

## CIS 58.62C- Final Project Spring 2007

**The purpose of the final project is to use Dreamweaver's built-in dynamic database support to create a fully functional website. This includes:**

- A. Detection page
  - 1. Detect the user's browser (using HHTTP\_USER\_AGENT)
  - 2. Detect what page the user has just come from (using HTTP\_REFERER)
  - 3. Detect the user's IP Address (using REMOTE\_ADDR)
  - 4. Redirects the user to the page appropriate for their browser.
- B. Register
  - 1. Checks for duplicate username
  - 2. Form validation (client-sided)
- C. Login
  - 1. Checks if user is registered and redirects to registration if not.
  - 2. Insert cookie to personalize the user's experience. They can store and retrieve information from the past session with the computer.
  - 3. Insert session variable which starts when the user enters your site and is destroyed when he/she leaves.
- D. Gallery Page (searches the gallery)
  - 1. Master detail page set behavior
  - 2. Link master page to detail page
  - 3. Recordset navigation bar
  - 4. Recordset Navigation status
- E. Image Detail Page
  - 1. Dynamically links to the images
- F. Update database
  - 1. Entry based on access level
  - 2. Add/delete/change records option
  - 3. Upload images to the gallery (\*)
- G. Search the database page and Search Results page (\*)
  - 1. SQL statements and parameterized queries
  - 2. Dynamic List Boxes

(\*) if time allows

**Create the following .php pages for a dynamic website**

- 1. Template page (this one will be a .dwt extension)
- 2. Detection page (index.php)
- 3. Registration and registration failed pages
- 4. Login and login failed pages
- 5. Secret and top secret pages (user groups)
- 6. Master/detail record set (admin only)
  - a. Master page
  - b. Insert Record page
  - c. Update/Delete record page
- 7. Image-based master/detail record set

- a. Image Gallery
- b. Image Detail page

- c. Image Search page
- 8. Log out page

Due May 20, 2007

## ***Final Project Part One – Overview***

You will be using a pre-defined MySQL database that includes the following tables:

### **Table: `tbl_users`**

Fields:           userID(Primary key-  
                  autonumber)  
                  firstName(text)lastName  
                  (text)  
                  username (text)  
                  pwd (text)  
                  address1 (text)  
                  address2 (text)  
                  city(text)  
                  state\_province(text)  
                  zip\_postal(text)  
                  country(text)

### **Table: `tbl_images`**

Fields:           photoID(primary key-  
                  autonumber)  
                  imageTitle (text)  
                  artistName (text)  
                  description of image  
                  (memo)  
                  imagePath (text)

### **Step One: Lay out a page design with the following characteristics:**

- 1. Consistent layout
- 2. Text links
- 3. Logo/banner
- 4. Logout option

### **Step Two: Create a template**

- 1. Launch Dreamweaver and create a php page which will eventually be your template.
- 2. Use a css template from [layoutgala.com](http://layoutgala.com), another site or create your own page
- 3. BEFORE you save this page into a template add any META tags/data or JavaScript into the head
- 4. Select File> Save as> Template.
- 5. Name it FinalProject.dwt.

## ***Final Project Part Two –Detecting and Redirecting the User Agent***

In this section you will create:

- Detection page (index.php)
- Public access page (register.php)

When you're developing commercial sites, it can be very useful to know which

browser your visitors are using. If you've spent time trying to develop nice looking pages for both Mozilla and Internet Explorer, you'll know that it's sometimes much easier to develop separate pages for each browser and creating a detection/redirect page allows you to do just that. We can find out this: information easily using the **ServerVariables collection**, so let's try this out.

### **Detecting which browser is viewing your page**

1. Create a new page from your template and save it as index.php.
2. Write "Hello your current browser is" on your index page
3. Open the Data Bindings window.
4. Select + Server Variable from the drop down menu.
5. Type in HTTP\_USER\_AGENT under Name:
6. Drag and drop the request variable onto your page, after the Hello your current browser is text:
7. Save the page and open it in your browser. The page will now tell you what browser you are running.

The output of a typical web page would look something like:

*Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.3) Gecko/20070309  
Firefox/2.0.0.3*

### **Detecting which page the user has come from:**

1. Write "You have come from" on your index page
2. Open the Data Bindings window.
3. Select + Server Variable from the drop down menu.
4. Type in HTTP\_REFERER under Name:
5. Drag and drop the request variable onto your page, after the "You have come from" text. This will only show a value if the user clicked on a link from another website that sent them to your page.

### **Detecting the user's IP address and redirecting:**

1. Write "Your IP address is" on your index page
2. Open the Data Bindings window.
3. Select + Server Variable from the drop down menu.
4. Type in REMOTE\_ADDR under Name:
5. Drag and drop the request variable onto your page, after the "Your IP address is" text:
6. The page will now tell you the user's IP address.
7. Hit the return key and type in Continue to the registration page or wait 5 seconds to be sent automatically
8. Link the underlined text above to your public access page.
9. Add the following code right after the closing </Title> tag in your detection page:  
<meta http-equiv="refresh" content="5;URL=register.php" />
10. Save and test your page.

## **Final Project Part Three – Implementing User Authentication**

In this section you will [download](#) to your defined final project site, and modify the following pages:

- Registration and registration failed pages
- Login and login failed pages
- Secret and top secret pages (user groups)
- Log out page

Based on an article by Jeffrey Bardzell available at [http://www.adobe.com/devnet/dreamweaver/articles/auth\\_users.html](http://www.adobe.com/devnet/dreamweaver/articles/auth_users.html)

Implementing a framework that allows users to register and log in is one of the most common tasks for web developers. Pages in the site are divided into those that are publicly accessible and those that require users to log in. A set of registration and log-in pages allow users to get through the barriers and access pages requiring authentication. Creating a user authentication framework is the centerpiece of this tutorial. Thanks to a series of Dreamweaver server behaviors, creating an authentication framework is much easier than you might think.

In this tutorial, you'll create a complete registration and log-on system, taking into account both when things go right and when they go awry (for example, log-on fails or a username is already in use). You will work on eight pages, five of which are specifically related to registration and log-on, two of which are protected, and the last one is simply an index page. Once you have a system like this set up, you can add as many restricted pages as you like. You can even implement roles-based authorization, in which users can access only certain pages in the site, even after they have logged in.

