WEB-BASED DATABASE ACCESS: A PEDAGOGICAL STRATEGY

Allan M. Hart and James M. Slack
Department of Computer and Information Sciences
Minnesota State University
Mankato, MN 56001
allan.hart@mnsu.edu, james.slack@mnsu.edu

Abstract

Database textbooks published within the last 5 years or so almost always include a chapter or two on web-based database access. While this is not surprising given the obvious importance of the internet, it presents unique challenges to those instructors who wish to cover this material. First, since it is still necessary to cover traditional database topics such as ER and EER diagramming, the relational model, SQL, etc., only a limited amount of time can be devoted to the topic of web-based access. Second, how does an instructor teach students how to *implement* a database with web-based access given that doing so requires some sort of "server side" programming? Not all server side programming technologies provide a good "fit" for departments with limited hardware and personnel resources.

In this paper we outline a strategy for meeting these challenges based on JSP and the recently released version 1.1 of the Java Standard Tag Library (JSTL).

1. Introduction

A brief look at recently published database texts reveals a trend toward including a web-based database component in introductory database classes. E.g., [1] includes a chapter entitled "The Internet Database Environment", [2] includes a chapter entitled "Web Technology and DBMSs", [3]'s chapter 6, entitled "Database Application Development" includes material on JDBC and SQLJ, [4]'s chapter 14 is entitled "Web Database Development" , [5] has several chapters devoted to web-based database access, and [6]'s Part V contains three chapters devoted to web-based access. Even the venerated [7] and [8] now include material devoted to the topic. While none of this should come as a surprise to anyone who has witnessed the explosive growth and importance of the internet, an instructor who wishes to follow this trend in the classroom faces a number of challenges.

2. Challenges

There are several challenges facing the instructor who wishes to include a web-based database access component in his/her introductory database class.

1. Because it is still necessary to cover the more traditional topics such as semantic modeling (ER and EER diagrams), the relational model, normalization, the relational algebra, SQL, etc., only a limited amount of time can be devoted to web-based database access. How do you cover a topic as broad as web-based access in a few short weeks?
2. Teaching students how to implement a database application that includes web-based access requires some sort of server side programming on the part of the students. Hence, some sort of server side programming technology must be chosen. Should it be ASP, Servlets, JSP, PHP, Perl or something else? While there are many choices available, not all are equally available. During these times of lean budgets, not all departments have the hardware, software or personnel resources necessary to allow students to engage in server side programming experiments with an arbitrary server side programming technology.
3. Once a particular server side technology has been chosen, a web server of some kind must be chosen. While it's possible to require students to install, maintain, and use a web server (Tomcat e.g.) on their own machines, precious time can be wasted by the instructor who has to explain to his/her students how to do just that.

3. Solutions

In this paper we outline a JSP-based strategy for meeting the challenges noted above. We have followed this methodology in our introductory database class and have found it to be a viable approach to incorporating a web-based database access component in that course. The strategy utilizes the recently released version 1.1 of the Java Standard Tag Library (JSTL). The JSTL includes a number of facilities for database access and provides an environment in which a student who has perhaps less than optimal programming skills can still implement a web-based database application with relative ease.

4. Scenario

At Minnesota State University, Mankato, our introductory database class consists of a mix of CS, CIS, and MIS majors with a dose of Business and Management majors thrown into the mix for good measure. While most of these students have had some experience with programming, the average student's experience often consists of a single CS1 course in either Python or Java. Typically, we split the class into teams of four to six students. Each team views itself as a database consulting company. This company has been approached by some business that wants the company to construct an Oracle database that includes an online shopping cart facility. Each team is thus required to implement a database application that allows for web-based access. Because the application involves the selling of some product or products, the database constructed by a particular team includes the usual Customer, Product, Orders, and OrderLine tables along with sundry support tables.

We chose this kind of project for our class because it underscores the notion of a transaction – an important concept for beginning database students and because it also underscores the important role that the internet has come to play in the database world. Over the last several years, we have tried several different approaches regarding the web-based online shopping cart portion of the course.

Initially, we tried an approach in which the students were given a "crash course" in servlet programming. The online shopping cart, as well as all web pages associated with the cart, were implemented in servlet code. While the students generally enjoyed this approach, many found servlet programming to be a formidable challenge – their java programming skills were simply not strong enough to enable them to master the ins and outs of servlet programming in a few weeks.

