Web Databases – Part 1.2
Web Database Programming
CS 5513

Contents

- Web interfaces to databases
- Servlets
- Java Server Pages (JSPs)
- Object-Relational model and JSPs example
- About using Java inside JSPs

Web Interfaces to Databases

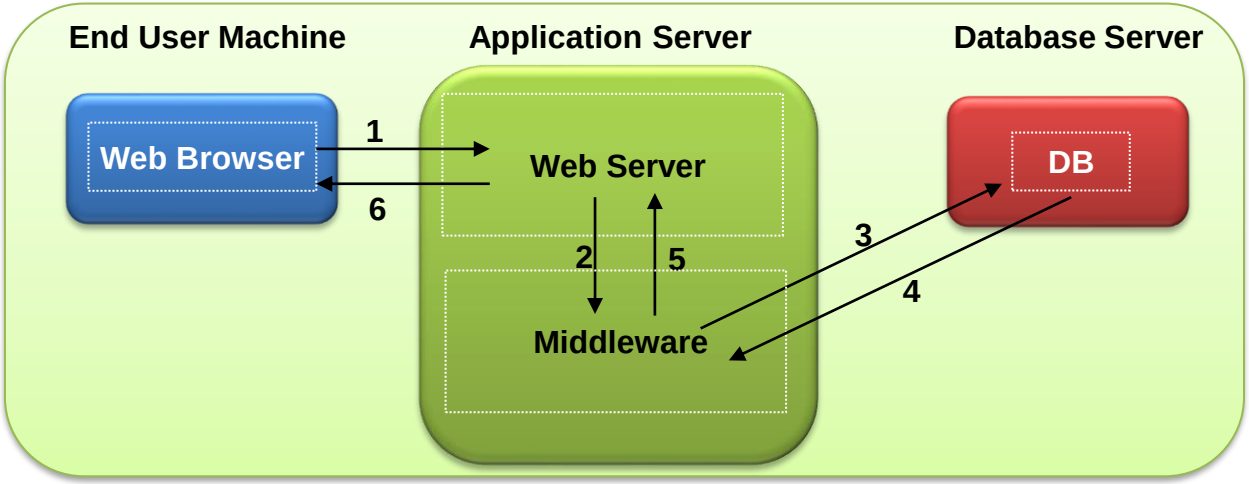
- Are the predominant interfaces for databases
- Instead of having the user type a query in SQL:

```
SELECT Employee.id, Employee.salary
FROM Employee E, Department D
WHERE E.dept = 120, E.salary > 6000
```

- The user can simply type the data in an html form:

Web Interfaces to Databases (Continued)

1. In a web browser, an end user submits a request for some dynamic data to the web server.
2. The web server passes it onto the middleware (for example Java Servlets, JSP).
3. The middleware writes the request in SQL queries and sends them to a back-end database using some API such as JDBC, ODBC.
4. The retrieved data are sent back to the middleware.
5. The middleware generates a web page for the data.
6. The web server sends the web page to the browser of the end user.



A generic web architecture

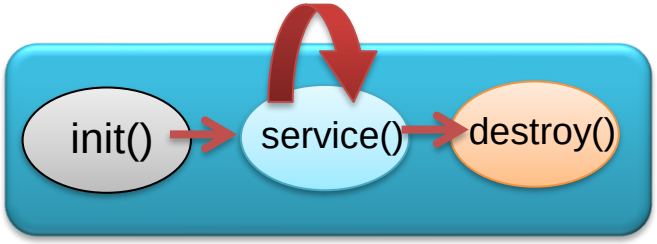
Servlets

- According to the Java EE API:

A servlet is a small program that runs within a web server. Servlets receive and respond to requests from web clients, usually across HTTP.