### **User Authentication as a Web Application**

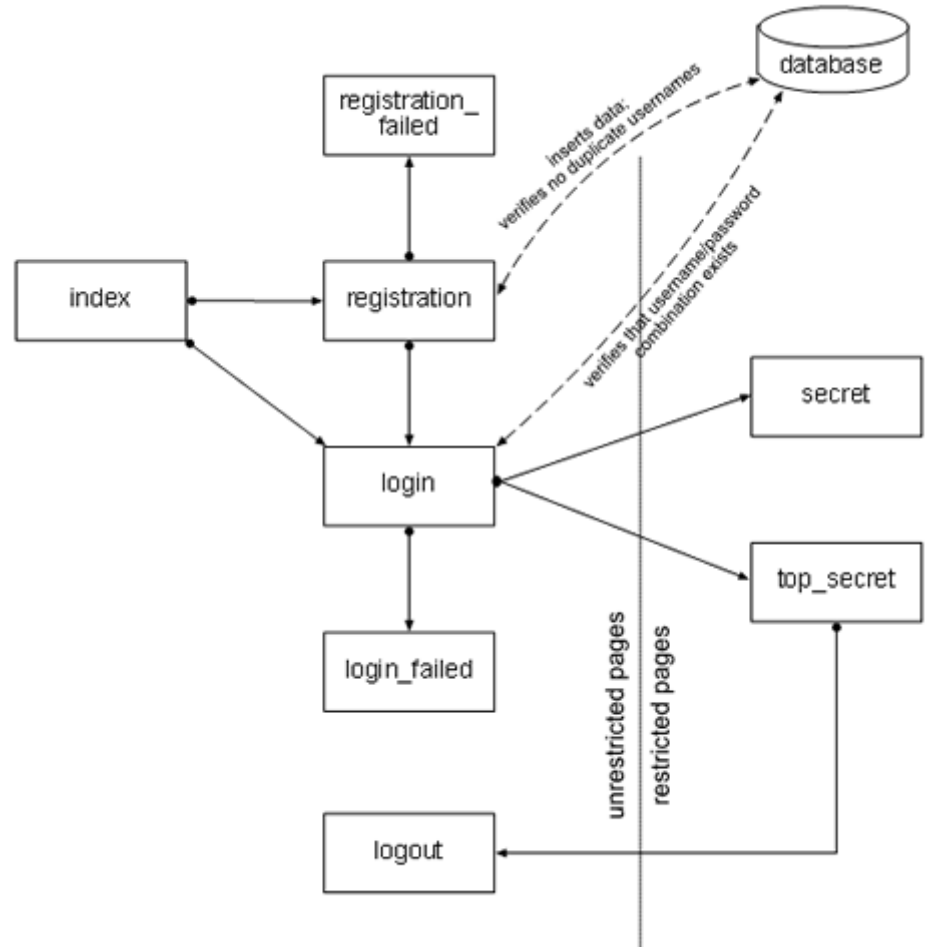
Before jumping in and starting to develop an application, you should understand exactly what your goals are and how it's all going to work. Users can only access restricted pages once they've successfully logged in. A database holds all the user profiles—including the usernames and passwords—of all the registered users.

Users must register at the site before they can log in. More specifically, users must have a profile in the database. The database in this tutorial contains only one table, a users table, which contains a half dozen or so profiles. To register, users fill out a web form, which inserts their registration information, including a username and password, into the database, thus creating the profile. In the event that a user chooses an already existing username, the registration fails, and redirects the user to a page that tells him that the name isn't available. Once registered, a user can log in.

To log in, a user supplies a username and password combination, again in a web form. ASP, ColdFusion, or PHP queries the database to see if any records contain both the username and the password together. If it does, the log-in is successful, and the application flags the user as having logged in successfully (more on that in a moment). Once flagged, the user can access any number of restricted pages; that is, a user only has to log in once per site. If there are no records with both the username and password, the log-on fails, and the application redirects the user to a page that indicates that the log-on failed.

Once logged in, users can also log out by clicking a link. Once logged out, the application redirects them to a page that indicates their logged out state.

The following diagram shows the layout of the site relative to this login framework. Solid lines represent the movement of the user through the site. Dashed lines represent the exchange of data between pages and the database.



Now that you understand the basics of the framework, you might wonder how the pages with restricted access determine whether the user logged in, and therefore whether to display the restricted content. Each restricted page has a script at the top that verifies whether the user logged in successfully, and if so, processes and displays the page. If the user has not logged in, the application typically redirects the user to a page that enables log on; the restricted page never loads.

## Introducing Sessions and Session Variables

The tricky part in the user authentication framework is how to flag a user across all website pages as logged in or not logged in, especially given your knowledge that the server forgets users and all their data in between every page request. In some cases, you may have overcome the server's forgetfulness by sending URL (or query string) and form variables between pages; this is a core capability of ASP, ColdFusion, and PHP. But

these solutions are temporary, requiring you to manually send data to every page that needs it. It is better to use something more persistent.

One type of persistent variable is the cookie. A cookie is a name-value pair stored on the user's hard drive and sent to the server when the user makes an HTTP request. Using this technique, you can simulate the effect of the server remembering a user across multiple pages, simply by having pages use and respond to cookie data.

In large part to facilitate applications that need to keep track of users and data over time, ASP, ColdFusion, and PHP have special built-in features that handle much of this work for you. One of these is the *session* variable scope. Session variables contain information for a single user in a single session. If the user leaves the site or closes the browser for a period of time (usually about 20 minutes), then the session ends, and the application flushes the information from the server's memory.

The session variable is perfect for handling the login flag. That is, once a user successfully logs in, the application sets a session variable indicating the successful log-on. Each page that restricts access needs merely to check for the presence of this session variable to determine whether to grant access to the page or redirect the user to a different page.

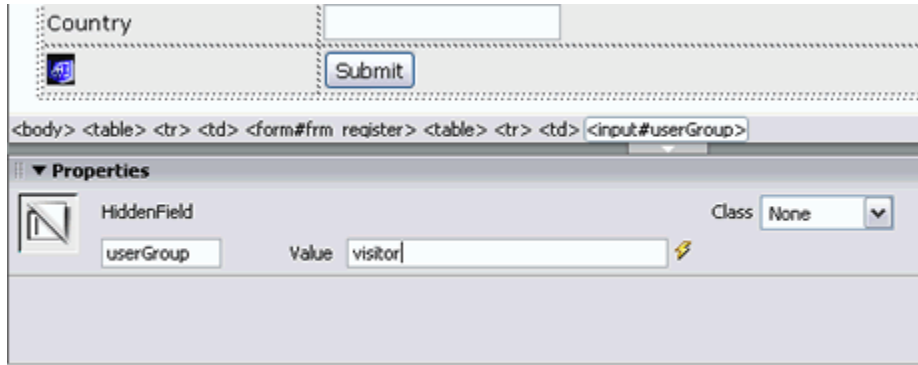
For the data to be persistent, you must store it somewhere. You typically store session variables as cookies on the user's computer and match them to temporary session variables stored on the server. The storage of data in two locations and its subsequent matching is usually invisible to the developer; that is, while you may set and retrieve session variables, for example, you won't be both setting and retrieving server and cookie variables, because that happens behind the scenes.

