Robert Arthur Hall. A Web Based Data Entry and Reporting Tool. A Master's Paper for the M.S. in I.S. degree. April, 2001. 65 pages. Advisor: Gregory B. Newby.

This project was to create a tool that provides a means for anyone to create the Web-based reports or forms by which remote users may update or view data contained in a database. This is accomplished by allowing an administrator to create the reports or forms, users to access a listing of those reports, and managers to view data entered via those updates. This paper focuses on design and implementation, including the pros and cons of the choices made. The appendixes contain code samples and all design documents created during this process.

Headings:

Front-end software

Information systems – Design

Internet

Java

JSP

Web

A WEB BASED DATA ENTRY AND REPORTING TOOL

by
Robert A. Hall

A Masters' paper submitted to the faculty of the
School of Information and Library Science of the
University of North Carolina at Chapel Hill in partial
fulfillment of the requirements for the degree of
Master of Science in Information Science.

Chapel Hill, North Carolina

April, 2001

Approved by:

_____

Advisor

Introduction

Background and Related Work

For this project, I created a tool that provides a means for anyone to create the Web-based reports or forms by which remote users may update or view data contained in a database. This is accomplished by allowing an administrator to create the reports or forms, users to access a listing of those reports, and managers to view data entered via those updates. This paper focuses on design and implementation, including the pros and cons of the choices made. The appendixes contains code samples and all design documents created during this process.

Originally, I conceived this project as a way for an Insurance company to accept claims from either remote agents or directly from customers. After a while, I decided to restart the project as a general database reporting tool, and the result of that decision was the Functional Specification document in Appendix A. There are several high quality tools that perform functions far beyond the scope of my project, but they generally require licensing fees. I intend to release my code as freeware. The main difference between those tools and the one I created are mainly concerning the GUI; I have taken a minimal approach to the GUI presentation, while the more expensive tools are generally have an impressing user interface.

Project Goals

The goal of this system is to provide easy access to a database through a Web browser, in order to facilitate data entry and reporting. Companies would be able to obtain data easily from anyone who has Internet access, and store it in a central location. This would be especially useful for registration, making claims to an insurance company, or any other use that requires periodic updates from users not necessarily near to the hosting company. The software is intended to be easy to use, simple to install, and provide useful results.

Users of Proposed System

As freeware, this project will be available to anyone with an Internet connection. It will most likely only be useful to those people who have the need for Internet (or Intranet) access to a backend database. As conceived, there are three types of users of this program:

- Administrator
- Manager
- Operator

Each of these types of users illustrates a different focus of the project. Administrators want to create reports to control the operators' input into the system. Managers require a way to report on what data has been entered. Operators want to be able to use this tool with little or no training to facilitate entering data into the database.

Problem Definition

A great many companies use relational databases such as DB2 or Oracle to store and maintain their important information. Usually, there are one or two people within the company who control access to that database, and therefore any access to the information must be filtered through these people. (This is usually a DBA or someone else familiar with SQL). There are tools that provide an easy to use GUI interface to interact with the data, but they are expensive. If there was a low cost, easy to use alternative, the Operators and Managers could perform data entry and other daily maintenance tasks without ever needing to learn SQL, go directly through a database administrator, or paying a lot of money. The Administrator who sets up the database, however, will need to know SQL, although technically no SQL is required to set up the reports, as long as the table names in the database are descriptive. In order to better describe what my project is intended to do, I have included the primary 'Use Case' for each of the various types of people who will be using my program. A Use Case is a listing of steps that a particular 'Actor', or user, may go through. There can be many use cases, each detailing a different 'path' through the application; the 'primary' use case is simply the one that should happen most often.

Administrator Use Case

1. Log into the system.
2. Choose the report to edit, or type in a new report name.
3. Chose a database table from the drop down menu, type in a Section Heading and a Label, then submit the page. The same page will reload; except that the column is added to the report. (This step is repeatable.)
4. Click on the 'ViewReport' button. This will load the report on a new page, as a user would see it.
5. Click 'Logout' to return to the login screen.

Other functions available to Administrator:

- Create account (either manager, user, or admin)
- Delete report
- Modify report

User Use Case

1. Log into the system.
2. Choose the report to view.
3. Type information into the available fields and submit the page, which will copy the data to the database and reload the page with empty fields. (This step is repeatable.)
4. Click 'Logout' to return to the login screen.

Other functions available to User:

- None.

Manager Use Case

1. Log into the system.
2. Chose the report to view.
3. Print the report for later viewing through the browser 'print' button.
4. Click 'SelectAnotherReport' button to choose another report to view.

Other functions available to Manager:

- None.

Design Strategy and Technology Choices

I created a functional specification document that details the requirements of the proposed system. It also contains three use case scenarios, one for each type of user. The choice of what technology to use was easy, given the nature of the requirements. The project was coded using JSP and Java. This means that the final product can be deployed on any Web server capable of serving JSP pages (Apache, BEA Weblogic, Websphere, and others) and can access any database for which JDBC drivers exist, including Oracle, DB2, and MySQL.

Architectural Design

Java

The version of Java that I used was JDK 1.3. I used no Java IDE or any other coding tools besides the javac command that comes with the JDK download. My primary text editor was vi. Outside of the JSP bean code, I coded several 'helper' Java classes, containing methods that facilitate connecting to and using a database.

DatabaseAccess.java

This is the bean that maintains the data between the various JSP pages. There are several functions that create HTML code for use in certain JSP pages.

ReportPool.java

Contains a singleton class that controls access to the database. A 'Singleton' class (i.e. a class which follows the singleton design pattern) has a getInstance() method that returns either a new instance of the class, or the existing instance, thus allowing only one instance of the class to exist at any given time. This is useful for something like connection pooling, allowing multiple people to use the same pool of already-available connections.

DBHelper.java

This class contains several methods that accept a ReportPool object and a query or update String, and return a result. The main reason I created this class was so I wouldn't have to worry about coding the same methods over and over again throughout the rest of my code.

ConnectionPool.java

The base class of ReportPool; this class contains all the functionality used to create a pool of available connections to a given database.

ReportData.java

This is a custom class which I created to hold all information about a particular report, except for reportname. This entire object is contained in the database as a BLOB, with the reportname as the primary key in that table. This class also contains a Hashtable containing all the objects of DataObject type stored in a particular report.

DataObject.java

This is a custom class that I created to hold information about a particular column on a report, including section, label, column name, and table name.

JSP

The version of JSP that I used was 1.2, the newest available at the time I started the project. During development, I used the ANT tool, which can, after being set up correctly, automatically compile and deploy the pages and classes being developed to whatever server that you want. The ANT tool comes with most versions of Tomcat. I used the Tomcat server version 3.2.1 to test and serve the pages that I created, as follows:

Controller.jsp

This JSP page contains no HTML or other ‘display’ code. It acts as the ‘traffic cop’, called upon by the other pages to tell them where the user should be sent next. It also contains all the behind the scenes processing.

CreateAccount.jsp

This page interfaces the functionally of being able to add a new Operator, Manager, or Administrator to the list of approved users of the system.

EditReports.jsp

Contains the necessary GUI components to interface with the Administrator in order to create the reports. It shows a drop down menu listing all the available database tables and columns, and text boxes for Section Heading and Label information. As a particular column is added to the report, it is listed here as well, and can also be removed. This information is stored in the database as a Java object, called ReportData. Only the Administrator can access this page.