More recently, we have tried an approach in which the students are given a short course in JSP programming. This approach has met with considerably more overall success than the servlet-based approach. Most students, even if their java programming skills are not optimal, have had some background with HTML. Such a student encounters much less difficulty in understanding basic JSP tags than he or she does in understanding servlet programming.

Most recently, we have supplemented the JSP short course with the JSTL. Because the JSTL includes simple tags for such important tasks as outputting text to web pages, iterating over collections of data (shopping cart data e.g.), formatting numbers, dates and currencies, transforming XML data and interacting with databases, it is an ideal choice for the type of semester project we require of our students.

5. The Short Course

While no pretense is made that our students will receive a *comprehensive* overview of JSP and the JSTL, we do try to ensure that they are exposed to the most important elements of these technologies.

5.1 Web Applications

The notion of a *web application* is, of course, of fundamental importance. A web application's structure is mandated by the Java Servlet Specification and is perhaps best explained by the use of the following example:

Figure 1. Web application layout

A web application's html and jsp files are kept in the sample_webapp folder.
The WEB-INF folder is special for a couple of reasons. First, the contents of that folder are protected from access by web browsers. Second, WEB-INF contains a special file, viz. *web.xml*, that functions as a *deployment descriptor*, i.e. it contains configuration information for the web application, an application description, and perhaps other customization. The *classes* folder contains any Java class files (servlet or otherwise) required by the application. The *lib* folder contains Java Archive (JAR) files; in particular, if the application involves the JSTL, *lib* will contain *standard.jar* and *jstl.jar* – the two .jar files that constitute the JSTL.

5.2 Scopes

Server side programming involves a concept that is often new to students who have no previous server side programming experience, viz. *scope*. A scope defines the length of time an object is available and to whom it is available. There are four different scopes for sharing information between pages, requests, and users:

- Page
- Request
- Session
- Application

The default scope (*page scope*) allows actions on objects placed in it only within one page. *Request scope* should be applied to objects that are required in all pages that process the same request. *Session scope* should be applied to objects that are to be shared by several requests from the same user. E.g., a shopping cart object is usually given session scope. *Applications scope* should be applied to objects that are to be shared by all users of the same application. E.g., a "catalog" object that contains information regarding what products are for sale, a description of the products, the product prices, etc. might be given application scope. Students need to become thoroughly familiar with these different scopes and how and why they are used.

5.3 The Expression Language

Originally developed as part of the JSTL, the Expression Language (EL) has, as of JSP 2.0, become part of JSP itself. Syntactically similar to JavaScript, the EL is a language used for accessing request data and application class data. An EL expression starts and ends with a delimiter. The opening delimiter is a dollar sign concatenated to a left curly brace ${. The closing delimiter is a right curly brace }. The following simple example shows how one can include an EL expression directly in the text of a JSP page:

3 + 2 + 1 = ${3 + 2 + 1}

The browsed JSP page would then have the "output":

3 + 2 + 1 = 6

The EL contains many of the same elements present in most programming languages. In particular, it contains the "." operator for accessing a bean property or Map entry, "[ ]" for array access, "(" and ")" for grouping, "? :" for conditional test, "+, -, *, /, %" for arithmetic operations, "==, !=, <, >, <=, >=" for relational tests, "&&, || and !" for boolean operations, "Empty" for null tests and "func(arg)" for performing function calls.

EL expressions can contain variables. A variable can be created by an application or be provided implicitly by the EL and is a reference to an object. E.g., one might use the JSP

action <jsp: useBean> to create an object (perhaps a shopping cart or an item in a shopping cart) that will be used by a JSP page. The EL contains a set of *implicit* variables that provide access to many of the important attributes of a web application. The following table lists these implicit variables.