From even this brief overview, you can understand why a session management framework is so important to authentication. It enables a single entry through a login page to provide access to multiple pages, without requiring the developer to manually create scripts that send the data from each page to the next.

## Implementing Registration

Log-in presupposes a collection of valid usernames and passwords in the database. One way to get those combinations into the database is to create a registration form that inserts them there. I have created the form for you:

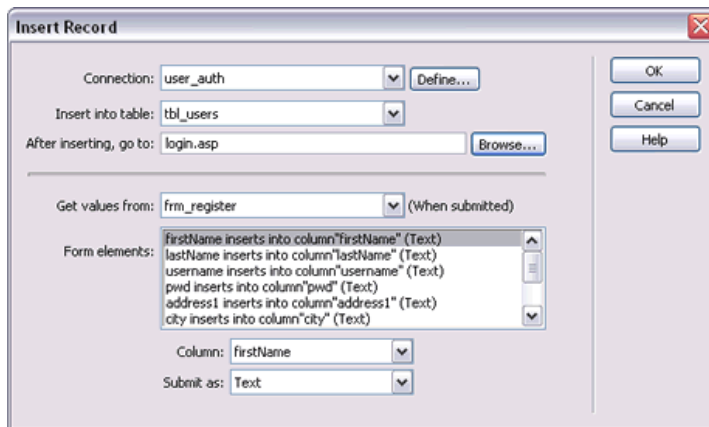
1. Open register.php in Dreamweaver. I have already created the web form.
2. Take a moment to click each of the form elements and find out their names in the Property inspector. Each item entered into a form field is stored as a variable using the name of the field. The first form field is the firstName field. If you type **Jeffrey** into the field and submit the form, the page submits a piece of data, `firstName=Jeffrey`.
3. Click the hidden form field (the yellow icon) in the lower-left corner of the form. In the Property inspector, name it **userGroup** and give it a value of **visitor**.



**Figure 2.** The hidden form field as shown in the Property inspector

Sometimes, you will need to pass data in a form that you don't want to collect from—or even provide access to—the user. This is one such situation. This user authentication framework will use roles-based authorization. Specifically, any registered user will be able to access some, but not all of the pages in the site. Some of the pages are restricted so that only special registered users can access them (think of admin pages, for example). One of the fields in the users table of the database is `userGroup`, which specifies whether the user has basic or full access to the site. Obviously, you don't want to let the users determine whether they should have basic or full access. By creating this hidden field, you assign everyone to the basic group by default. An administrator will have to manually upgrade a given user's status, if she should be in the full-access group. The form is now ready; it's time to add the script.

4. Click the Submit button. In the Server Behaviors panel, click the New Server Behavior (+) button, and choose Insert Record. The Insert Record dialog box appears. The Insert Record server behavior enables you to capture data from a web form and insert it as a new record in a given database table.
5. In the Insert Record dialog box, choose `user_auth` as the Connection (or whatever you named your database connection). Verify that you selected `tbl_users` in the Insert Into Table field. In the After Inserting Go To field, enter `login.php`. Accept the rest of the defaults and click OK.

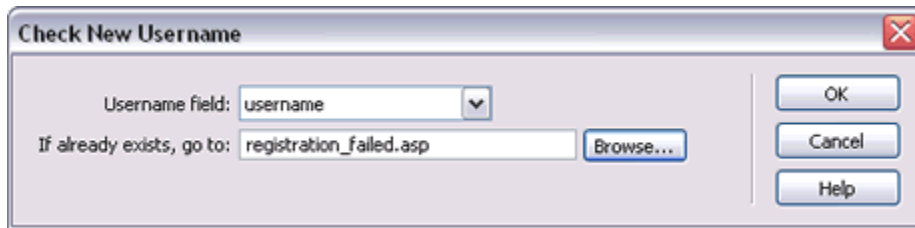


**Figure 3.** The Insert Record dialog box

The lower half of the dialog box enables you to specify which form data your application should insert into which database field; it is like a matching game. For example, you want the data from the firstName web form field to go into the firstName column in the database. If—as is the case here—you name the form fields the same as the database fields, Dreamweaver guesses by default that they are a match, and therefore, you don't have to manually specify each one.

With this server behavior in place, you have made it possible for users to create new profiles. However, you have not taken into account that two users might create profiles under the same username; you want to make this impossible.

6. Click to select the Submit button again. In the Server Behaviors panel, click the New Server Behavior (+) button, and choose User Authentication > Check New Username. The Check New Username dialog box appears.
7. In the Check New Username dialog box, select username (or FORM.username, as ColdFusion is likely to display it) from the Username field pop-up menu and type **registration\_failed.php** in the If Already Exists, Go To field. Click OK.



**Figure 4.** The Check New Username dialog box

Since you asked for the user's e-mail address to use as their username, this shouldn't be a problem for legitimate users.

Whenever you create a form, make sure you deploy some sort of form validation, whether it's a client-side JavaScript behavior or a server-side script. In this case, I have already added a Validate Form behavior (a regular JavaScript behavior found in the Behaviors, not Server Behaviors, panel) to fulfill this function.

8. Save and if necessary, upload, register.php. Close the file. If you like, you can test this component of the site now or you can wait until you have added the login component.

## Implementing a Log-on

Now that your registration page functions, you can build the login page. Remember, the role of the login page is to obtain (through a form) the user's username and password. Then, it will compare these values with the records in tbl\_users. If there's a match, the login script sets a session variable indicating that the user is logged in. If there is no match, the user is redirected to a login failed page.

1. Open and explore login.php. Like registration.php, this page contains a web form. However, this web form has only two fields—one for username and one for password.
2. Click to select the Submit button, and use the Server Behaviors panel to begin adding a User Authentication > Log In User behavior. This behavior will do all the work of both verifying that the user's credentials match a pair in the database; that a session variable will be set, if the log-on was successful; and that the user will be redirected to an appropriate location, based on whether the log-on was successful.
3. In the top quarter of the Log In User dialog box, verify that frm\_login is the selected form, that username is set as the Username Field, and that pwd is set as the Password field. In the second quarter of the dialog box, choose user\_auth as the Connection, tbl\_users as the Table, username as the Username Column, and pwd as the Password Column.