ErrorPage.jsp

If a user performs an action that is not accounted for in my code, this page is called. It holds a link back to the login, and a brief explanation of the error. If this page appears, it would necessitate a bug report to direct future improvements.

Login.jsp

This page accepts a userid and password from the user. The IdType of the user (i.e. Administrator, User, or Manger) is associated with the id in the database.

ShowReports.jsp

This shows a listing of all reports available to a user, and allows him or her to choose one. The Administrator has several more options here than the manager or user. The User and Manager types can only chose a report to view; an Administrator can also choose a report to edit, or delete, or create a new one.

ViewReport.jsp

This is the page that shows the finished report, ready to have data inserted into the database. If a particular field is left blank, nothing will be placed in the database for that field only. In other words, not all fields need to be filled out for a successful submit. The information concerning the report is pulled out of the database, parsed into sections, and the labels for each column are put beside text boxes

Architecture

I used a standard Model-View-Controller pattern for this project. In MVC design, the Model component represents the application data, plus methods that operate on that data, with no user interface. The View component presents the data to the user. The Controller component translates user actions into operations on the Model. The Model, in turn, updates the View to reflect changes to data. For my project, the JSP pages and the Bean represent the View component, and the Controller.jsp page represents the Controller component. The Model component is the back-end database, as well as the code I wrote to interface with it. Basically, each JSP page contains an html form that, when submitted, saves all gathered information into a bean and loads a ‘controller’ page which performs any necessary functions based on current state, and then forwards the user to the next page in the sequence. The Controller page is the only page to interact with the database through my DBHelper class; all other pages only interact with the Bean.

Database Layer

Any database that has associated JDBC drivers can be used for this project. Instructions on how to configure the program to use a particular database are included in the documentation. Instructions for setting up and maintaining any database are out of scope for my project.

Interface Design

Because of several of the original requirements, no Javascript was used in the interface. At the core, these pages are simply HTML forms passing information to the Controller page, which knows where to send the user based on the information given. The Controller page also holds all of the interaction with the database; all other pages interact only with the Bean.

Advantages of Design

State is maintained throughout the program, and can be easily checked at any point. Almost all 'processing' code is contained in the 'controller' page, making updates and maintenance much easier. Having a separate JSP page for each function helps to make the interface much more easily understood and, again, easier to update.

Disadvantages of Design

Placing most of the processing code in a single 'controller' page means that that code is fairly long, requiring the code to be well commented and formatted for readability. Also, any error in the controller code makes the entire project unable to run. A Java-enabled browser must be available to use this tool Another problem I ran into (which I will detail later in my 'challenges' section) is that there is a strong dependency on the functions allowed by the particular JDBC drivers and database. Just because a particular function is available in the Java API doesn't necessarily mean that it has been coded into the JDBC driver, or that the database allows it.

Implementation

Criteria for Success

My first level of success will be achieved when each type of user can go through their associated 'Use Case' scenario and not encounter any errors or problems that cause the program to be unusable. Unfortunately, I was unable to fit a full usability study within this project, so no objective evaluation of

success will be made of the project itself. I will consider this project to be a complete success when someone downloads it from a freeware site and successfully implements it.

Challenges

The greatest challenge for me was in writing the Functional Specifications document, as I have had no experience with such documents. I basically followed the RUP®, or Rational Unified Process, which can be obtained (for evaluation) at http://www.rational.com. This is a software engineering process that uses software 'best practices' as tested by Rational Software and throughout the industry. It places extensive guidelines, templates, and examples on-line to describe, mentor, and facilitate varied development activities. This includes, but is not limited to, areas such as business modeling, Web architectures and testing. It is also a highly customizable framework, adaptable to many project sizes. Using some of the templates described, I was able to pull together the documents required for this project quite easily.

The actual coding of this project was complex, and made difficult by a few particular snags. The worst problem, I think, was storing a Java object in the database to maintain the report information. It turns out that you have to implement the java.io.Serializable interface, and encode the Java object as an array of bytes before you can store it. To pull it back out of the database, you need to interpret it as a java.io.InputStream and cast it back to the class type that you actually want to work with.

Deployment Plan

I have created a WAR file containing all the JSP pages, bean code, helper classes, and all the documentation for this project. A WAR file is a form of ZIP file with a standardized directory structure used for easier deployment of JSP websites to Web servers such as Tomcat and BEA Weblogic. This WAR file can be created automatically through use of the ANT tool that I mentioned earlier. Once created, the file can simply be placed in the appropriate directory of the Web server in use. The database.properties file would have to be modified to point the application to the correct database, and the database creation scripts would have to be run against the database to create the database tables required for use of this application. After that, implementations vary. As an example, for Tomcat you must

1. Place a line in the server.xml file to map the directory for the new web site.

2. Restart the server.

Future Modifications

A full usability test would probably suggest modifications to the GUI, as well as possible functionality changes. I think future modifications would be made to:

- Improved, more interactive, GUI
    - Drag and drop report creation
    - Graphics instead of text for report types and help menus
- Better way of displaying what tables and fields are available for a report.
- Update the type or password of an approved user without having to delete them first.
- Allow editing of column entries within a report, without deleting and re-adding them.
- More 'User Types'.
- Templates to follow when creating reports.
- Automatic charting based on given columns from the database

Conclusion

This project thesis was a natural extension of several classes that I have taken during the course of my instruction at this University of North Carolina at Chapel Hill. I strongly feel that Java and its related technologies are on the upswing, and will be here for many years to come, meaning that a tool like this one could possibly be used for many years, even if only as a learning tool. While I was working on this project, I discovered much information about how design documents relate (and don't relate) to actual coding, and how important it is to plan out how you are going to do something and why. There were several instances where I had a choice of competing technology, and the requirements document was what determined the final choice.

This project, in general, followed the pattern of most design projects:

Inception

Specification

Design

Development

Deployment

One exception is that it has yet to go through the Maintenance phase, which could last for many years. I was able to meet every required feature, as defined in my specification document, except possibly for number 5 (Must allow an Administrator to link fields in one or more tables to emulate ‘joins’), which was rendered meaningless by the way in which I coded the project. This function can certainly be performed, but my program allows no special provision for doing so. I also completed five of the 12 desired features. I had no special graphics, no sorting, no math functions, cannot be ported automatically into an Excel spreadsheet, no format customization, no online help menu, and no authentication through an LDAP server. If a second version of my software is requested, I will be able to finish coding these items. There was no technological barrier to completion, except for a lack of time. The goal of this system was to provide easy access to a database through a Web browser, in order to facilitate data entry, and reporting. I feel that I have met that goal, and allowed for future improvements.

Appendix A: Bibliography

Ambler, Scott W. (1998). Process Patterns: Building Large-Scale Systems Using Object Technology. Cambridge, UK: Cambridge University Press.

Cockburn, Alistair. (2000). Writing Effective Use Cases. Boston, MS: Addison-Wesley.

Hall, Marty. (2000). Core Servlets and JavaServer Pages. Upper Saddle River, NJ: Prentice Hall PTR.

Kruchten, Philippe. (2000). The Rational Unified Process: An Introduction. Reading, MS: Addison-Wesley.

Langr, Jeff., (1999). Java Style, Patterns for Implementation. Upper Saddle River, NJ: Prentice Hall PTR.

Appendix B: Functional Specifications

Purpose of Project

The goal of this system is to provide easy access to a database through a Web browser, in order to facilitate data entry, and reporting. Companies would be able to obtain data easily from anyone who has Internet access, and store it in a central location. This would be especially useful for registration, making claims to an insurance company, or any other use that requires periodic updates from users not necessarily near to the hosting company. This tool will be posted to the Internet under the Open Source license.