- Advantages:
  - Servlets can be loaded into memory when the web server starts; hence, each request is handled by a Java thread.
  - If N requests to the same Program come up, there will be N threads, but only a single Servlet class processes the requests.
  - Optimization techniques like caching previous computations and having database connections open are some of the advantages

Servlets (Cont)



Lifetime of a Servlet

```
@Override
protected void
doGet(HTTPServletRequest request,
              HTTPServletResponse response)

@Override
protected void
doPost(HTTPServletRequest request,
              HTTPServletResponse response)
```

Methods that an instance of Servlet must override

- A user enters a URL into a web browser. The browser generates an HTTP request for this URL and sends it to the appropriate server.
- The server invokes the init() method of the servlet. This happens only when the servlet is first loaded into memory
- The server invokes the servlet's service() method. The servlet then processes the HTTP request
- It remains in the server's address space and is available to process other HTTP requests received from clients.
- When the server decides to unload the servlet from the memory, it calls the destroy() method.

Servlets (Cont)

- The user types a URL: http://www.test101.com/doc/test.html , and the users web server generates an HTTP request:

```
GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35

bookId=12345&author=Tan+Ah+Teck
```

Request Line

Request Headers

Request Message Header

A blank line separates header & body

Request Message Body

The request is mapped into the HTTPServletRequest object

```
@Override
protected void
doGet(HTTPServletRequest request,
                HTTPServletResponse response)
{
   String userAgent =
                request.getHeader("User-Agent");
}
```

Example of how to retrieve a header

- The HTTP request is received by the web server. The server maps this request to a particular servlet.

Left figure from https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

Servlets (Cont)

```
@Override
protected void
doGet(HTTPServletRequest request,
            HTTPServletResponse response)

@Override
protected void
doPost(HTTPServletRequest request,
            HTTPServletResponse response)
```

Methods an instance of servlet must override

- If the HTTP request is a Get, then the doGet of the corresponding Servlet is run.
- If the HTTP request is a Post, then the doPost of the corresponding Servlet is run.

Example Servlet Code

```
import java.io.IOException;
import java.io.PrintWriter;


import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;



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

   public PersonQueryServlet() {
      super();
   }


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

      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<HEAD><TITLE>My First Servlet</TITLE></HEAD>");
      out.println("<BODY>");
      out.println("<H1>Hello, World!</H1>");
      out.println("</BODY>");
      out.close();
   }
}
```

Servlet Sessions

- Servlet API supports handling of sessions
  - Sets a cookie on first interaction with browser, and uses it to identify session on further interactions
- To check if session is already active:
  - if (request.getSession(false) == true)
    - .. then existing session
    - else .. redirect to authentication page
  - authentication page
    - check login/password
    - request.getSession(true): creates new session
- Store/retrieve attribute value pairs for a particular session
  - session.setAttribute(“userid”, userid)
  - session.getAttribute(“userid”)

Servlets and JDBC

- Suppose we have the table:

SSN	Name
000-75-1234	Le Gruenwald

```
create table Faculty(
      ssn varchar(15) not null,
      name varchar(25),
       primary key(ssn));
```

```
public final class Query extends HttpServlet {
 protected void doPost(HttpServletRequest request,
                       HttpServletResponse response)
                       throws ServletException, IOException
 {
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head><title>Query2</title></head>");
    out.println("<body>");

   /* Get the parameters from the form data  */
   String ssn          = request.getParameter(“SSN”);
   String name         = request.getParameter(“name");
   String password = request.getParameter("password");
    try {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        Connection conn = DriverManager.getConnection(url, login,
                 password);
        Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(“INSERT INTO FACULTY
                 values(ssn, name));
    }
     catch (SQLException sqle) {}   out.close();} }
```

Java Server Pages (JSP)

- The idea is that Java is “injected” into HTML code.
- The JSP code is then compiled into Java (servlets) and HTML code.
- Although JSP and Java servlet can finish the same task, it is more convenient to write and modify regular HTML than to have a zillion println statements that generate the HTML.
- By separating the look from the content, a big project can be done by different people in a perfect way. That is, web page design experts can build the HTML, and leave rooms for Java servlet specialists to insert the dynamic content.
- JSP reaps all the benefits provided by Java servlets and web container environment, but they have an added advantage of being simpler and more natural program for web enabling enterprise developer.

