    doTable(request, response);
}

public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, java.io.IOException {
    doTable(request, response);
}
```

Servlets and JDBC (3)

```
private void doTable(HttpServletRequest request,HttpServletResponse response)
        throws ServletException, java.io.IOException {
    response.setContentType("text/html");
    java.io.PrintWriter out = response.getWriter();
    out.println("<html><head><title></title></head>");
    if (con == null)
        out.println("<body><h1>Unable to connect to DB</h1></body></html>");
    else
    {   try {
          Statement stmt = con.createStatement();
          ResultSet rst = stmt.executeQuery("SELECT ename,salary FROM emp");
          out.print("<table><tr><th>Name</th><th>Salary</th></tr>");
          while (rst.next())
          { out.println("<tr><td>"+rst.getString(1)+"</td>"
                              +"<td>"+rst.getDouble(2)+"</td></tr>");
          }
          out.println("</table></body></html>");
          out.close();
        } catch (SQLException ex) { out.println(ex); }
    } } }
```

JSP and Servlets Discussion

Java Server Pages (JSPs) are HTML files with embedded Java code. They are easier to produce because the HTML file is the main product and the Java code is secondary.

When a JSP file is actually used, it is converted to a `Servlet` and run. Apache Tomcat handles the conversion and execution of the `Servlet`.

The advantage of JSP over `Servlets` is that the HTML page can be edited by users not familiar with the Java language using standard HTML editors and the Java code added separately by programmers.

JavaScript

JavaScript is a *scripting* language used primarily for web pages.

- JavaScript was developed in 1995 and released in the Netscape web browser (since renamed to Mozilla Firefox).

Despite the name, JavaScript is not related to Java, although its syntax is similar to other languages like C, C++, and Java.

- There are some major differences between JavaScript and Java.

From the database perspective, JavaScript is used to make HTML forms more interactive and to validate input client-side before sending it to the server.

Hello World Example - JavaScript Code

helloWorld.html

```
<html>
<head><title>HelloWorld using JavaScript</title></head>
<body>

<h1>
    <script type="text/javascript">
        document.write("Hello, world!");
    </script>
</h1>

</body>
</html>
```

<script> tag indicating code

JavaScript code

document is HTML document
document.write() puts that text into the document at this location

JavaScript and JSP

Your JSP code can either include JavaScript code:

- 1) Directly in the HTML code
- 2) By outputting it using out.println()
  - With servlets, you only have option #2.

Remember, the JSP/Servlet code is run on the server, and the JavaScript code is run on the client.

- The JavaScript code cannot access the database directly.
- The JSP/Servlet code cannot interact with the user (unless you use AJAX or some other method to send server requests).

Hello World written by JSP Code

helloWorld_JS.jsp

```
<html>
<head><title>HelloWorld using JavaScript</title></head>
<body>

<h1>
    <%
        out.println("<script>");
        out.println("document.write(\"Hello, world!\")");
        out.println("</script>");
    %>
</h1>

</body>
</html>
```

JavaScript for Data Validation

JavaScript is commonly used to validate form input on the client-side without sending it to the server. This reduces the load on the server, and more importantly, improves the browsing experience for the user.

JavaScript Data Validation Example

validateNum.html

```
<html><head><title>Number Field Validation</title></head><body>
<script>
function validate(num){
    if (parseInt(num))
        alert("You entered number: " + parseInt(num));
    else
        alert("Invalid number entered: " + num);
}
</script>

Enter a number:
<form name="input"> <input type="text" name="numField"
                           onchange="validate(this.value)">
</form>
</body></html>
```

Validation code

Validate when field is changed. Can also use:
- `onblur` – triggered when field loses focus
- `onsubmit` – triggered when form is submitted

AJAX

AJAX allows client-side JavaScript to request and receive data from the server without refreshing the page.

AJAX (Asynchronous JavaScript+XML) was named by Jesse Garret in 2005. However, XML is not required for data communication between client and server (JSON is more common).

AJAX uses the **`XMLHttpRequest Object`** to communicate requests and receive results from the server.

Communication can either be synchronous or asynchronous.

AJAX Validate Form Fields Example

validateUserEmail.html