Table 1: Implicit EL variables

Variable name	Description
pageScope	A collection (a java.util.Map) of all page scope variables
resuestScope	A collection (a java.util.Map) of all request scope variables
sessionScope	A collection (a java.util.Map) of all session scope variables
applicationScope	A collection (a java.util.Map) of all application scope variables
Param	A collection (a java.util.Map) of all request parameter values as a single string value per parameter
paramValues	A collection (a java.util.Map) of all request parameter values as a String array per parameter
Header	A collection (a java.util.Map) of all request header values as a single String value per header
headerValues	A collection (a java.util.Map) of all request header values as a String array per header
Cookie	A collection (a java.util.Map) of all request cookie values as a single javax.servlet.http.Cookie value per cookie
initParam	A collection (a java.util.Map) of all application initialization parameter values as a single String per value
pageContext	An instance of the javax.servlet.jsp.PageContext class, providing access to various request data

5.4 JavaBeans

While not absolutely necessary in a JSP page, a bean is, nonetheless, a component that often is used in a JSP page. Doing so helps to separate presentation logic from business logic and, so, helps in the adherence to a MVC architecture.[1]

Creating a bean is a relatively easy matter. A bean is simply a Java class that adheres to certain coding standards. In particular, a bean class must have a no-argument constructor. A bean's properties (essentially the private fields of the class) must be accessed through *getter* and *setter* methods. While not mandated, a bean should also implement the java.io.Serializable interface so that a bean's state can be saved and restored.

The following code, adapted from [9], demonstrates how one might implement a shopping cart item and a shopping cart. The shopping cart item is not itself a bean but the shopping cart is:

```
import java.text.*;
import java.io.*;

public class ShoppingCartItem implements Serializable {
        private int itemNumber;
        private double unitPrice;
        private int count;

        public ShoppingCartItem(int itemNumber, double unitPrice) {
                this.itemNumber = itemNumber;
                this.unitPrice = unitPrice;
                this.count = 1;
        }

        public int getItemNumber() {
                return itemNumber;
        }

        public double getUnitPrice() {
                return unitPrice;
        }

        public int getCount() {
                return count;
        }

        public void setItemNumber(int itemNumber) {
                this.itemNumber = itemNumber;
        }

        public void setUnitPrice(double unitPrice) {
                this.unitPrice = unitPrice;
        }

        public void setCount(int count) {
                this.count += count;
        }

        public void incrementCount(int delta) {
                count += delta;
        }
}
```

```
import java.util.*;
import java.text.*;
import java.io.*;

public class ShoppingCart implements Serializable {
        private List items;
        private NumberFormat currencyFormat;

        public ShoppingCart() {
                items = new ArrayList();
        }

        public void setAddItem(int itemNumber, double unitPrice) {
                changeItemCount(itemNumber, 1, unitPrice);
        }

        public void setRemoveItem(int itemNumber, double unitPrice) {
                changeItemCount(itemNumber, -1, unitPrice);
        }

        public ShoppingCartItem getItem(int i) {
                return (ShoppingCartItem) items.get(i);
        }

        public String getTotalPrice() {
                Iterator i = items.iterator();
                double price = 0.0;
                while (i.hasNext()) {
                        ShoppingCartItem item = (ShoppingCartItem) i.next();
                        price += item.getUnitPrice() * item.getCount();
                }
                return currencyFormat.format(price);
        }

        public void changeItemCount(int itemNumber, int delta, double unitPrice) {
                ShoppingCartItem item = new ShoppingCartItem(itemNumber, unitPrice);
                if (items.contains(item)) {
                        ShoppingCartItem existingItem;
                        existingItem = (ShoppingCartItem) items.get(items.indexOf(item));
                        existingItem.incrementCount(delta);
                        if (existingItem.getCount() <= 0) {
                                items.remove(existingItem);
                        }
                } else {
                        if (delta > 0) {
                                items.add(item);
                        }
                }
        }
}
```

Whether the instructor decides to teach students the intricacies of bean programming or simply provides students with appropriate shopping cart item and shopping cart classes is, time constraints notwithstanding, not important. What is important is that the student

have these classes available, are aware of the methods and data contained in them, and know how to employ those methods in a JSP page.

5.5 JSTL

The JSTL is composed of five libraries, viz. the core, xml, database, formatting and function tag libraries. Each tag in a given library has a number of attributes. The following tables summarize the most important tags available in the core and database tag libraries.[2]

5.51 The Core Library

The core tag library provides support for variable management, conditional logic, imports, looping, output, and URL manipulation. Important tags in this library include the following.[3]

The <c:out> tag is used for writing data.

Table 2: <c:out>

Attribute	Description	Required	Default
Value	Information to output	Yes	None
Default	Fallback information to output	No	Body
escapeXml	True if the tag should escape XML characters	No	True

The <c:set> tag is used to save data to memory.