Java Server Pages (JSP)

- Inside a JSP there are certain variables that are *implicitly* defined. Some of them are:
  - request, response, session, out, config…
- The request variable corresponds to the servlet's HTTPServletRequest, and the response corresponds to HTTPServletResponse.

```
<html>
<head> <title> Hello
</title> </head>
<body>
<h1>Hello
<%=
request.getParameter("na
me") }
%>
<% out.println("hello");
%>
</h1>
</body>
</html>
```

JSP

Java Server Pages (JSP)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class PersonQueryServlet extends
      HttpServlet {
  public void doGet (HttpServletRequest
      request, HttpServletResponse response)
                  throws ServletException,
      IOException
  {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HEAD><TITLE>My First
                Servlet</TITLE></HEAD>");
        out.println("<BODY>");
        if(request.getParameter("name") ==
      null) {
                out.println("Hello, World!");
        }
        else {
                out.println("Hello " +
                request.getParameter("name"));
         }
        out.println("</BODY>");
       out.close();
  }}
```

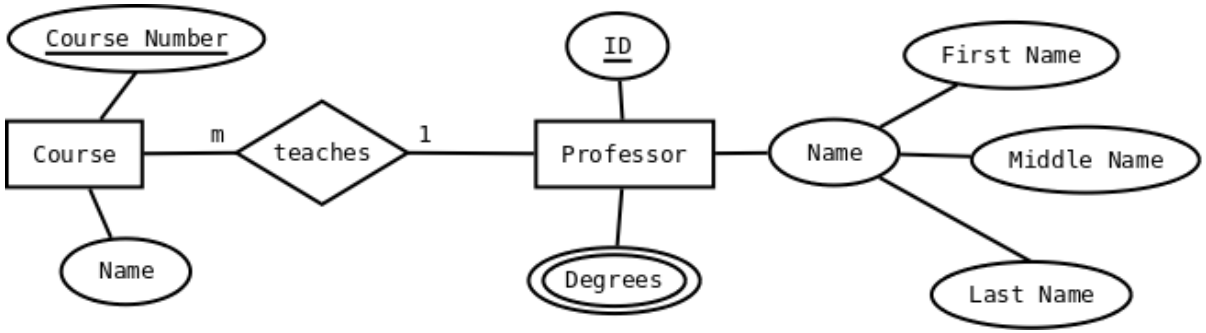
Java Servlets

```
<html>
<head> <title> Hello </title>
</head>
<body>
<% if
(request.getParameter("name")
        == null) {
 %>
<h1>Hello World</h1>
<% } else {
%>
<h1>Hello
<% request.getParameter("name"));
}
%>
</h1>
</body>
</html>
```

JSP

Object-Relational Model and JSPs Example

- In this example, we'll use JSP to create a web interface to an object-relational database that is modeled with the following ER diagram



- The queries that we want to implement are:
  - Insert a professor
  - Insert a course
  - Find professors with a given degree
- We want to use Oracle's Object-Relational model

Object-Relational Model and JSPs Example(ii)

- In this example we'll present the following:
  - A **database creation file** that creates types and tables using Oracle's object-relational model. This is contained in the file `create_tables.sql`
  - A **java class** implementing the database queries to insert a student, insert an instructor and find a professor with a given degree. This class is contained in the file `DataHandler.java`
  - The **JSP files,** `insert_professor_form.jsp, insert_course_form.jsp, and find_professor_form.jsp`, which contain the forms in which the user can input the parameters to the queries online.
  - The **JSP files**, `insert_professor.jsp, insert_course.jsp, and find_professor.jsp`, which connect the forms with the database, and act as controllers.

Object-Relational Model and JSPs Example (iii)

- **Database creation file**