Terms Used / Glossary

Administrator- The person responsible for setting up the forms and reports for the end user(s).

Operator- The people entering data into the system through the forms.

Developer – The team responsible for the development of the system.

JDBC – Java DataBase Connection. Drivers which make connecting to a database through Java possible.

JSP – Java Server Pages. A server-side GUI presentation layer.

Manager – Person who uses and analyses the reports.

SILS – School of Information and Library Science at UNC at Chapel Hill

Ruby – The server in use at UNC at Chapel Hill for development of the system.

UNC – University of North Carolina

Stakeholders

Designer: Wants to create a great tool in order to start building a reputation on the Internet.

Administrator: Wants to create reports for users to input data, and a means to report on what data has been entered.

Manager: Wants to be able to have accurate status reports presented in real time.

Users: Want to be able to use this tool in order to shorten the learning curve when they realize their information needs to be both Web accessible and stored in a database.

1c. What is in scope, what is out of scope?

Technology

4a. What technology requirements are there for this system?

This tool will be coded in JSP. The requirements are:

1. A JSP Server
2. A database
3. JDBC drivers appropriate to the chosen database
4. Java 1.3
5. A Web browser
6. A computer to run the JSP server, and a client for testing. (They can be the same computer while in development.)

In the development phase, the tool will access a MYSQL database already installed at SILS. The Alpha deployment will allow no computer, except for the Designer, to access the tool. The Beta deployment will be placed on a publicly accessible location on the SILS servers. The final product will not depend on any particular system, and will require some configuration on the part of the Administrator to use successfully. Java and JSP are portable between any systems. T

Critical Requirements

1. Must be able to use JDBC drivers to connect to a database.
2. Must allow the Administrator to view what tables are available in the database.
3. Must allow a means by which Administrators can create report formats.
4. Must allow a means by which an Administrator can choose what tables to include in a report.
5. Must allow an Administrator to link fields in one or more tables to emulate ‘joins’.
6. Must allow an Administrator to choose what fields in a particular table to display for a report.

7. Must allow the Administrator to save the report for future use
8. Must allow the Administrator to delete existing reports.
9. Must allow a means by which users can enter data into the database.
10. The tool will be web-based, and should not depend on any particular browser for function.
11. Some form of documentation describing the functionality of the tool, and how to use it.
12. A form of security based on database access.

Desired Features

1. Should be able to be customized to link to any database for which JDBC drivers exist.
2. For the report designer, show the list of tables graphically, including which tables are linked by foreign keys.
3. The reports should have the ability to sort or multiple sort on displayed fields
4. The report designer should allow ‘labels’ that are different from the column names in the database.
5. Dynamically show all tables and their fields
6. The report designer will allow for choosing various math functions for numerical data.
7. The report designer will allow for truncation of string data.
8. Port data into an Excel spreadsheet
9. Ability to customize format of report.
10. Ability to update existing reports.
11. An on-line help menu.
12. The ability to authenticate users through an LDAP server.

Scope of Project

Explaining how to install the required software required for successful operation of this tool is out of scope. This includes: A database, Java 1.3, a JSP Server, a Web Browser, Ability to access the JSP server via an Internet or Intranet. Enabling users to connect to a database, manipulate the information, and have it print to the screen in a printable format is in scope. If the database has no JDBC drivers associated with it, then

connecting to it is out of scope. It is assumed that the database tables have already been created. No 'special case' business rules from any particular user will be coded into the project by the designer. After release, the code will be maintained by the Designer on as 'as time allows' basis. Any coding efforts by other parties will be coordinated on an 'as needed' basis.

Appendix C: README.TXT File

Database Access Tool

This software is intended to provide an easy and fast way to provide internet access to your data. It is provided free of charge. Please send all bugs and suggestions for improvement to hallr@ils.unc.edu.

I. Required Software

To use this software, you already must have installed

-A database server with existing JDBC drivers

-An application server such as Tomcat or BEA Weblogic

Please note that the application server and database server need not be on the same computer, and that all software required to run either of the above also needs to be installed. (For example, Java 1.3 needs to be installed before Tomcat will run correctly.)

II. Installation Instructions

1. Uncompress the zip file into a directory on the server you'll be using for development.

2. Modify the build.xml file so that the information is accurate for where you want the program to be deployed.

3. Modify the database.properties file so that it is using the correct JDBC drivers, URL, and login information.

4. Type 'build' from the same directory containing the build.xml file.

5. Restart your application server.

III. SQL Table Creation Scripts

The following scripts will create all the database tables required in order to use this software. You may change the id and password for the initial admin account, but please don't change the id type. Please note that this tool makes no provision for changing passwords or deleting users at this time.

CREATE TABLE reportinfo (reportid int not null auto_increment, primary key(reportid), reportname varchar(50) not null, reportdata longblob, reporttype text);
CREATE TABLE reportusage (username varchar(10) not null, reportname varchar(50) not null, primary key(username, reportname), submits int);
CREATE TABLE admin (username varchar(10) not null, primary key(username), password varchar(50) not null, idtype varchar(10) not null, firstname varchar(50), lastname varchar(50));
INSERT INTO admin (username, password, idtype) VALUES ('admin', PASSWORD('admin'), 'admin');

IV. Using the Software

The initial page of the website is Login.jsp, but there is a convenience index.html file included in the project that will forward you to this page automatically if Javascript is enabled in your browser. (Note that no other Javascript has been included in this project.)

Once the admin has logged in, he or she can create new managers or users, and can create reports. A report of 'manager' type will show the data in that table to the viewer, while a report of 'user' type will allow the viewer to submit data directly into the listed fields.