With these settings, you provide parameters that the script needs to compare the authentication entered into the login form with the list of registered users in the database.

4. In the third quarter of the Log In User dialog box, enter secret.php for the successful login redirection, and login\_failed.php

for the failed login redirection. Ensure that Go To Previous URL is checked. In the fourth quarter of the dialog box, select Username, Password, and Access Level, and choose userGroup in the Get Level From menu. Click OK.

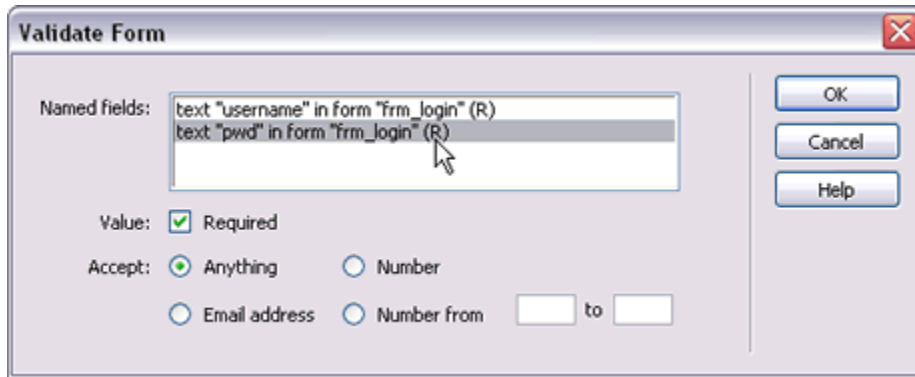
In this step, you are accomplishing two goals. First, you are specifying where the user should be redirected depending on the success or failure of the log-on.

The Go To Previous URL setting needs some explanation. There are two ways that users will access this login page. First, they can access it directly by following the Log In link on the homepage. But they'll also see this page if they

try to access a page that requires log-on (such as secret.php), and they haven't logged in yet. That is, the login page may intercept their progress in the site. Upon successful log-on, users won't want to be redirected to the homepage—not when they were requesting secret.php. By checking this option, you add functionality to the script that ensures that once these users have logged in, they are redirected to the page they attempted to load before they logged in.

The second goal you are accomplishing is to create the separate user group functionality discussed earlier. This will enable you to distinguish between basic access users and full access users who have logged in. Since these access levels are stored in the userGroup field of tbl\_users, you specify that information in the Get Level From menu.

5. Click the Submit button. In the Behaviors panel, click the New Behavior (+) button and add a Form Validation behavior that makes both fields required.



**Figure 6.** The Validate Form dialog box

Every form needs some mechanism for validation.

6. Save and upload the page.

Open register.php, and create a profile. Then log in with any e-mail address as a username and password. As always, try to break the application as well. Register the same name twice. Try logging in with the wrong password, try typing your phone number instead of your e-mail address, and so on.

## Site Registration

Please take a few moments to register for the site.

First Name	<input type="text" value="Erebus"/>
Last Name	<input type="text" value="Bardzell"/>
Email Address	<input type="text" value="erebus@allectomedia.com"/>
Password	<input type="text" value="erebus"/>
Address	<input type="text" value="100 Morpheus Cave Rd."/>
City	<input type="text" value="Bloomington"/>
State/Province	<input type="text" value="IN"/>
Zip/Postal	<input type="text" value="47000"/>
Country	<input type="text" value="USA"/>
<input type="button" value="Submit"/>	

**Figure 7.** The Site Registration page

## Restricting Access to Pages

By now, you have tested your registration and login pages, and they should work as expected. The only thing is, until you implement page restriction features to the pages you want to block access to, your registration and login framework is not very useful. In this task, you will add the server behaviors that prevent users from accessing pages, unless they've first logged in.

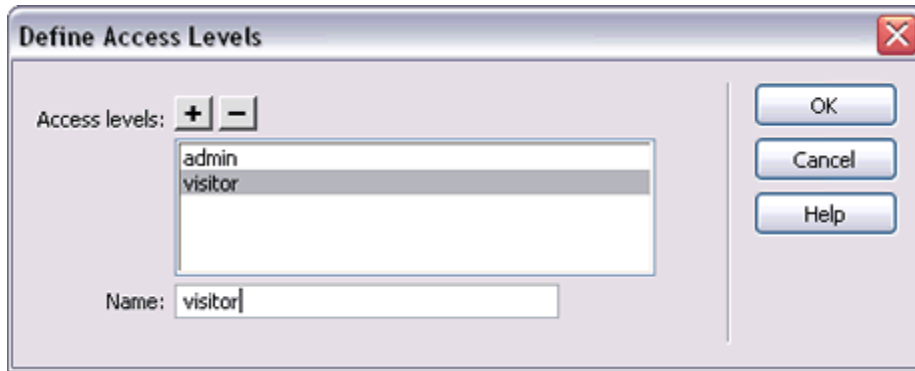
1. Back in Dreamweaver, open secret.php.

This is one of the pages that users must log in to see.

2. Click anywhere on the page. In the Server Behaviors panel, insert a User Authentication > Restrict Access to Page server behavior. In the Restrict Based On group, choose Username, Password, and Access Level. This dialog box not only lets you restrict access to the page; it also lets you restrict access to page based on a user's access level. Of course, since no such levels are defined in the Select Level(s) area, you must define some.

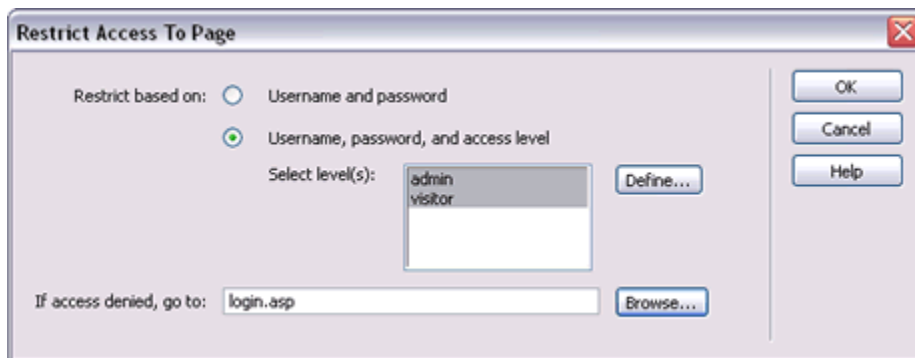
**Figure 9.** The Restrict Access To Page dialog box

3. Click the Define button. In the Define Access Levels dialog box, place the cursor in the Name field, type visitor, and press the + button. Repeat the process to add admin. Click OK. Dreamweaver won't check to make sure these groups actually exist; it takes your word for it, so make sure you spell them correctly. These correspond to the available values in the userGroup field of tbl\_users. Once Dreamweaver knows the users' names, Dreamweaver can grant access to pages to users in either or both groups, and deny access to users not in either.