```
-----------------------------------------------------------------
-- Filename: create_tables.sql
--  This is a class example of how to implement a simple
--  database using the object-relational model of Oracle.
-----------------------------------------------------------------
-- Define the name type
CREATE TYPE name_obj AS OBJECT (
     first_name  VARCHAR2(32),
     middle_name VARCHAR2(32),
     last_name   VARCHAR2(32)
) FINAL;
/
-- Define the degrees type
CREATE TYPE degrees_obj AS VARRAY(10) OF
VARCHAR2(64);
/
-- Define the professor type
CREATE TYPE professor_obj AS OBJECT (
      id NUMBER(5),
      name name_obj,
      degrees degrees_obj
) FINAL;
/
-- Define the course type
CREATE TYPE course_obj AS OBJECT (
     course_number NUMBER(5),
     course_name VARCHAR2(64),
     professor REF professor_obj
) FINAL;
/

-----------------------------------------------------------------
-- Now creating the tables
-----------------------------------------------------------------

-- Create a table for professors
CREATE TABLE professor_tab OF professor_obj
(id PRIMARY KEY)
OBJECT IDENTIFIER IS PRIMARY KEY;
/
-- Create a table for courses
CREATE TABLE course_tab OF course_obj
(course_number PRIMARY KEY,
FOREIGN KEY (professor) REFERENCES professor_tab ON DELETE SET
NULL)
OBJECT IDENTIFIER IS PRIMARY KEY;
```

Object-Relational Model and JSPs Example (iv)

- **Java Class implementing database queries**

```
package jsp_oracle_test;

import java.sql.*;
import oracle.jdbc.pool.OracleDataSource;

/**
 * This class is in charge of communicating with the Oracle Database
 * to perform the queries to insert a professor, to insert a course,
 * and to retrieve the professors on their degrees.
 */
public class DataHandler {
   String jdbcUrl  = "jdbc:oracle:thin:@//oracle18.cs.ou.edu:1521/orclpdb";
   String userid   = ”user"; //your Oracle username
   // For security reasons, we suggest that you DON'T PUT YOUR PASSWORD
   //  IN YOUR SOURCE FILE LIKE THIS:
   String password = “pwd"; // your Oracle password
   Connection conn;

   /**
    * This class gets the current DB connection. This is not to be used in
    * production environments. You should use a connection pool instead.
    *
    * @return
    * @throws SQLException
    */
   public Connection getDBConnection() throws SQLException{
      final OracleDataSource ds = new OracleDataSource();
      ds.setURL(jdbcUrl);

      if(conn == null) {
         conn = ds.getConnection(userid, password);
         System.out.print("Connection made");
      }

      conn.setAutoCommit(true);
      return conn;
   }

   /**
    * Query 1.- Insert a professor given his/her attributes
    *
    * @param id
    * @param firstName
    * @param middleName
    * @param lastName
    * @param degrees   The degrees of the professor
    * @return
    * @throws SQLException
    */
public boolean insertProfessor(String id, String firstName,
      String middleName, String lastName, String[] degrees)
   throws SQLException {
   // Connect to the database.
                        System.out.print("Connecting");
   getDBConnection();
   int updatedRows  = 0;

   /**
    * Here we create a string with '?' placeholders.
    */
   String degreesStr = "degrees_obj(";
   for(int i = 0; i < degrees.length-1; i++) {
      degreesStr +=  "?, ";
   }

   if(degrees.length > 0) {
      degreesStr += "?";
   }
   degreesStr += ")";

   // Elaborate a string with the content of the insertion query
   final String sqlInsertProfessor =
   "INSERT INTO professor_tab VALUES(?, name_obj(?,?,?), " +
   degreesStr + ")";

   try (
      PreparedStatement stmt = conn.prepareStatement(sqlInsertProfessor);
   ){
      stmt.setString(1, id);
      stmt.setString(2, firstName);
      stmt.setString(3, middleName);
      stmt.setString(4, lastName);