Appendix D: CODE

DatabaseAccess.java

```
import java.sql.*;

import amp.dbconnect.*;

import java.util.*;

import java.io.*;


public class DatabaseAccess {


    private String action;

    private Hashtable allData;

    private ReportData currentReport;

    private String dataLabel;

    private String dataObject;

    private String dataSection;

    private String error;

    private String idType;

    private boolean loggedIn;

    private String newIdType;

    private String newPassword;

    private String newPassword2;

    private String newReportName;

    private String newUser;

    private String password;

    private String prevPage;

    private String reportName;

    private String selectField;
```

```
private String reportType;
private String userName;

public String getAction() {
   return action;
}
public Hashtable getAllData() {
   return allData;
}
public ReportData getCurrentReport() {
   return currentReport;
}
public String getDataLabel() {
   return dataLabel;
}
public String getDataObject() {
   return dataObject;
}
public String getDataSection() {
   return dataSection;
}
public String getError() {
   return error;
}
public String getIdType() {
   return idType;
}
public boolean getLoggedIn() {
```

```
    return loggedIn;
}
public String getNewIdType() {
    return newIdType;
}
public String getNewPassword() {
    return newPassword;
}
public String getNewPassword2() {
    return newPassword2;
}
public String getNewReportName() {
    return newReportName;
}
public String getNewUser() {
    return newUser;
}
public String getPassword() {
    return password;
}
public String getPrevPage() {
    return prevPage;
}
public String getReportName() {
    return reportName;
}
public String getReportType() {
    return reportType;
```

```
}
public String getSelectField() {
   return selectField;
}
public String getUserName() {
   return userName;
}
public void setAction(String inA) {
   action=inA;
}
public void setAllData(Hashtable inH) {
   allData = inH;
}
public void setCurrentReport(ReportData inRD) {
   currentReport = inRD;
}
public void setDataLabel(String inS) {
   dataLabel = inS;
}
public void setDataObject(String inS) {
   dataObject = inS;
}
public void setDataSection(String inS) {
   dataSection = inS;
}
public void setError(String inS) {
   error = inS;
}
```

```
public void setIdType(String inS) {
   idType = inS;
}
public void setLoggedIn(boolean iB) {
   loggedIn = iB;
}
public void setNewIdType(String inS) {
   newIdType=inS;
}
public void setNewPassword(String inS) {
   newPassword = inS;
}
public void setNewPassword2(String inS) {
   newPassword2=inS;
}
public void setNewReportName(String iS) {
   newReportName = iS;
}
public void setNewUser(String inS) {
   newUser=inS;
}
public void setPassword(String inS) {
   password = inS;
}
public void setPrevPage(String inS) {
   prevPage = inS;
}
public void setReportName(String inS) {
```

```
        reportName = inS;
    }
    public void setReportType(String inS) {
        reportType = inS;
    }
    public void setUserName(String inS) {
        userName = inS;
    }
    public String getReportNames() throws SQLException {
        String where = "";
        if ( !idType.equals("admin") ) {
            where = " where reporttype = '"+idType+"' ";
        }
        String sqlString = "select reportname from reportinfo "+where+" order by reportname";
        String answer = "";
        ResultSet rs = DBHelper.getQuery(ReportPool.getInstance(),sqlString);
        while (rs.next()) {
            answer += "<option
value="+rs.getString("reportname")+">"+rs.getString("reportname")+"</option>\n";
        }
        return answer;
    }
    public boolean verifyNewReportName() throws SQLException {
        if (newReportName == null || newReportName.length() == 0 ) {
            return false;
        }
        String sqlString = "select reportname from reportinfo where UPPER(reportname) =
UPPER('"+newReportName+"')";
```

```
        ResultSet rs = DBHelper.getQuery(ReportPool.getInstance(),sqlString);
        if (rs == null || !rs.next()) {
            return true;
        } else {
            return false;
        }
    }
    public void deleteReport() throws SQLException {
        String sqlString = "delete from reportinfo where reportname='"+reportName+"'";
        int result = DBHelper.doUpdate(ReportPool.getInstance(),sqlString);
        if (result == 0) {
            error = "No delete performed. Unknown error.";
        }
    }
    public boolean insertReport() throws SQLException {
        ReportData newReport = new ReportData();
        String sqlString = "insert into reportinfo (reportname, reporttype, reportdata) values (?,?,?)";
        int result = DBHelper.doPSInsert(ReportPool.getInstance(), sqlString, reportName, "user",
newReport.encode());
        reportType = "user";
        if (result==0) {
            error = "Update failed. No report added.";
            reportType = "";
            return false;
        }
        return true;
    }
    public String getReportTypes() {
```

```
        String answer = "";
        if ( reportType != null && reportType.equals("manager") ) {
            answer += "<option value=\"manager\" selected>Manager</OPTION>";
        } else {
            answer += "<option value=\"manager\">Manager</OPTION>";
        }
        if ( reportType != null && reportType.equals("user") ) {
            answer += "<option value=\"user\" selected>User</OPTION>";
        } else {
            answer += "<option value=\"user\">User</OPTION>";
        }
        return answer;
    }
    public boolean validate() throws SQLException {
        String sqlString= "select idtype from admin where "+
                    "UPPER(username) = UPPER('"+userName+"') and password =
PASSWORD('"+password+"')";
        ResultSet rs = DBHelper.getQuery(ReportPool.getInstance(), sqlString);
        if ( rs != null && rs.next() ) {
            idType = rs.getString("idtype");
            loggedIn = true;
            return true;
        } else {
            loggedIn = false;
            return false;
        }
    }
    public boolean validateAccount() throws SQLException {
```

```
    // Get all of the input fields from the form
    // Validate the user
    if ((newUser == null) || (newUser.length() == 0)) {
       return false;
    } // ends if
    // Validate the password
    if ((newPassword == null) || (newPassword.length() == 0)) {
       return false;
    } //ends if
    // Validate the re-entry of the password
    if ((newPassword2 == null) || !newPassword.equals(newPassword2)) {
       return false;
    } //ends if
    // Make sure the given user does not exist
    if (findUser(newUser)) {
       return false;
    } // ends if
    // All of the data was valid and we have a unique user
    // ID. Add the user to the database
    // The JDBC Connection and PreparedStatment objects
    String sqlString = "insert into admin (username, password, idtype) " +
                  "values ('"+newUser+"', PASSWORD('"+newPassword+"'), '"+idType+"')";
    int rs = DBHelper.doUpdate(ReportPool.getInstance(), sqlString);
    return true;
  } //ends validateAccount
  public boolean findUser(String user) throws SQLException {
    String sqlString = "SELECT username from admin where username='"+user+"'";
    ResultSet rs = DBHelper.getQuery(ReportPool.getInstance(), sqlString);
```

```
        if (rs == null || !rs.next()) {
            return false;
        }
        return true;
    } // ends findUser()
    public Hashtable getDataObjects() throws SQLException {
        Hashtable newHash = new Hashtable();
        DatabaseMetaData dbmd = DBHelper.getDBMD(ReportPool.getInstance());
        ResultSet rs = dbmd.getColumns(null,null,"","");
        while (rs.next()) {
            DataObject dd = new DataObject(rs.getString("TABLE_NAME"),
rs.getString("COLUMN_NAME"),
                                    rs.getInt("DATA_TYPE"), rs.getString("TYPE_NAME"));
            newHash.put(dd.getKey(),dd);
        }
        return newHash;
    }
    public String getDataEntries() throws SQLException {
        String answer = "";
        String temp="";
        if (allData == null || allData.size() == 0) {
            return answer;
        }
        for (Enumeration e = allData.keys(); e.hasMoreElements();) {
            temp = (String)e.nextElement();
            answer += "<option value="+temp+">"+temp+"</option>\n";
        }
        return answer;
```

```
    }
    public ReportData getReportData() throws SQLException {
        String sqlString = "select reporttype, reportdata from reportinfo where reportname =
"'+reportName+"'";
        ResultSet rs = DBHelper.getQuery(ReportPool.getInstance(), sqlString);
        ReportData newData = null;
        if (rs != null && rs.next() ) {
            //Blob newBlob = rs.getBlob("reportdata");
            reportType = rs.getString("reporttype");
            try {
                ObjectInputStream oo = new ObjectInputStream(rs.getBinaryStream("reportdata"));
                newData = (ReportData)oo.readObject();
            } catch (Exception e) {
                error = e.toString();
                System.out.println(e.toString());
                return null;
            }