**Figure 10.** The Define Access Levels dialog box

4. Back in the Restrict Access to Page dialog box, Ctrl-click (Windows) or Command-click (Macintosh) to select both visitor and admin from the Select Level(s) area. In the If Access Denied, Go To field, enter login.php. Click OK.



**Figure 11.** The Restrict Access To Page dialog box

You've done two things in this step. You've granted access to the page to users in either the visitor or admin group. Had you wanted to grant access to this page to only one of those groups, you would have selected only the one group. Once you've created the admin section of the site, you'll use this dialog box and to permit only members of the admin group.

The other thing you've done is redirect the user to login.php if access is denied. This is how that interception described earlier happens. A user tries to access a restricted page without logging in, and she or he sees the login dialog box. Once log-on is achieved, the restricted page she or he was trying to access appears.

5. Repeat steps 1-4 for top\_secret.php, except grant access to top\_secret.php only to users in the admin group.

This page now requires authentication as well.

6. Save and upload all of the pages you have worked on in this lesson. Starting from the homepage (index.php), try directly accessing secret.php or top\_secret.php.

The authentication framework is fully functional. If you try to access a protected page, the login screen will appear. If you've created a registration account, use it. Or, remember you can use username: osiris@allectomedia.com and password: osiris to test. Once you've logged in, the application will automatically redirect you to the page you first requested.

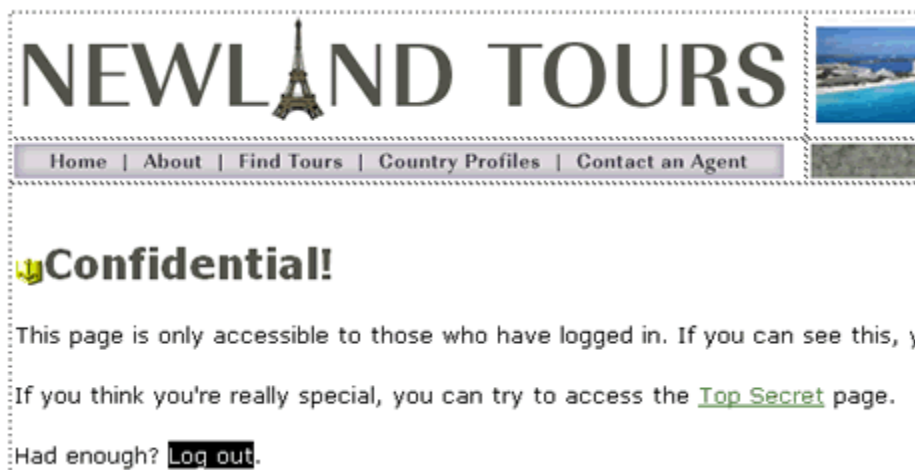
The log-on that you created yourself should let you into secret.php, but not top\_secret.php. Only admin users can access that page. On the login page, try the following credentials to get into top\_secret.php.

Username: jdiamond@santarosa.edu  
Password: 12345

## Implementing Log Out

Users may also want to log out. For example, imagine your site had confidential financial information, or other personal information. Enabling log out is easy in Dreamweaver, though users often need to close their browsers in addition to logging out to fully kill the session.

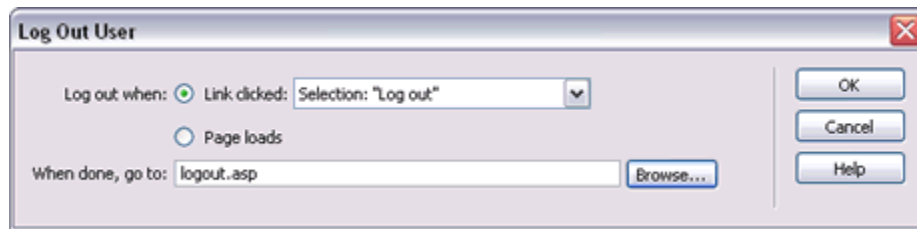
1. Back in Dreamweaver, open secret.php. This page already has some text on it about logging out, but it is not yet functional.
2. Drag to select the words log out. Add a User Authentication > Log Out User server behavior.



**Figure 12.** The access denied page in the browser

The Log Out User dialog box opens.

3. In the Log Out User dialog box, verify that Link Clicked “Log Out” is selected in the Log Out When category. In the When Done Go To field, type logout.php.



**Figure 13.** The Log Out User dialog box

Clicking this link redirects the user to logout.php, and it also removes the current session from the server's memory.

Unfortunately, since browsers often remember (the proper term is “cache”) the contents of recently visited pages, it is often possible for the user to press the Back button and access sensitive content again, even after logging out. Therefore, the user should also close the browser. Logging out and closing the browser ultimately ends the session. A note has been added to this effect on logout.php.

4. Repeat steps 1-3 with top\_secret.php.

The login framework is now completely functional, from beginning to end. You can add as many pages as you want and restrict them by role as appropriate.

## ***Final Project- Part Four- Creating Image List and Image Detail Pages***

In this section you will create the following pages:

- Image-based master/detail record set
  - Image Gallery (gallery.php)
  - Image Detail page (imageDetail.php)
  - Image Search page (imageSearch.php)

The creation of these pages will include the following elements:

- Master detail page set behavior
- Links from the master page to detail page
- Recordset navigation bar
- Recordset Navigation status
- Dynamically Alternating Rows extension

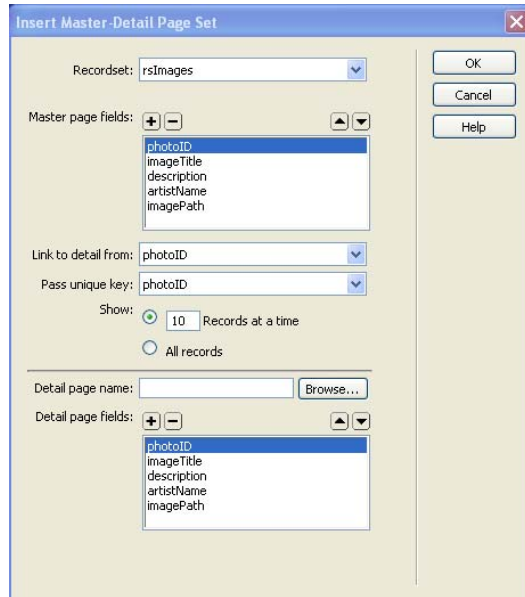
### **The Master-Detail pages**

Master and detail pages are sets of pages used to organize and display recordset data. These pages provide a visitor to your site with both an overview and a detailed view. The master page lists all of the records and contains links to detail pages that display additional information about each record.