      // Now set the degrees
      for(int i=0; i < degrees.length; i++) {
         stmt.setString(5+i, degrees[i]);
      }

      // Run the query
      updatedRows  = stmt.executeUpdate();
      System.out.print("Insert new professor");
   } catch(SQLException e) {
      e.printStackTrace();
   }

   return updatedRows != 0;
}
```

Note: This class continues in the next slide

Object-Relational Model and JSPs Example (v)

- **Java Class implementing database queries (ii)**

```
//**
 * Query 2: Inserts a course given its attributes
 * @param courseNumber
 * @param name The name of the course
 * @param instrID The id of the instructor that teaches the course
 * @return
 * @throws SQLException
 */
public boolean insertCourse(int courseNumber, String name, int profID)
   throws SQLException{

   int updatedRows  = 0;
   String sqlInsertCourse =
                        "INSERT INTO course_tab "
                        + "SELECT ?, ?,  REF(prof) "
                        + "FROM professor_tab prof "
                        + "WHERE prof.id = ?";

   // Connect to the database
   getDBConnection();

   // Prepare the SQL query
   try(PreparedStatement stmt = conn.prepareStatement(sqlInsertCourse);){
                           stmt.setInt(1, courseNumber);
                           stmt.setString(2, name);
                           stmt.setInt(3, profID);

                           // Run the query
                           updatedRows  = stmt.executeUpdate();
                        } catch(SQLException e) {
                        e.printStackTrace();
   }
   return updatedRows != 0;
}

/**
 * Query 3: Retrieves professors that have a given degree
 *
 * @param degree
 * @return
 * @throws SQLException
 */
public ResultSet findProfessors(String degree)
   throws SQLException {

   // Connect to the database.
   getDBConnection();

   String sql =
                        "SELECT P.id AS id, "    +
                        "   P.name.first_name AS first_name, "   +
                        "   P.name.middle_name AS middle_name, "   +
                        "   P.name.last_name AS last_name, "   +
                        "   P.degrees  AS degrees " +
                        "FROM professor_tab P " +
                        "WHERE ? IN (SELECT * FROM table(P.degrees))";

   // Prepare the SQL query.
   PreparedStatement stmt = conn.prepareStatement(sql);
   stmt.setString(1, degree);

   // Run the query
   ResultSet result = stmt.executeQuery();
   return result;
}
}
```

Note: This is a continuation of the code in the previous slide, so it also belongs to DataHandler.java

Object-Relational Model and JSPs Example (viii)

- **JSP file insert_professor_form.jsp** that implements the form to insert an instructor online

```
<!– Filename: insert_professor_form.jsp-->
<!– This jsp file contains the code to display a form to input the parameters for a new professor --
>

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Insert a Professor</title>
  </head>
  <body>
    <h2>Insert a professor</h2>
    <form action="insert_professor.jsp">
      <table border=1>
        <tr>
          <th colspan="2">Enter the professor data:</th>
        </tr>
        <tr>
          <td>ID:</td>
          <td><div style="text-align: center;">
              <input type=text name=professor_id>
          </div></td>
        </tr>
        <tr>
          <td>First Name:</td>
          <td><div style="text-align: center;">
              <input type=text name=professor_firstname>
          </div></td>
        </tr>
        <tr>
          <td>Middle Name:</td>
          <td><div style="text-align: center;">
              <input type=text name=professor_middlename>
          </div></td>
        </tr>
        <tr>
          <td>Last Name:</td>
          <td><div style="text-align: center;">
              <input type=text name=professor_lastname>
          </div></td>
        </tr>
        <tr>
          <td>Degrees (Comma separated):</td>
          <td><div style="text-align: center;">
              <input type=text name=professor_degrees>
          </div></td>
        </tr>
        <tr>
          <td><div style="text-align: center;">
              <input type=reset value=Clear>
          </div></td>
          <td><div style="text-align: center;">
              <input type=submit value=Insert>
          </div></td>
        </tr>
      </table>
    </form>
  </body>
</html>
```

Object-Relational Model and JSPs Example (ix)

- **JSP file insert_course_form.jsp** that implements the form to insert a course online