Table 3: <c:set>

Attribute	Description	Required	Default
value	Information to save	No	Body
target	Name of the variable whose property should be modified	No	None
property	Property to modify	No	None
var	Name of the variable to store information	No	None
scope	Scope of variable to store information	No	page

The JSTL has four tags used for conditional logic, viz. <c:if>, <c:choose>, <c:when>, and <c:otherwise>. The <c:choose> and <c:otherwise> tags accept no attributes. The <c:if> tag allows one to apply a standard conditional test.

Table 4: <c:if>

Attribute	Description	Required	Default
Test	Condition to evaluate	Yes	None
Var	Name of the variable to store the condition's result	No	None
Scope	Scope of the variable to store the condition's result	No	Page

The <c:when> tag is used with the <c:choose> tag much like a “case” is used with a “switch” in Java:

Table 5: <c:when>

Attribute	Description	Required	Default
Test	Condition to evaluate	Yes	None

The core library provides support for two looping tags, viz. <c:forEach> and <c:forTokens>. The former is used for general data while the latter is used for substrings of a given string.

Table 6: <c:forEach>

Attribute	Description	Required	Default
items	Information to loop over	No	None
begin	Element to start with (0 = first item, 1 = second item…)	No	0
end	Element to end with (0 = first item, 1 = second item…)	No	Last item in the collection
step	Process every step items	No	1 (all items)
var	Name of the variables to expose the current item	No	None
varStatus	Name of the variable to expose the loop status	No	None

The <c:param> tag is used for accessing request parameters.

Table 7: <c:param>

Attribute	Description	Required	Default
name	Name of the request parameter to set in the URL	Yes	None
value	Value of the request parameter to set in the URL	No	Body

5.52 The Database tag library

The database tag library is obviously the library of most interest in our database course. This library provides support not only for standard database queries (selects, updates, etc.) but also for transactions.

The <sql:query> tag is used for those queries that return a result set (usually select queries).

Table 8: <sql:query>

Attribute	Description	Required	Default
Sql	SQL command to execute (should return a ResultSet)	No	Body
dataSource	Database connection to use (overrides the default)	No	Default database
maxRows	Maximum number of results to store in the variable	No	Unlimited
startRow	Number of the row in the result at which to start recording	No	0
Var	Name of variable to expose the result from the database	No	None
Scope	Scope of variable to expose the result from the database	No	Page

The <sql:update> tag is used for, you guessed it, updating a database.

Table 9: <sql:update>

Attribute	Description	Required	Default
sql	SQL command to execute (should not return a ResultSet)	No	Body
dataSource	Database connection to use (overrides the default)	No	Default database
var	Name of the variable to store the count of the affected rows	No	None
scope	Scope of the variable to store the count of the affected rows	No	page

The <sql:param> tag is used for those queries where one or more parts of the query need to be filled in by a parameter.

Table 10: <sql:param>

Attribute	Description	Required	Default
value	Value of the parameter to set	No	Body

The <sql:dateParam> us used like the <sql:param> tag except that the data provided is a date object.

The <sql:transaction> tag allows the user to formulate groups of <sql:query> and <sql:update> entries into transactions.

Table 11: <sql:transaction>

Attribute	Description	Required	Default
dataSource	Database connection to use (overrides the Default)	No	Default database
Isolation	Transaction isolation (READ_COMMITTED, READ_UNCOMMITTED, REPEATABLE_READ, or SERIALIZABLE)	No	Database's default

5.6 Accessing a Database Using the JSTL

Figures 2 and 3 exhibit a database transaction before and after a "valid" transaction and before and after an invalid transaction:

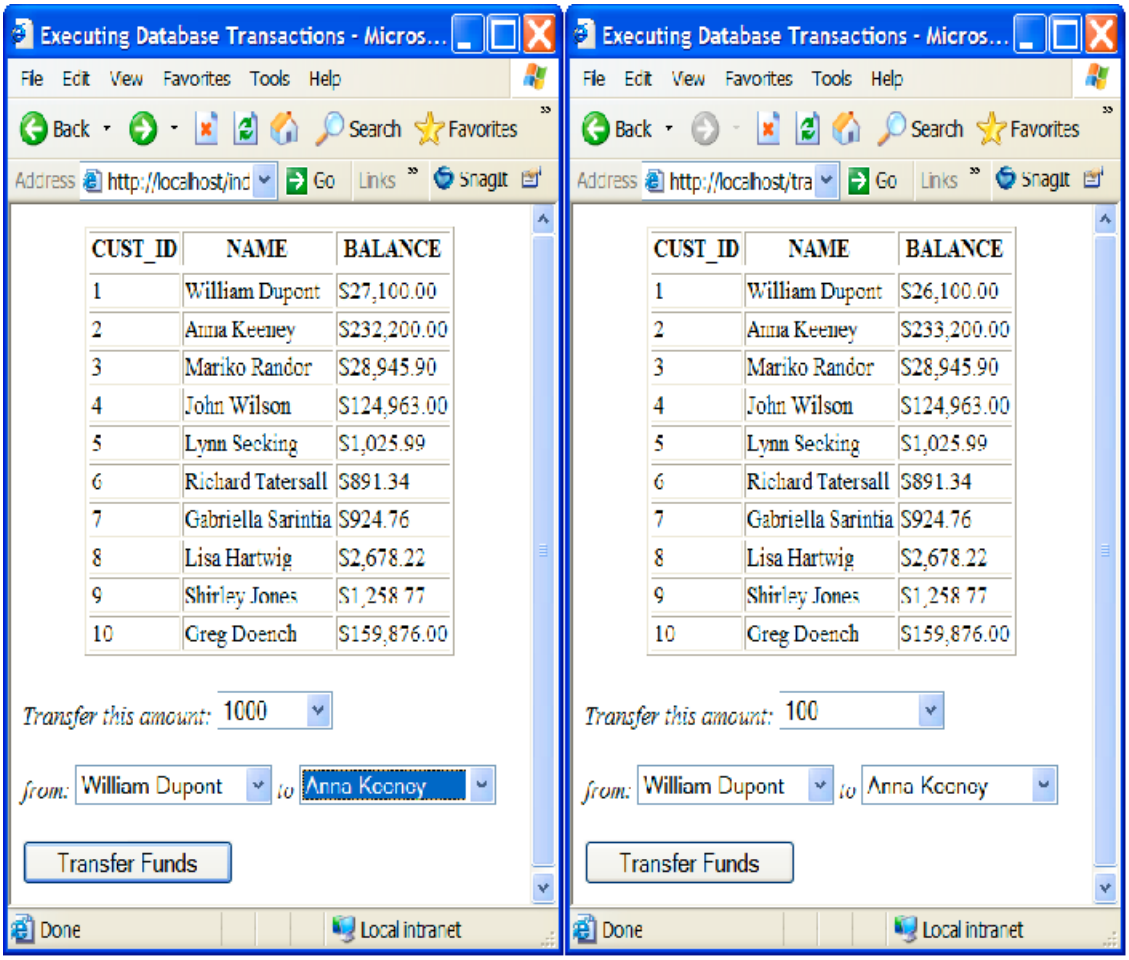

Figure 2: Before and after a valid transaction

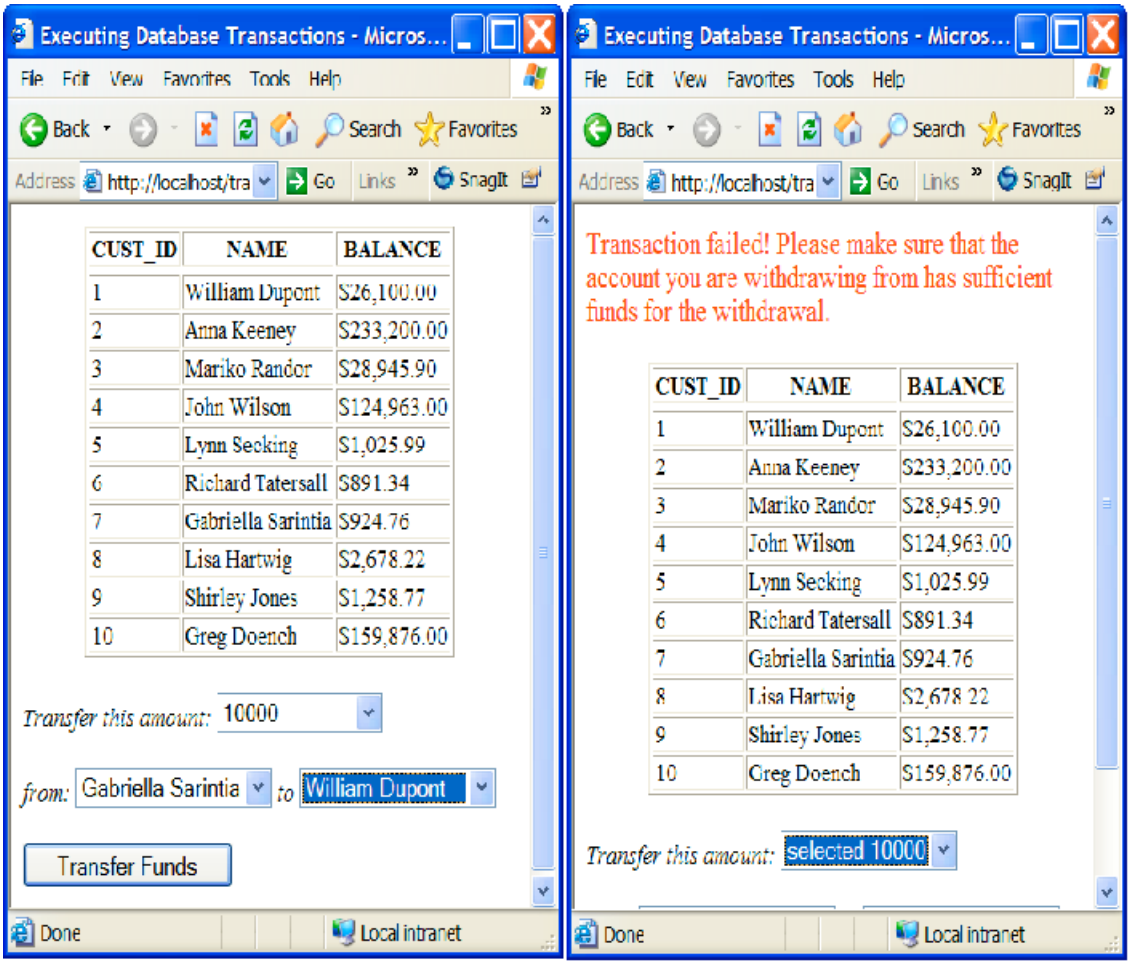

Figure 3: Before and after an invalid transaction

The code that generates these figures is adapted from chapter 9 of [11]. There are two files involved, viz. index.jsp and transfer_funds.jsp. For the sake of brevity, the former is not displayed here. One can guess from the figures above most of its content. The code for the latter file is displayed in figure 6. Note the ease with which transactions are coded.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">.
<html>.
    <head>.
        <title>Executing Database Transactions</title>.
    </head>.
    .
    <body>.
        <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>.
        <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>.
        <%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>.
        <$-- The transaction is enclosed in a <c:catch> action.  If the transaction fails,.
        the <sql:transaction> action will throw an exception, which <c:catch> stores in a .
        scoped variable name transactionException --$>.
        <c:catch var='transactionException'>.
            <$-- The transaction... --$>.
            <sql:transaction>.
                <$-- Withdraw money from the "from" customer's account --$>.
                <sql:update>.
                        UPDATE ACCOUNTS SET BALANCE = BALANCE - ? WHERE CUST_ID = ?.
                        <sql:param value='${param.amount}'/>.
                        <sql:param value='${param.fromCustomer}'/>.
                </sql:update>.
                <$-- Deposit the money withdrawn from the "from" customer's account.
                in the "to" customer's account --$>.
                <sql:update>.
                        UPDATE ACCOUNTS SET BALANCE = BALANCE + ? WHERE CUST_ID = ?.
                        <sql:param value='${param.amount}'/>.
                        <sql:param value='${param.toCustomer}'/>.
                </sql:update>.
            </sql:transaction>.
        </c:catch>.
        .
        <$-- If the transactionException scoped variable is not empty, the transaction failed --$>.
        <c:if test='${not empty transactionException}'>.
            <$-- Display the error message --$>.
            <font size = '4', color= 'red'>.
                Transaction failed!  Please make sure that the account you.
                are withdrawing from has sufficient funds for the withdrawal..
            </font>.
        </c:if>.
        .