        }
        if (newData == null) {
            newData = new ReportData();
            reportType = null;
        }
        return(newData);
    }
    public String getDataListing() throws SQLException {
        String answer ="<table><form name=\"myname\" action=\"Controller.jsp\" method=\"post\"><TR>";
        answer +="<input type=\"hidden\" value=\"EditReports\" name=\"prevPage\">";
        if (currentReport == null) {
```

```
            return(answer+"</tr>\n");
        }
        Hashtable myHash = currentReport.getData();
        String temp=null;
        DataObject tempData=null;
        if (myHash == null) {
            return(answer+"</tr>\n");
        }
        for (Enumeration e=myHash.keys(); e.hasMoreElements();) {
            temp = (String)e.nextElement();
            tempData = (DataObject)myHash.get(temp);
            answer+="<td bgcolor=\"#d6e4ec\">\n";
            answer+="<p>"+tempData.getSection()+", "+tempData.getLabel()+": ";
            answer+="<input name=\"selectField\" value=\""+temp+"\">";
            answer+="<input type=\"submit\" name=\"action\" value=\"RemoveField\"></p>";
            answer+="</td>";
        }
        answer +="</TR></form></table>";
        return answer;
    }
    public void updateType() throws SQLException {
        String sqlString = "update reportinfo set reporttype = '"+reportType+"' where reportname =
'"+reportName+"'";
        int result = DBHelper.doUpdate(ReportPool.getInstance(), sqlString);
        if (result==0) {
            error="Error in update; report type not changed.";
        }
    }
```

```
public void updateReport() throws SQLException {
    DataObject temp = (DataObject)allData.get(dataObject);
    temp.setLabel(dataLabel);
    temp.setSection(dataSection);
    currentReport.addData(temp);
    DBHelper.doPSUpdate(ReportPool.getInstance(), reportName, currentReport.encode());
}
public void deleteDataObject() throws SQLException {
    DBHelper.doPSUpdate(ReportPool.getInstance(), reportName, currentReport.encode());
}
public void clearReports() {
    reportName = null;
    reportType = null;
    newReportName = null;
    allData = null;
    dataObject = null;
    dataLabel = null;
    currentReport = null;
}
public Hashtable getSections() {
    Hashtable myHash = new Hashtable();
    DataObject temp=null;
    for (Enumeration e = currentReport.getData().elements(); e.hasMoreElements(); ) {
        temp = (DataObject)e.nextElement();
        if ( myHash.containsKey(temp.getSection()) ) {
            Vector newVect = (Vector)myHash.get(temp.getSection());
            newVect.add(temp);
        } else {
```

```
            Vector tempVect = new Vector();
            tempVect.add(temp);
            myHash.put(temp.getSection(), tempVect );
          }
        }
        return myHash;
    }
    public String getDataTables() {
        String answer = "";
        if (idType.equals("manager")) {
            answer = pullData();
        } else {
            String key = "";
            Hashtable myHash = getSections();
            DataObject dob = null;
            for (Enumeration e=myHash.keys(); e.hasMoreElements(); ) {
                key = (String)e.nextElement();
                Vector myVect = (Vector)myHash.get(key);
                answer+="<table border=\"0\" cellpadding=\"0\" cellspacing=\"2\"
width=\"398\"><tr><td>"+key+"</td></tr>\n";
                for (Enumeration ee=myVect.elements(); ee.hasMoreElements();) {
                    dob = (DataObject)ee.nextElement();
                    answer+="<tr><td bgcolor=\"#d6e4ec\">"+dob.getLabel()+"<input type=text size=30
name="+dob.getKey()+"></td></tr>\n";
                }
                answer += "</table>\n";
            }
        }
```

```
        return answer;
    }
    public void addOneSubmit() {
        String sqlString = "select submits from reportusage where reportname = '"+reportName+"'"+
                        " and username = '"+userName+"'";
        String insertString="insert into reportusage (reportname, username, submits) "+
                         "values ('"+reportName+"', '"+userName+"', 0)";
        int submits = 0;
        try {
            ResultSet rs = DBHelper.getQuery(ReportPool.getInstance(), sqlString);
            if ( rs != null && rs.next() ) {
                submits = rs.getInt("submits")+1;
                DBHelper.doUpdate(ReportPool.getInstance(),
                            "update reportusage set submits = "+submits+" where reportname = '"+reportName+"'
and username = '"+userName+"'");
            } else {
                DBHelper.doUpdate(ReportPool.getInstance(), insertString);
            }
        } catch (Exception e) {
            System.out.println(e.toString());
        }
    }
    public String pullData() {
        Hashtable myHash = new Hashtable();
        DataObject temp = null;
        String answer = "";
        String tableString = "";
        String columnString="";
```

```
String dataString="";
Hashtable labelHash=new Hashtable();
//Hashtable inHash = currentReport.getData();
Hashtable sectionHash = getSections();
String section = "";
try {
   // gets a hashtable with key: tablename and value: Vector of dataobjects
   for (Enumeration ex = sectionHash.keys(); ex.hasMoreElements();) {
      section = (String)ex.nextElement();
      Vector sectionList = (Vector)sectionHash.get(section);
      for (Enumeration e = sectionList.elements(); e.hasMoreElements();) {
         temp = (DataObject)e.nextElement();
         if ( myHash.containsKey(temp.getTablename()) ) {
            Vector newVect = (Vector)myHash.get(temp.getTablename());
            newVect.add(temp);
         } else {
            Vector tempVect = new Vector();
            tempVect.add(temp);
            myHash.put(temp.getTablename(), tempVect );
         }
      }
      for (Enumeration e = myHash.keys(); e.hasMoreElements();) {
         tableString = (String)e.nextElement();
         Vector newVect = (Vector)myHash.get(tableString);
         columnString = "";
         dataString = "<tr>";
         String valueString=null;
         for (Enumeration ee=newVect.elements();ee.hasMoreElements();) {
```

```
            temp = (DataObject)ee.nextElement();
            columnString+=temp.getColumnname()+", ";
            dataString += "<td><b>"+temp.getColumnname()+"</b></td>";
            labelHash.put(temp.getColumnname(), temp.getLabel());
         }
         dataString+="</tr>";
         columnString = columnString.substring(0,columnString.length()-2);
         String sqlString="select "+columnString+" from "+tableString;
         ResultSet rs = DBHelper.getQuery(ReportPool.getInstance(), sqlString);
         answer += "<table border=1><tr>";
         ResultSetMetaData rsmd = rs.getMetaData();
         while (rs.next()) {
            dataString+="<tr>";
            for (int x=1; x<=rsmd.getColumnCount(); x++) {
               dataString+="<td>"+rs.getString(rsmd.getColumnName(x))+"</td>";
               //get all column header labels from labelHash using column names from metadata.
            }
            dataString+="</tr>";
            answer+=dataString;
         }
         answer += "</table>";
      }
   }
} catch (Exception e) {
   System.out.println(e.toString());
   answer = "";
}
return answer;
```

```
}
public boolean sendData() {
    Hashtable myHash = new Hashtable();
    DataObject temp=null;
    String tempString="";
    String columnString = "";
    String dataString = "";
    Hashtable inHash = currentReport.getData();
    try {
        for (Enumeration e = inHash.elements(); e.hasMoreElements(); ) {
            temp = (DataObject)e.nextElement();
            if ( myHash.containsKey(temp.getTablename()) ) {
                Vector newVect = (Vector)myHash.get(temp.getTablename());
                newVect.add(temp);
            } else {
                Vector tempVect = new Vector();
                tempVect.add(temp);
                myHash.put(temp.getTablename(), tempVect );
            }
        }
        for (Enumeration e = myHash.keys(); e.hasMoreElements(); ) {
            tempString = (String)e.nextElement();
            Vector newVect = (Vector)myHash.get(tempString);
            columnString = "";
            dataString = "";
            String valueString=null;
            for (Enumeration ee=newVect.elements();ee.hasMoreElements();) {
                temp = (DataObject)ee.nextElement();
```

```
                valueString = (String)temp.getValue();
                if (valueString != null && valueString.length() > 0) {
                    columnString+=temp.getColumnname()+", ";
                    dataString +="'"+valueString+"', ";
                }
                valueString = null;
            }
            columnString = columnString.substring(0,columnString.length()-2);
            dataString = dataString.substring(0, dataString.length()-2);
            String sqlString="insert into "+tempString+" ("+columnString+") "+
                        "values ("+dataString+")";
            DBHelper.doUpdate(ReportPool.getInstance(), sqlString);
        }
      } catch (SQLException e) {
        System.out.println(e.toString());
        error = e.toString();
        return false;
      }
      return true;
   }
}
```