```
<!– Filename: insert_course_form.jsp-->
<!– This jsp file contains the code to display a form to input the parameters for a course -->

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Insert Course</title>
  </head>
  <body>
    <h2>Insert a course</h2>
    <form action="insert_course.jsp">
      <table border=1>
        <tr>
          <th colspan="2">Enter the course data:</th>
        </tr>
        <tr>
          <td>Course number:</td>
          <td><div style="text-align: center;">
              <input type=text name=course_number>
          </div></td>
        </tr>
        <tr>
          <td>Course Name:</td>
          <td><div style="text-align: center;">
              <input type=text name=course_name>
          </div></td>
        </tr>
        <tr>
          <td>Professor ID:</td>
          <td><div style="text-align: center;">
              <input type=text name=course_professor_id>
          </div></td>
        </tr>
        <tr>
          <td><div style="text-align: center;">
              <input type=reset value=Clear>
          </div></td>
          <td><div style="text-align: center;">
              <input type=submit value=Insert>
          </div></td>
        </tr>
      </table>
    </form>
  </body>
</html>
```

Object-Relational Model and JSPs Example (x)

- **JSP file query_professor_form.jsp** that implements the form to find professors with a given degree online

```
<!– Filename: query_professor_form.jsp-->
<!– This jsp file contains the code to obtain the professors with a given degree -->


<!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>Query.- Input professor degree</title>
  </head>
  <body>
    <center>
      <!-- This page is designed to find professors with a given degree -->
      <h2> Query.- Find professor with a given degree</h2>
      <form action="query_professor.jsp">
        <table border=1>
          <tr>
            <th colspan="2">Enter the degree you want to search for:</th>
          </tr>
          <tr>
            <td>Degree:</td>
            <td><div style="text-align: center;"><input type=text name=degree></div></td>
          </tr>
          <tr>
            <td><div style="text-align: center;">
                <input type=reset value=Clear>
            </div></td>
            <td><div style="text-align: center;">
                <input type=submit value=Search>
            </div></td>
          </tr>
        </table>
      </form>
    </center>
  </body>
</html>
```

Object-Relational Model and JSPs Example (xii)

- **JSP file insert_professor.jsp** that calls the database handler to insert an instructor

```
<<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!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>Query result</title>
  </head>
  <body>
    <center>
      <%@page import="cs5513.DataHandler"%><%@page
                                  import="java.sql.ResultSet"%><%@page
                                                import="java.sql.Array"%>

      <%
        // The handler is the one in charge of establishing the connection.
        DataHandler handler = new DataHandler();

        // Get the parameters from the form.
        String idStr      = request.getParameter("professor_id");
        String firstName  = request.getParameter("professor_firstname");
        String middleName = request.getParameter("professor_middlename");
        String lastName   = request.getParameter("professor_lastname");
        String[] degrees  = request.getParameter("professor_degrees").split(",");

        /*
        * If the user hasn't filled out all the fields. This is very
        * simple checking
        */
        if (idStr.equals("") || firstName.equals("") || middleName.equals("")
        || lastName.equals("")) {
        response.sendRedirect("insert_professor_form.jsp");
        } else {
        // Now perform the query with the data from the form.
        boolean success = handler.insertProfessor(idStr, firstName, middleName,
        lastName, degrees);