```
<html><head><title>Validate User Name and Email</title></head><body>
<script>
function checkUserName(){
     var name = document.getElementById("username").value;
     var xmlhttp = new XMLHttpRequest();
     xmlhttp.open("GET","validateUserEmail.jsp?name="+name,false);
     xmlhttp.send(null); // Synchronous version, no additional payload
     if (xmlhttp.responseText != "VALID")
          alert(xmlhttp.responseText);
}
function checkEmail(){
     var email = document.getElementById("email").value;
     var xmlhttp = new XMLHttpRequest();
     xmlhttp.open("GET","validateUserEmail.jsp?email="+email,false);
     xmlhttp.send(null); // Synchronous version, no additional payload
     if (xmlhttp.responseText != "VALID")
          alert(xmlhttp.responseText);
}
</script>
<form name="input">
     Name: <input type="text" id="username">
          <a href="javascript: checkUserName()">Check</a> <br>
     Email: <input type="text" id="email">
          <a href="javascript: checkEmail()">Check</a>
</form></body></html>
```

AJAX Validate Form Fields Example

validateUserEmail.jsp

```
<%  String name = request.getParameter("name");
    if (name != null)
    {   // This is simple validation code.
        if (name.length() < 4)
            out.println("Name too short. Must be at least 4 chars.");
        else
            out.println("VALID");
    }
    else
    {
        String email = request.getParameter("email");
        if (email != null)
        {   if (!email.contains("@"))
                out.println("INVALID");
            else
                out.println("VALID");
        }
        else
            out.println("INVALID INPUT");
    }
%>
```

AJAX Example - Province/State

provinceState.html

Canada Version

US Version


```
<html><head><title>Province/State</title></head>
<body bgcolor="white">
<form name="input">
  Country: <select id="country" onchange="changeCountry()">
       <option value="CA">Canada</option>
       <option value="US">United States</option>
    </select><br>
 <p id="stateText">State:</p><select id="state"></select>
</form>
```

Province/State for Canada/US

provinceState.html (2)

```
<script>var xmlhttp = new XMLHttpRequest();
function changeCountry_callBack()
{    if (xmlhttp.readyState == 4 && xmlhttp.status == 200)
     {    var text = xmlhttp.responseText;
          if (text != "INVALID")
          {    var stateList = document.getElementById("state");
               // Remove all current items in list
               for (var i = stateList.length-1; i >= 0; i--)
                 stateList.remove(i);
               // Add elements to list
               var values = text.split(',');
               for (var i=0; i < values.length; i++)
               { var stateCode = values[i];
                 var newOpt = document.createElement('option');
                 newOpt.text = stateCode;
                 newOpt.value = stateCode;
                 stateList.add(newOpt, null);
               }
          }
     }
}
```

Province/State for Canada/US

provinceState.html (3)

```
function changeCountry(){
    var countryList = document.getElementById("country");
    var stateText = document.getElementById("stateText");

    if (countryList.selectedIndex == 1)
        stateText.innerHTML = "State: ";
    else
        stateText.innerHTML = "Province:";

    xmlhttp.open("GET","provinceState.jsp?country="+countryList.value,true);
    xmlhttp.onreadystatechange = changeCountry_callBack;
    xmlhttp.send(null);   // Asynchronous version
}

changeCountry();      // Initialize list with default as Canada
</script>
</body>
</html>
```

Province/State for Canada/US – JSP

provinceState.jsp

```
<%@ page import="java.sql.*" %> <%
String country = request.getParameter("country");
if (country == null)
    out.println("INVALID");
else
{   try
    ( Connection con = DriverManager.getConnection(url,uid,pw); )
    {   String sql = "SELECT stateCode FROM states where countrycode=?";
        PreparedStatement pstmt = con.prepareStatement(sql);
        pstmt.setString(1, country);
        ResultSet rst = pstmt.executeQuery();
        StringBuffer buf = new StringBuffer();
        while (rst.next()) {
            buf.append(rst.getString(1));
            buf.append(',');
        }
      if (buf.length() > 0)
         buf.setLength(buf.length()-1);
      out.println(buf.toString());
    }
    catch (SQLException ex){  out.println("INVALID"); }
    }
%>
```

Connection Pools

A ***connection pool*** is a group of database connections managed by a (web) server.

- All connections in the pool are open. Clients request a connection from the pool and return it when done.
- This results in improved performance as connections are shared across clients and do not pay an open/close penalty every time a connection is used.

Using a connection pool in Tomcat with JNDI:

```
Context root = new InitialContext();                    // Get root of JNDI tree
String path = "java:comp/env/jdbc/workson";             // Name key
DataSource ds = (DataSource) root.lookup(path);         // JNDI lookup
con = ds.getConnection();                               // Get connection from pool
...
con.close();                                            // Return connection to pool
```

Connection Pools Question

Question: **True or False:** A connection pool will speed up the execution time of a query (not considering connection time).

A) true

B) false

Configuring your Web Application

Each web application (*webapp*) has its own directory that stores HTML/JSP files and has a subdirectory `WEB-INF` that contains:

- `classes` directory – stores all Java class files
- `lib` directory – store all jars used by your Servlets
- `web.xml` – is a ***deployment descriptor*** that provides mapping from URL path to Servlet class name. Example:

```
<web-app>
    <servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>HelloServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
        <url-pattern>/servlet/HelloServlet</url-pattern>
    </servlet-mapping>
</web-app>
```

Internal name used in mapping (you decide)

Servlet Java class name

Internal name (same as line #1)

URL to get to Servlet (from user's perspective)

PHP

PHP (www.php.net) is a general-purpose scripting language used extensively in web development.

PHP supports several different ways of connecting to databases including a custom MySQL connector as well as support for ODBC and PHP Data Objects (PDO).

Unlike JDBC, each database has its own database extension which has different features and methods.

PHP MySQLi Example
Procedural Version

```
<?php
// Connecting, selecting database
$mysqli = mysqli_connect("cosc304.ok.ubc.ca", "rlawrenc",
                         "<pw>", "db_rlawrenc")
          or die("Could not connect: " . mysql_error());

// Performing SQL query
$query = "SELECT * FROM emp";
$result = mysqli_query($mysqli, $query);
if (mysqli_connect_errno($mysqli))
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
```

PHP MySQLi Example Procedural Version (2)

```
// Printing results in HTML
echo "<table>\n";
while ($line = mysqli_fetch_assoc($result)) {
    echo "\t<tr>\n";
    foreach ($line as $col_value) {
        echo "\t\t<td>$col_value</td>\n";
    }
    echo "\t</tr>\n";
}
echo "</table>\n";
// Free resultset
mysqli_free_result($result);
// Closing connection
mysqli_close($mysqli);
?>
```

PHP MySQLi Example
Object-Oriented Version

```
<?php
$mysqli = new mysqli("cosc304.ok.ubc.ca", "<user>", "<pw>", "workson");
if ($mysqli->connect_errno)
  echo "Failed to connect to MySQL: " . $mysqli->connect_error;
$result = $mysqli->query($query);
echo "<table>\n";
while ($line = $result->fetch_assoc()) {
    echo "\t<tr>\n";
    foreach ($line as $col_value) {
        echo "\t\t<td>$col_value</td>\n";
    }
    echo "\t</tr>\n";
}
echo "</table>\n";
$result->free();
$mysqli->close();
?>
```

Conclusion

JSP and `Servlets` can use JDBC for constructing dynamic web content and answering user queries via HTML.

JSP and `Servlets` are used with HTML forms to allows user to enter queries or information to the database.

- JSP/ `Servlets` can be run using Apache Tomcat which supports connection pooling.

JavaScript is used for browser-based validation and interactivity. AJAX supports requests to server for data.

Connecting and querying a database with PHP (and other languages) is very similar to using Java/JDBC.

Objectives

- Write a simple JSP page that uses JDBC.
- Explain the relationship between JSP and `Servlets` and how dynamic web pages are created.
- Create client-side code in JavaScript for data validation.
- Explain the general idea with AJAX.
- Explain what a connection pool is and why it is beneficial.
- Write database access code using PHP.


UBC
THE UNIVERSITY OF BRITISH COLUMBIA