DataObject.java

```
import java.io.*;

public class DataObject implements Serializable {

   private String columnname;
   private String tablename;
```

```
private String label;
private String section;
private Object value;
private int dataType;
private String type;

public String getColumnname() {
   return columnname;
}
public String getTablename() {
   return tablename;
}
public String getLabel() {
   return label;
}
public String getSection() {
   return section;
}
public Object getValue() {
   return value;
}
public int getDataType() {
   return dataType;
}
public String getType() {
   return type;
}
public void setLabel(String inS) {
```

```
    label = inS;
}
public void setSection(String inS) {
    section = inS;
}
public void setValue(Object inO) {
    value = inO;
}
public void setDataType(int inI) {
    dataType = inI;
}
public void setType(String inS) {
    type = inS;
}
public DataObject(String inLabel, String table, String column) {
    columnname = column;
    tablename = table;
    label = inLabel;
}
public DataObject(String table, String column, int dataT, String dataName) {
    columnname = column;
    tablename = table;
    label = column;
    dataType = dataT;
    type = dataName;
}
public DataObject() {
    columnname = "";
```

```
        tablename="";
        label = "";
    }
    public String getKey() {
        return(tablename+":"+columnname);
    }
}
```

DBHelper.java

```
import java.sql.*;


public class DBHelper {


    public static synchronized ResultSet getQuery(ConnectionPool myPool, String inQuery) {
        Connection con = null;
        Statement st = null;
        ResultSet rs = null;
        try {
            con = myPool.getConnection();
            if (con.isClosed()) {
                myPool.closeAllConnections();
                myPool.getConnection();
            }
            st = con.createStatement(   ResultSet.TYPE_SCROLL_INSENSITIVE,
                                        ResultSet.CONCUR_READ_ONLY
                                    );
            rs = st.executeQuery(inQuery);
            return rs;
        } catch (SQLException sqle) {
```

```
            System.out.println(sqle.toString());
            st = null;
            rs = null;
            return rs;
        } finally {
            st = null;
            if (con != null) {
                myPool.free(con);
            }
        }
    } // ends getQuery
    public static synchronized int doPSInsert(ConnectionPool myPool, String prepared, String reportname,
String reporttype, byte[] data) {

        Connection con = null;
        PreparedStatement pstmt = null;
        int result = 0;
        try {
            con = myPool.getConnection();
            if (con.isClosed()) {
                myPool.closeAllConnections();
                myPool.getConnection();
            }
            pstmt = con.prepareStatement(prepared);
            pstmt.setString(1, reportname);
            pstmt.setString(2, reporttype);
            pstmt.setBytes(3, data);
```

```
            result = pstmt.executeUpdate();
            return result;
        } catch (SQLException sqle) {
            System.out.println(sqle.toString());
            pstmt = null;
            return(0);
        } finally {
            pstmt = null;
            if (con != null) {
                myPool.free(con);
            }
        }
    }

    public static synchronized void doPSUpdate(ConnectionPool myPool, String reportName, byte[] data) {

        Connection con = null;
        PreparedStatement pstmt=null;
        try {
            con = myPool.getConnection();
            if (con.isClosed()) {
                myPool.closeAllConnections();
                myPool.getConnection();
            }
            pstmt = con.prepareStatement("update reportinfo set reportdata=? where reportname =
"'+reportName+"'");
            pstmt.setBytes(1,data);
            pstmt.executeUpdate();
```

```
    } catch (SQLException sqle) {
        System.out.println(sqle.toString());
        pstmt = null;
    } finally {
        pstmt = null;
        if (con != null) {
            myPool.free(con);
        }
    }
}

public static synchronized int doUpdate(ConnectionPool myPool, String inQuery) {
    Connection con = null;
    Statement st = null;
    int result = 0;
    try {
        con = myPool.getConnection();
        if (con.isClosed()) {
            myPool.closeAllConnections();
            myPool.getConnection();
        }
        st = con.createStatement();
        result = st.executeUpdate(inQuery);
        return result;
    } catch (SQLException sqle) {
        System.out.println(sqle.toString());
        st = null;
        result = 0;
```

```
        return result;
    } finally {
        st = null;
        if (con != null) {
            myPool.free(con);
        }
    }
} // ends doUpdate

public static synchronized DatabaseMetaData getDBMD(ConnectionPool myPool) {
    Connection con = null;
    DatabaseMetaData dbmd = null;
    try {
        con = myPool.getConnection();
        if (con.isClosed()) {
            myPool.closeAllConnections();
            myPool.getConnection();
        }
        dbmd = con.getMetaData();
        return dbmd;
    } catch (SQLException sqle) {
        System.out.println(sqle.toString());
        return null;
    } finally {
        dbmd = null;
        if (con != null) {
            myPool.free(con);
        }
```

```
        }
    } // ends getDBMD
} // ends DBHelper
```

ReportData.java

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

public class ReportData implements Serializable {

    private Hashtable dataHash;
    private String sqlString;

    public ReportData() {
        dataHash = new Hashtable();
        sqlString = "";
    }
    public void addData(DataObject d) {
        dataHash.put(d.getKey(), d);
    }
    public boolean removeData(String inS) {
        if (dataHash.remove(inS) == null) {
            return false;
        } else {
            return true;
        }
    }
    public Hashtable getData() {
        return dataHash;
```

```
}
public int getNumFields() {
   return dataHash.size();
}
public byte[] encode() {
   byte[] codedReport = null;
   try {
      ByteArrayOutputStream o = new ByteArrayOutputStream();
      ObjectOutputStream oos = new ObjectOutputStream( o );
      oos.writeObject( this );
      oos.flush();
      codedReport = o.toByteArray();
   } catch (Exception e) {
      System.out.println(e.toString());
      codedReport = null;
   }
   return codedReport;
}
public String toString() {
   return(dataHash.toString());
}
public void clearData() {
   DataObject temp = null;
   for (Enumeration e=dataHash.elements();e.hasMoreElements();) {
      temp = (DataObject)e.nextElement();
      temp.setValue(null);
   }
}
```

```
}
```

ReportPool.java

```
import java.sql.*;

import java.util.*;

import java.io.*;


public class ReportPool extends ConnectionPool {


   private static ReportPool pool = null;


   private ReportPool(Properties inProp) throws SQLException {
      super(inProp.getProperty("driver"),
            inProp.getProperty("url"),
            inProp.getProperty("userid"),
            inProp.getProperty("password"),
            Integer.parseInt(inProp.getProperty("connections")),
            Integer.parseInt(inProp.getProperty("max_connections")),
            Boolean.getBoolean(inProp.getProperty("waitifbusy"))
           );
   }
   private ReportPool() throws SQLException {
      this(getProps());
   }
   public static synchronized Properties getProps() {
      Properties myProps = new Properties();
      try {
         myProps.load(new FileInputStream("database.properties"));
      } catch (Exception e) {
```