You can build master and detail pages by inserting a data object to create a master page and detail page in one operation or by using server behaviors to build the master and detail pages in a more customized way. When using server behaviors to build master and detail pages, you first create a master page to list the records and then add links from the list to the detail pages. After you select a few options, the Master-Detail Page Set application object generates the forms and scripting for both the master list and the page containing the details for you. Additionally, the application object creates record navigation and a record counter in the master page.

After you select a few options, the Master-Detail Page Set application object generates the forms and scripting for both the master list and the page containing the details for you. Additionally, the application object creates record navigation and a record counter in the master page.

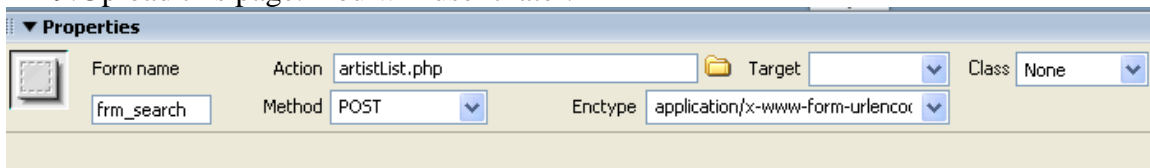
1. Make sure you are connected to your database.
2. Create new page from template and save it as gallery.php
3. Create a recordset from your images table, calling it rsImages. You can bring in all the fields from this table or just the ones that show up in your master detail page set.
4. Choose Insert > Data Objects > Master Detail Page Set. The Insert Master Detail Pages Set dialog box appears.
5. In the dialog box, in the Recordset pop-up menu, select rsImages.
6. In the Master Page Fields list, click the minus (-) button to remove any fields from the list of data which you do not want to appear in the master page list.
7. In the **Link To Detail From** pop-up menu, select **artistName**.
8. To view the detail page for an image, a link needs to be created for each image that appears in the master list. When a site visitor clicks the artistName in the master page the appropriate detail page will open.
9. In the Pass Unique Key pop-up menu, accept the default value, **photoID** and make sure Numeric is not checked.
10. In Show, set it to display **5** records at a time.
11. In **Detail Page Name**, click Browse, then in the **Select File** dialog box, navigate to the **imageDetail** file in your site folder.
12. In the Detail Page Fields, set the fields you want to appear in the detail page, by doing the following: click the minus (-) button for whichever fields you do not want to see. The dialogue box should look similar to the one below.



13. Dreamweaver updates the master and detail pages, and adds all the necessary server scripts and queries for both the list page and the detail page.
14. The master list page updates. It includes a table for the database data, a table for recordset page navigation, and a record set counter.
15. The detail page also updates. It includes a table that lists details for each record in the master page.

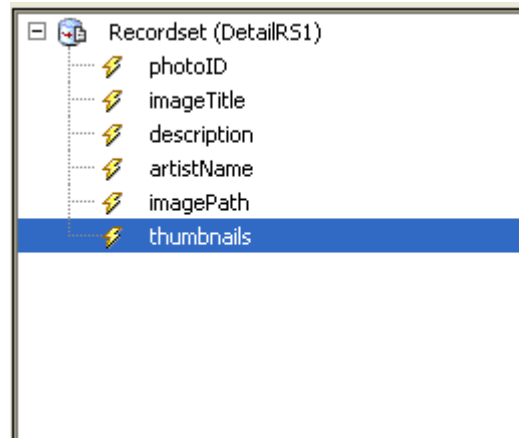
### Insert a Search by Artists Form

1. Remove the Repeat Region Behavior from the Server Behaviors panel
2. Add two rows below the Image Information Row
3. Re-select the Image info row tag in the tag selector bar at the bottom of the document
4. Add the Repeat Region behavior to the **row only**
5. Merge the cells in the row directly beneath and place the “Show If” Navigational Information into the row
6. Merge the cells in the bottom row and insert a form tag called frm\_search
7. Add a text field called artistName and add a search button in this form on the bottom row
8. Select the form tag and place artistList.php in to browse to page box
9. Upload this page. You will use it later.

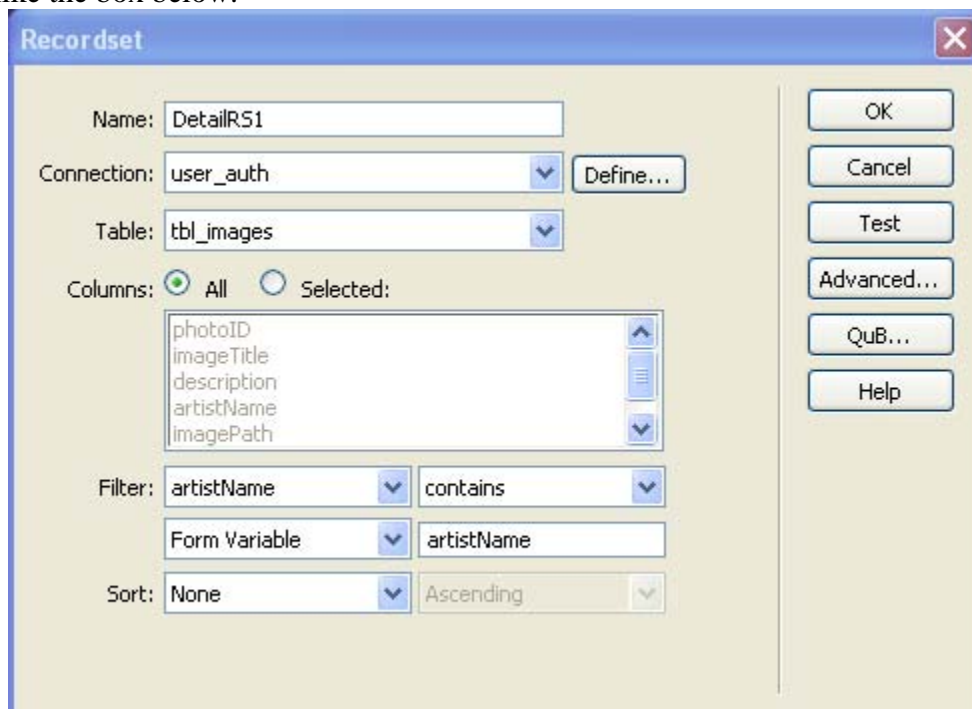


10. Select the imageDetail.php page and save it as artistList.php
11. Select the bottom row of the table called imagePath and rename it Thumbnail

12. Double-click on the image with the lightning bolt and select “thumbnail as the field to bind.



13. Double-click the Recordset in the Server Behaviors panel and make sure it look like the box below.



14. Upload the file.

15. Select the gallery.php file and test the search functionality.

### **Final Project- Part Five Create Management Pages (admin only)**

Now that you have a working gallery, we will develop the necessary pages for inserting, updating and deleting gallery information. In addition, administrators may want to be able to update their member databases. We will transfer the update gallery functionality to the update Users.