        <$-- Import the JSP page with the form for transferring funds --$>.
        <c:import url='index.jsp'/>.
    </body>.
</html>.
```

Figure 4: Demonstrating Transactions

6. The Server

In section 2 we noted that one challenge facing the instructor who wishes to include a web-based database access component in his/her database course is the issue of a web server. Teaching students to install, use and maintain a servlet container like Tomcat is, while possible, not likely to be feasible in a course in which traditional database topics are the main focus. We choose to use ServletApp.[4] ServletApp is based on Jetty an open source HTTP server and servlet/JSP container. ServletApp is relatively small (about 2 MB) and its installation, configuration and maintenance requirements are minimal. A student can install ServletApp on their home machines by simply unzipping it. No other configuration responsibilities beyond having a recent JDK installed and downloading an

appropriate JDBC driver are required. The current version of ServletApp supports the JSP 2.0 and servlet 2.4 specifications. Meeting these specifications is a requirment for the use of version 1.1 (the latest) of the JSTL. Using the JSTL a student can construct a prototype of a web-based database application by mastering a handful of tags from the JSTL instead of having to master the complexities of servlet programming or some other server side programming technology.

7. Conclusions

We have tried several approaches in our attempts to incorporate a web-based database access component in our beginning database course. Initially we tried a servlet-based approach but found this to be too demanding for students whose programming skills were, in many cases, less than optimal. A JSP-based approach was attempted in the Spring semester 2003. This proved to be a big improvement over the servlet-based approach with many more students expressing interest and excitement about JSP. During the fall semester 2003 we again used a JSP-based approach with some JSTL thrown in for good measure. Unfortunately, we were only able at that time, to utilize version 1.0 of JSTL. During the current semester, we are utilizing the approach outlined in the pages above. We expect that the simplicity provided by JSTL version 1.1 will finally make the integration of a web-based database access component in our introductory database course a pleasant reality for both instructor and student.

8. Resources