```
            System.out.println(e.toString());
        }
        return myProps;
    }

    public static synchronized ReportPool getInstance() throws SQLException {
        if (pool==null) {
            pool = new ReportPool();
        }
        return(pool);
    }
}
```

Controller.jsp

```
<%@ include file="JspHeader.jsp"%>
<%
  if ( dataTool.getPrevPage() == null ) {
%>
<jsp:forward page="Login.jsp">
<jsp:param name="error" value="Invalid system state.  Please log in."/>
</jsp:forward>
<%
  }
  if ( dataTool.getPrevPage().equals("Login")) {
    if(!(dataTool.validate())) {
%>
<jsp:forward page="Login.jsp">
<jsp:param name="error" value="Not validated!"/>
</jsp:forward>
```

```
<%
   } // ends bad validation
%>
<jsp:forward page="SelectReports.jsp"/>
<%
  } // ends LogIn page
  // begins ViewReports
  if ( dataTool.getPrevPage().equals("ViewReport")) {
     if (dataTool.getAction().equals("Submit")) {
        Hashtable myHash = dataTool.getCurrentReport().getData();
        dataTool.addOneSubmit();
       for(Enumeration e = myHash.elements(); e.hasMoreElements();) {
          DataObject temp = (DataObject)e.nextElement();
          temp.setValue(request.getParameter(temp.getKey()));
       }
       dataTool.sendData();
%>
<jsp:forward page="ViewReport.jsp"/>
<%
     } else if(dataTool.getAction().equals("EditReport")) {
%>
<jsp:forward page="EditReports.jsp"/>
<%
     } else if(dataTool.getAction().equals("Logout")) {
%>
<jsp:forward page="Login.jsp"/>
<%
     } else if(dataTool.getAction().equals("Clear")) {
```

```
%>
<jsp:forward page="ViewReport.jsp"/>
<%
     } else if(dataTool.getAction().equals("ViewAnotherReport")) {
%>
<jsp:forward page="SelectReports.jsp"/>
<%
     }
  } //ends ViewReports / Starts SelectReports
  if ( dataTool.getPrevPage().equals("SelectReports")) {
    if (dataTool.getAction().equals("ViewReport")) {
%>
<jsp:forward page="ViewReport.jsp"/>
<%
    }
    if (dataTool.getAction().equals("AddReport")) {
        if(dataTool.verifyNewReportName()) {
            dataTool.setReportType("user");
            dataTool.setReportName(dataTool.getNewReportName());
            if (!dataTool.insertReport()) {
%>
<jsp:forward page="SelectReports.jsp"/>
<%
            }
%>
<jsp:forward page="EditReports.jsp"/>
<%
        } else {
```

```
            dataTool.setError("Report Already Exists or bad report name.");
%>
<jsp:forward page="SelectReports.jsp"/>
<%
        }
    } else if (dataTool.getAction().equals("DeleteReport")) {
        dataTool.deleteReport();
%>
<jsp:forward page="SelectReports.jsp"/>
<%
    } else if (dataTool.getAction().equals("EditReport")) {
%>
<jsp:forward page="EditReports.jsp"/>
<%
    } else if (dataTool.getAction().equals("LogOut")) {
%>
<jsp:forward page="Login.jsp"/>
<%
    } else if (dataTool.getAction().equals("CreateUser")) {
%>
<jsp:forward page="CreateAccount.jsp"/>
<%
    }
  } // ends SelectReports, begins EditReports
  if ( dataTool.getPrevPage().equals("EditReports") ) {
      if (dataTool.getAction().equals("LogOut")) {
%>
<jsp:forward page="Login.jsp"/>
```

```
<%
    }
    if (dataTool.getAction().equals("AddField")) {
       dataTool.updateReport();
    }
    if (dataTool.getAction().equals("RemoveField")) {
       dataTool.deleteDataObject();
    }
    if (dataTool.getAction().equals("ChangeType")) {
       dataTool.updateType();
    }
    if (dataTool.getAction().equals("ViewReport")) {
%>
<jsp:forward page="ViewReport.jsp"/>
<%
    }
    if (dataTool.getAction().equals("SelectAnotherReport")) {
%>
<jsp:forward page="SelectReports.jsp"/>
<%
    }
%>
<jsp:forward page="EditReports.jsp"/>
<%
  } // ends EditReports/ begins CreateAccount
if (dataTool.getPrevPage().equals("CreateAccount")) {
   if(dataTool.getAction().equals("EditReports")) {
%>
```

```
<jsp:forward page="EditReports.jsp"/>
<%
   }
   if(!dataTool.validateAccount())
   {
      dataTool.setError("Invalid user Information, or user alread exists. New user not added.");
   }
%>
<jsp:forward page="SelectReports.jsp"/>
<%
} // ends CreateAccount
%>
```

CreateAccount.jsp

```
<%@ include file="JspHeader.jsp"%>
<html>
<head>
<title>Web Database Front End: Create Account</title>
</head>
<br><dir>
<body background="back.gif" text="00000" >
<center>
<table border="1" bgcolor="#FFFFFF">
<tr><td><B><FONT color="#336666" size=6>The Web Database Front End<br>
<center>Create Account</center></B></font></td></tr>
</table></center>
<hr width=85%>
<FONT color="#336666" size=5>
<p><center>Please fill in the following information. All fields are required.<center></font></p>
```

```
</dir>
<form action="Controller.jsp" method="post">
<center>
<table>
<tr><td>User ID:</td>
<td><input type=text name="newUser" size=30 maxlength=30></td>
</tr>
<tr><td>Password:</td>
<td><input type=password name="newPassword" size=10 maxlength=10></td>
</tr>
<tr><td>Re-enter Password:</td>
<td><input type=password name="newPassword2" size=10 maxlength=10></td>
<TD><SELECT NAME="newIdType" SIZE=1>
      <OPTION VALUE="admin">Admin</OPTION>
      <OPTION VALUE="manager">Manager</OPTION>
      <OPTION VALUE="user">User</OPTION>
   </SELECT>
</TD>
<TD><INPUT TYPE="hidden" NAME="prevPage" VALUE="CreateAccount"></TD>
</tr>
</table>
<br>
<input type=submit name="action" value="AddAccount">
<INPUT TYPE="SUBMIT" NAME="action VALUE="EditReports">
<br>
<%@include file="Error.jsp"%>
<em><font size=4>
```

```
Please contact  <a href="mailto:hallr@ils.unc.edu"> Robert A. Hall </a> with any
questions/comments.</font></em>
</center>
</form>
</html>
```

EditReports.jsp

```
<%@ include file="JspHeader.jsp"%>
<%
   dataTool.setPrevPage("EditReports");
   if (dataTool.getNewReportName() !=null) {
      dataTool.setReportName(dataTool.getNewReportName());
      dataTool.setNewReportName(null);
   }
   dataTool.setAllData(dataTool.getDataObjects());
   dataTool.setCurrentReport(dataTool.getReportData());
   String tempString = dataTool.getDataListing();
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
   <title>Database Access Tool: Edit Report Page</title>
</head>
<body bgcolor="#ffffff">
<p>
</p>
<form name="FormName" action="Controller.jsp" method="post">
   <table border="0" cellpadding="0" cellspacing="2" width="435">
      <tr>
```