In this section you will create the following pages:

- Insert Gallery Record (insertGalleryRecord.php)

- Update Gallery Record (updateGalleryRecord.php)
- Delete Gallery Record (deleteGalleryRecord.php)
- Update Users Page (updateUsers.php)

### **Insert Gallery Record (insertGalleryRecord.php)**

An insert page consists of two building blocks:

- An HTML form that lets users enter data
- An Insert Record server behavior that updates the database

When a user clicks Submit on a form, the server behavior inserts records in a database table. You can add these building blocks in a single operation using the Record Insertion Form data object or you can add them separately using the Dreamweaver form tools and the Server Behaviors panel.

*Note: The insert page can contain only one record-editing server behavior at a time. For example, you cannot add an Update Record or a Delete Record server behavior to the insert page.*

The Insert Gallery page offers the possibility to add a new gallery records to the database. The actual insert action and form will be generated automatically through the use of the "Insert Record Form Wizard"

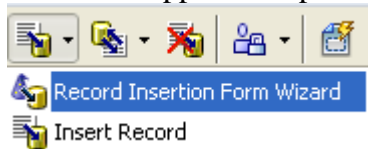
To build the insert page, you will have to follow the next steps:

1. Open the index.php page, clear out the Server behaviors and save the page as insertRecord.php
2. Click inside an empty place where the insert form will be placed.
3. Use the "Insert Record Form Wizard" to add both HTML form and application logic to the page. Configure the user interface for your own table and fields.
4. When done configuring, click the OK button to add the elements into the page.

### **Configure the Insert Record Form Wizard**

The "Insert Record Form Wizard" has three steps.

1. Select the Application panel Record Insertion Form Wizard icon.



- Fill out the form so it looks like the image below and click OK. Make sure to remove the photoID field by select the – button next to Form fields.

Record Insertion Form

Connection: user\_auth

Table: tbl\_images

After inserting, go to: gallery.php

Form fields:

Column	Label	Display As	Submit As
imageTitle	ImageTitle:	Text field	Text
description	Description:	Text field	Text
artistName	ArtistName:	Text field	Text
imagePath	ImagePath:	Text field	Text
thumbnails	Thumbnails:	Text field	Text

Label: PhotoID:

Display as: Text field

Submit as: Numeric

Default value:

OK, Cancel, Help

- The Labels will be automatically recognized from the field names if they match with the form names.
- Upload and text your form. It is as easy as that!

## About record update pages

Your application can contain a set of pages that lets users update existing records in a database table. The pages normally consist of a search page, a results page, and an update page. The search and results page let users retrieve the record and the update page lets users modify the record.

## Complete the update page in one operation

An update page has three building blocks:

- A filtered recordset to retrieve the record from a database table
- An HTML form to let users modify the record's data

- An Update Record server behavior to update the database table

You can add the final two building blocks of an update page in a single operation using the Record Update Form data object. The data object adds both an HTML form and an Update Record server behavior to the page.

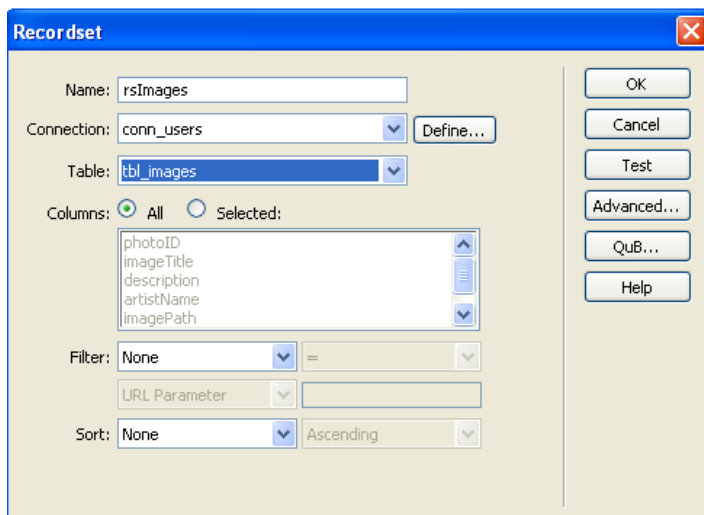
Before you can use the data object, your web application must be able to identify the record to update, and your update page must be able to retrieve it.

After the data object places the building blocks on the page, you can use the Dreamweaver design tools to customize the form to your liking, or the Server Behaviors panel to edit the Update Record server behavior.

Note: The update page can contain only one record-editing server behavior at a time. For example, you cannot add an Insert Record or a Delete Record server behavior to the update page.

1. Open your index page and delete all the Server behaviors and any meta-refresh tags you may have placed in the head.
2. Save the page as updateGalleryRecord.php
3. Select Insert > Data Objects > Update Record > Record Update Form Wizard.