Those who are interested in investigating JSTL for possible use in their courses might find the following books helpful. [10] and [11] are both excellent and provide detailed information regarding JSTL version 1.0. [12] is concerned with JSP in general and provides less detailed information regarding JSTL. However, [12]'s coverage is up to date with JSTL version 1.1.

References

[1] Hoffer, J.A., Prescott, M.B. and McFadden, F.R. *Modern Database Management.* Sixth Edition. Upper Saddle River, New Jersey: Prentice Hall PTR. 2002.

[2] Connolly, T. and Begg, C. *Database Systems*. Third Edition. Addison Wesley Harlow England. 2002.

[3] Ramakrishnan, R. and Gehrke, J. *Database Management Systems.* Third Editon. Mcgraw Hill. New York. 2003.

[4] Rob, P. and Coronel C. *Database Systems: Design, Implementation & Management.* Sixth Edition. Thompson Course Technology. Boston. 2004.

[5] Ricardi, G. *Principles of Database Systems with Internet and Java Applications.* First Edition. Addison Wesley. 2001.

[6] Kroenke, D. *Database Processing.* Ninth Edition. Pearson/Prentice Hall. 2004.

[7] Elmasri, R. and Navathe, S.B. *Fundamentals of Database Systems.* Fourth Edition. Pearson/Addison Wesley. 2004.

[8] Date, C.J. *An Introduction to Database Systems.* Eighth Edition. Pearson/Addison Wesley. 2004.

[9] Fields, D.K., Kolb, M.A., Bayern, S. *Web Development with Java Server Pages*. Second Edition. Manning 2002

[10] Bayern, S. *JSTL In Action*. First Edition. Manning 2003.

[11] Geary, D.M. *Core JSTL: Mastering the JSP Standard Tag Library.* First Edition, Prentice Hall/PTR 2003

[12] http://hart.cs.mnsu.edu/home/downloads/servletapp

[13] Bergsten, H. *JavaServer Pages*. Third Edition. O'Reilly 2004

Notes

[1] I've "bitten my tongue" over this claim several times. There are those who would argue (correctly) that the approach taken in the pages that follow is in direct violation of MVC. They argue that embedding SQL calls in a JSP page violates MVC and that SQL calls should *not* appear in the view component. To this I can only plead "guilty as charged". However, I would remind such critics that, given the time constraints in an introductory database class, embedding SQL calls in JSP pages in a *prototype* web application is a forgivable sin.

[2] For the sake of brevity, we omit the xml, formatting, and function libraries.

[3] The summary is adapted from Appendix A of [10].

[4] An earlier version of ServletApp was presented at the 36th annual meeting of MICS in Duluth MN 2003. The current version of ServleApp has grown a bit in both size and capability while still maintaining an overall simplicity. ServletApp can be downloaded from [12]