```
    <td bgcolor="#e4e4e4">
      <p>
      Editing Report Name: <%=dataTool.getReportName()%>
      </p>
    </td>
  </tr>
  <tr>
    <td>
      <hr>
    </td>
  </tr>
  <tr>
    <td>Label:
      <input type="text" name="dataLabel" size="24">
    <td>Section Heading:
      <input type="text" name="dataSection" size="24">
      <select name="dataObject" size="1">
      <%=dataTool.getDataEntries()%>
      </select>
      <input type="submit" name="action" value="AddField">
    </td>
    <TD>Report Type: <%=dataTool.getReportType()%>
      <select name="reportType" SIZE="1">
            <%=dataTool.getReportTypes()%>
      </select>
      <INPUT TYPE="SUBMIT" NAME="action" VALUE="ChangeType">
    </TD>
  </tr>
```

```
        <tr>
          <td>  
            <hr>
          </td>
        </tr>
        <INPUT TYPE="SUBMIT" NAME="action" VALUE="SelectAnotherReport">
        <INPUT TYPE="SUBMIT" Name="action" VALUE="ViewReport">
        <INPUT TYPE="submit" NAME="action" VALUE="LogOut">
</FORM>
<tr>
   <td>  </td>
</tr>
<tr>
   <td bgcolor="#d6e4ec">
      <p>
      <input type="hidden" value="EditReports" name="prevPage">
      </p>
   </td>
</tr>
</table>
<%=tempString%>
<%@ include file="Error.jsp"%>
</body>
</html>
```

Error.jsp

```
<hr>
<%
  String temp = dataTool.getError();
```

```
  if (temp == null) {
    temp = "";
  } else {
    dataTool.setError("");
  }
%>
<CENTER><FONT COLOR="red"><%= temp %></FONT></CENTER>
```

ErrorPage.jsp

```
<%@ page isErrorPage="true" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Database Access Tool:  Error Page</TITLE>
<LINK REL=STYLESHEET HREF="JSP-Styles.css" TYPE="text/css">
</HEAD>
<BODY>
<center>
<br><br><BR><BR>
   <table><tr><th class = "title">Database Access Tool:  Error Page</th></tr></table>
   <p><h3>We apologize for the inconvienience. Please click the button below to continue.</h3></p>
<FORM  ACTION="Login.jsp" METHOD="POST" TITLE="Return">
          <INPUT TYPE="submit" NAME="Return" VALUE="Return to Login Page" ALT="Return to
Login Page">
</FORM>
</table>
<br>
<p>Error: <%=exception%></p>
</center>
```

```
</body>

</html>
```

JspHeader.jsp

```
<%@ page errorPage="/ErrorPage.jsp" %>

<jsp:useBean id="dataTool" scope="session" class="DatabaseAccess"/>

<jsp:setProperty name="dataTool" property="*"/>

<%@ page import="java.sql.*, amp.dbconnect.*, amp.reports.*, java.util.*"%>
```

Login.jsp

```
<%@ include file="JspHeader.jsp"%>

<%

   dataTool.setUserName(null);

   dataTool.setPassword(null);

   dataTool.setAction(null);

   dataTool.setIdType(null);

   dataTool.clearReports();

%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>

<head>

<title>Database Access Tool: Login Page</title>

</head>

<body background="back.gif" text="00000" >

<center>

<h2>Database Access Tool: Login Page</h2>

<br>

<h3> Please type your ID and password to continue. </h3>

<br>
```

```
<table>
<FORM  ACTION="Controller.jsp" METHOD="POST" TITLE="login">
        <TR><td>UserId: </td><td><INPUT TYPE="text" NAME="userName" SIZE="20"
MAXLENGTH="20"></td></TR>
   <TR><TD>Password:</TD><TD><INPUT TYPE="password" NAME="password" SIZE="20"
MAXLENGHT="20"></TD></TR>
   <INPUT TYPE="hidden" NAME="prevPage" VALUE="Login">
   <TR><TD><INPUT TYPE="submit" NAME="action" VALUE="LogIn"></TD></TR>
</FORM>
</table>
<br>
<p>Note that your password is case-sensitive and a maximum of 20 characters long.</p>
</center>
<BR><BR><BR><BR>
<%@ include file="Error.jsp"%>
<em><font size=4>
Please contact  <a href="mailto:hallr@ils.unc.edu"> Robert A. Hall </a> with any questions/comments.
</font></em>
</body>
</html>
```

SelectReports.jsp

```
<%@ include file="JspHeader.jsp"%>
<%
   dataTool.clearReports();
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
```

```
   <TITLE>Database Access Tool: Select Reports</TITLE>
</head>
<body bgcolor="white">
<p>
</p>
<form name="selectReport" action="Controller.jsp" method="post">
   <table border="0" cellpadding="0" cellspacing="0" width="392">
      <tr>
         <td bgcolor="#e4e4e4">
            <p>
            Select Report
            </p>
         </td>
      </tr>
      <tr>
         <td>
            <p>
              
            </p>
         </td>
      </tr>
<%
   if(dataTool.getIdType().equals("admin")) {
%>
      <tr>
         <td bgcolor="#d6e4ec">New Report Name<b>:
            <input type="text" name="newReportName" size="24">
            </b></td>
```

```
        </tr>
        <tr bgcolor="white">
            <td>  </td>
        </tr>
<%
    }
%>
        <tr bgcolor="white">
            <td bgcolor="#d6e4ec">Existing Reports:
                <select name="reportName" size="1">
                <%=dataTool.getReportNames()%>


                </select>
            </td>
        </tr>
        <tr bgcolor="white">
            <td>  </td>
        </tr>
        <tr>
            <td>
                <input type="submit" name="action" value="ViewReport">
                <input type="submit" name="action" value="LogOut">
<%
            if (dataTool.getIdType().equals("admin") ) {
%>
                <input type="submit" name="action" value="AddReport">
                <input type="submit" name="action" value="DeleteReport">
                <input type="submit" name="action" value="EditReport">
```

```
        </td>
      </tr>
      <TR>
        <TD>
          <HR>
        </TD>
      </TR>
      <TR>
        <TD>
          <INPUT TYPE="SUBMIT" NAME="action" VALUE="CreateUser">
<%
        }
%>
      </TD>
      </TR>
   </table>
<%@ include file="Error.jsp"%>
   <p>
   <input type="hidden" value="SelectReports" name="prevPage">
   </p>
</form>
</body>
</html>
```

ViewReport.jsp

```
<%@ include file="JspHeader.jsp"%>
<%
dataTool.setCurrentReport(dataTool.getReportData());
dataTool.setPrevPage("ViewReport");
```

```
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
   <title>Database Access Tool: View Report</title>
</head>
<body bgcolor="#ffffff">
<CENTER>
   <H2>Viewing Report: <%=dataTool.getReportName()%></H2>
</CENTER>
<form name="ViewReport" action="Controller.jsp" method="post">
   <input type="hidden" value="prevPage" name="ViewReport">
         <%=dataTool.getDataTables()%>
           <p>
   <input type="submit" name="action" value="Logout">
   </p>
<%
            if (dataTool.getIdType().equals("manager")) {
%>
   <input type="submit" name="action" value="ViewAnotherReport">
<%
            }
            if (dataTool.getIdType().equals("user")) {
%>
   <input type="submit" name="action" value="Submit">
   <input type="reset" name="actionSubmit" value="Clear">
<%
           }
```

```
            if (dataTool.getIdType().equals("admin")) {
%>
   <input type="submit" NAME="action" VALUE="EditReport">
   </p>
<%
            }
%>
</form>
</body>
</html>
```