        if (!success) {
        %>
      <h2>There was a problem inserting the professor</h2>
      <%
        } else {
        %>
      <h2>The professor:</h2>
      <ol>
        <li>ID: <%=idStr%></li>
        <li>First Name: <%=firstName%></li>
        <li>Middle Name: <%=middleName%></li>
        <li>Last Name: <%=lastName%></li>
      </ol>
      <h2>was inserted.</h2>
      <%
        }
        }
        %>
    </center>
  </body>
</html>
```

Object-Relational Model and JSPs Example (xi)

- **JSP file insert_course.jsp** that calls the database handler to insert a course

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!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>Query result</title>
  </head>
  <body>
    <center>
      <%@page import="cs5513.DataHandler"%><%@page

                                  import="java.sql.ResultSet"%><%@page


                                  import="java.sql.Array"%>

      <%
        // The handler is the one in charge of establishing the connection.
        DataHandler handler = new DataHandler();

        // Get the parameters from the form.
        String courseNumStr = request.getParameter("course_number");
        String courseName   = request.getParameter("course_name");
        String profIDStr    = request.getParameter("course_professor_id");
        /*
        * If the user hasn't filled out all the fields. This is very
        * simple checking
        */
        if (courseNumStr.equals("") || courseName.equals("") || profIDStr.equals("")) {
        response.sendRedirect("insert_course_form.jsp");
        } else {
        int courseNum = Integer.parseInt(courseNumStr);
        int profID    = Integer.parseInt(profIDStr);
        // Now perform the query with the data from the form.
        boolean success = handler.insertCourse(courseNum, courseName, profID);

        if (!success) {
        %>
      <h2>There was a problem inserting the course</h2>
      <%
        } else {
        %>
      <h2>The course:</h2>
      <ol>
        <li>Course number: <%=courseNumStr%></li>
        <li>Course Name: <%=courseName%></li>
      </ol>
      <h2>was inserted.</h2>
      <%
        }
        }
        %>
    </center>
  </body>
</html>
```

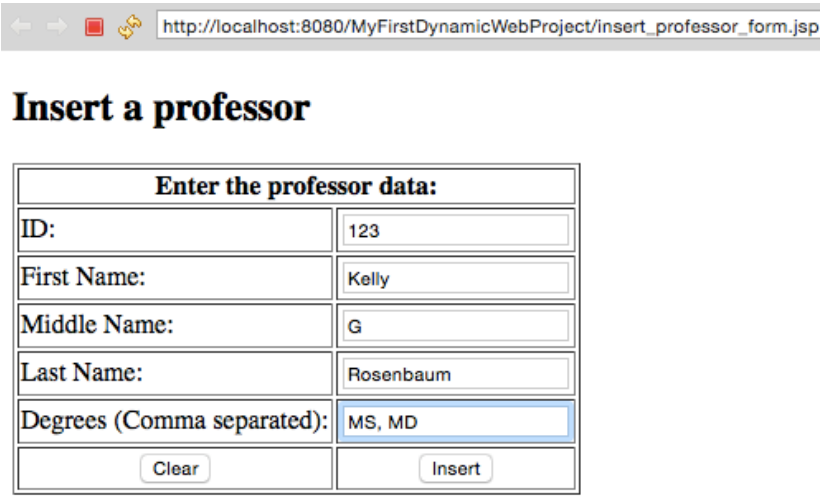
Object-Relational Model and JSPs Example (xiii)

- **JSP file query_professor.jsp** that calls the database handler to find professors

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!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>Query.- Find professors with a given degree</title>
  </head>
  <body>
    <center>
      <%@page import="cs5513.DataHandler"%>
      <%@page import="java.sql.ResultSet"%>
      <%@page import="java.sql.Array"%>
      <%@page import="java.util.Set" %>
      <%

        // The handler is the one in charge of establishing the connection.
        DataHandler handler = new DataHandler();

        // Get the parameters from the form.
        // Remember to validate the parameters!!
        String degree = request.getParameter("degree");