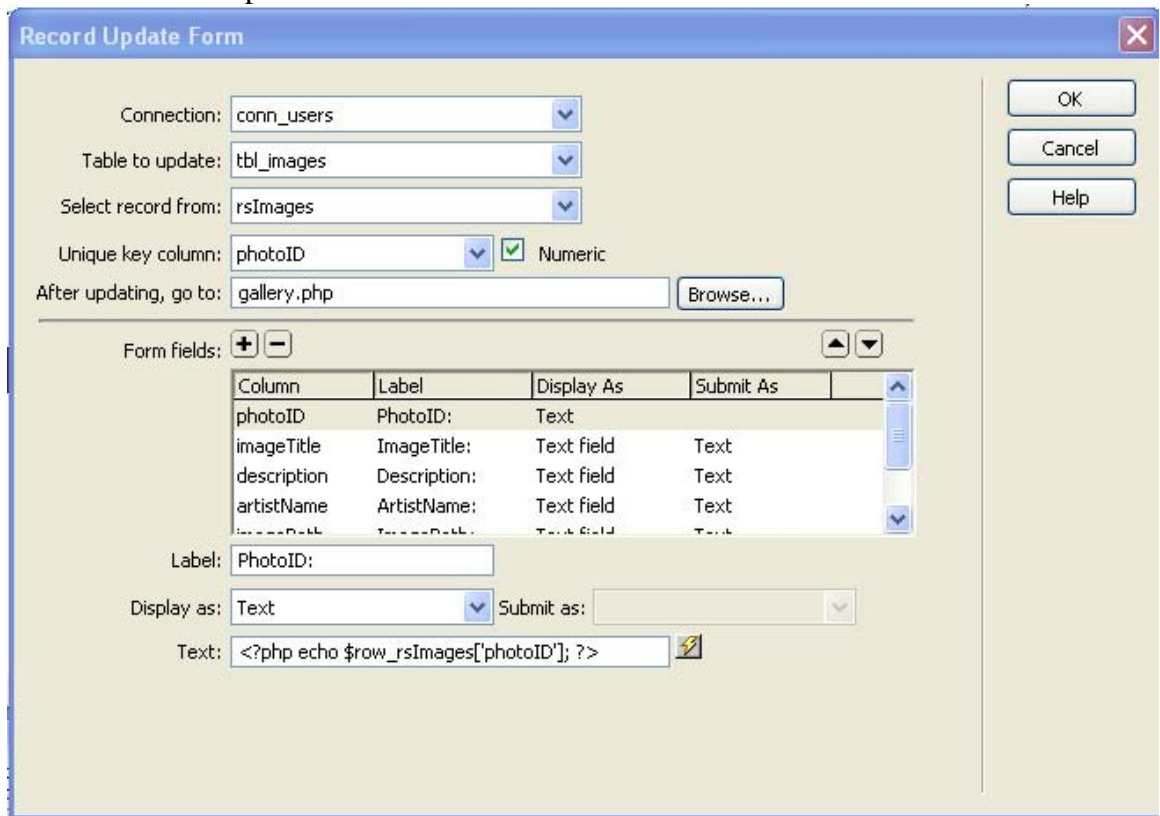
You will be asked to define a Recordset for this page. Create a recordset called rsImages and click OK.



4. The Record Update Form dialog box appears.
5. In the Connection pop-up menu, select a connection to the database. Click the Define button if you need to define a connection.
6. In the Table to Update pop-up menu, select the database table that contains the record to update.
7. In the Select Record From pop-up menu, specify the recordset that contains the record displayed in the HTML form.
8. In the Unique Key Column pop-up menu, select a key column (usually the record ID column) to identify the record in the database table. If the value is a number,

select the Numeric option. A key column usually accepts only numeric values, but sometimes it accepts text values.

9. In the After Updating, Go To box enter gallery.php
10. In the Form Fields area, specify which columns in your database table each form object should update.
11. Click OK. Dreamweaver adds both an HTML form and an Update Record server behavior to your page. The data object adds both an HTML form and an Update Record server behavior to your page. The form objects are laid out in a basic table, which you can customize using the Dreamweaver page design tools. (Make sure all the form objects remain within the form's boundaries.) To edit the server behavior, open the Server Behaviors panel (Window > Server Behaviors) and double-click the Update Record behavior.



12. In order for the Update page to function properly we have to pass it a URL parameter so it knows which record to update.
13. Double-click on the Recordset (rsImages) and select Filter: photoID = URL parameter photoID.
14. Select gallery.php page highlight the text in the row under Image Title.



15. With the text highlighted, click the + sign in the server behaviors panel and select Go to Detail page and fill it out as per the image below.

Go To Detail Page

Link: "<?php echo \$row\_rsImages[\"imageI

Detail page: updateGalleryRecord.php Browse...

Pass URL parameter: photoID Set to the Value Of

Recordset: rsImages

Column: photoID

Pass existing parameters:  URL parameters  
 Form parameters

OK  
Cancel  
Help

16. Upload both the gallery.php and the updateGalleryRecord.php pages.

## About record delete pages

Your application can contain a set of pages that lets users delete records in a database. The pages normally consist of a search page, a results page, and a delete page. A delete page is usually a detail page working in tandem with a results page. The search and results pages let the user retrieve the record and the delete page lets the user confirm and delete the record. After creating the search and results pages, you add links on the results page to open the delete page and then build a delete page that displays the records and a Submit button.

## Build the delete page

After completing the page listing the records, switch to the delete page. The delete page shows the record and asks the user if they're sure they want to delete it. When the user confirms the operation by clicking the form button, the web application deletes the record from the database. Building this page consists of creating an HTML form, retrieving the record to display in the form, displaying the record in the form, and adding the logic to delete the record from the database. Retrieving and displaying the record consists of defining a recordset to hold a single record—the record the user wants to delete—and binding the recordset columns to the form.

*Note: The delete page can contain only one record-editing server behavior at a time. For example, you cannot add an Insert Record or an Update Record server behavior to the delete page.*

1. Open the updateGalleryRecord.php page and save it as deleteGalleryRecord.php

- In the Server Behavior panel click the – sign and remove the Update Record behavior.
- Select the Submit button and change the Value in the Property Inspector to Delete Record.
- With this button still selected, click the + in the Server behavior panel and select Delete Record. Fill out the form as per the image below.

- Save and upload the file.
- We now have to pass the URL parameter to this page so it knows what record we want to delete.
- Go to the gallery.php page and select the image icon in the table



- With the image selected go to the Server behavior panel click on the + sign and select Go to Detail Page. Fill out the form as per the image below.

![Screenshot of the 'Go To Detail Page' configuration dialog in a web development tool. The dialog has a blue header with the text 'Go To Detail Page'. Below the header, there are several input fields: 'Link:' with a dropdown set to 'Selection: ](<?php echo $r'; 'Detail page:' with a text box containing 'deleteGalleryRecord.php' and a 'Browse...' button; 'Pass URL parameter:' with a text box containing 'photoID' and the text 'Set to the Value Of'; 'Recordset:' with a dropdown set to 'rsImages'; 'Column:' with a dropdown set to 'photoID'; and 'Pass existing parameters:' with two checkboxes, 'URL parameters' and 'Form parameters', both of which are unchecked.)

9. Upload both the deleteGalleryRecord.php and the gallery.php pages. Your gallery page now serves three different functions. It allows you to see the details of each gallery item, update records and delete records.
10. All that is left to do is to make all of the correct links appear on your pages. For this you can use a server sided include for all links EXCEPT the Log Out link. This one must be done by hand because it is a php behavior.

**Congratulations. You have completed the final project!**