        // Now perform the query with the data from the form.
        ResultSet professors = handler.findProfessors(degree);
        %>
      <table cellspacing="2" cellpadding="2" border="1">
        <tr>
          <td align="center">
            <h4>PROFESSOR ID</h4>
          </td>
          <td align="center">
            <h4>FIRST NAME</h4>
          </td>
          <td align="center">
            <h4>MIDDLE NAME</h4>
          </td>
          <td align="center">
            <h4>LAST NAME</h4>
          </td>
        </tr>
        <%
          while(professors.next()) {
          // Print the degrees as a comma, separated.
          String idStr      = professors.getString("id");
          String firstName  = professors.getString("first_name");
          String middleName = professors.getString("middle_name");
          String lastName   = professors.getString("last_name");
          // Print each attribute
          out.println("<tr>");
          out.println("<td align=\"center\">" + idStr
          + "</td><td align=\"center\"> "
          + firstName
          + "</td><td align=\"center\"> "
          + middleName
          + "</td><td align=\"center\"> " +
          lastName );
          out.println("</tr>");
          }
          %>
      </table>
    </center>
  </body>
</html>
```

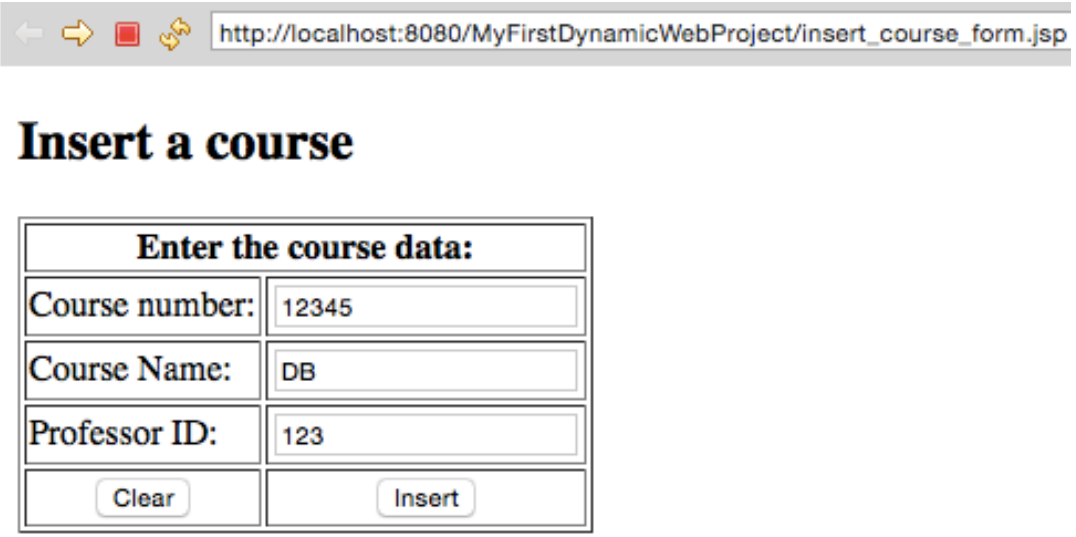
Object-Relational Model and JSPs Example (xiv)

- Insert_professor.jsp form used to provide the attributes of a new professor



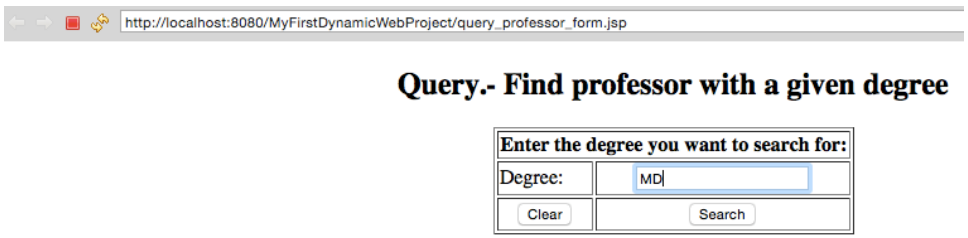
Object-Relational Model and JSPs Example (xv)

- Insert_course.jsp form used to provide the attributes of a course.



Object-Relational Model and JSPs Example (xvi)

- Query_professor.jsp form used to find a professors with a given degree



Result of searching the professors with an “MD”