

Introduction

Hello and Welcome to the ASP Special Online Module Format!

This module has been designed as an alternative delivery format of course material. It is hoped that you will find it to be just as valuable as our previous sessions have been. Before we begin, let me just take a moment to tell you how this module is designed. To begin, a summary or table of contents is provided so you can be aware of what each section contains. This is done to allow those who have already worked past some concepts to move more expediently to material that is new to them. Secondly, your expected assignments are explicitly listed in the beginning, during and at the end of this module. This was done to make it easier for you to not only know the expectations, but to think about the assignments before, during and at the end of the reading. Also, note that I have not enforced hard deadlines for any of the ASP assignments—this is done to ensure full comprehension of the concepts and to reduce stress and anxiety. Although it is recommended that you don't get too far behind. Lastly, this module contains a large amount of information and IT IS NOT my expectation that you have this entire module completed before our next meeting. Rather, I just wanted to ensure that you had enough information to continue towards successful progression of your ASP website. Again, I hope that you enjoy this experience and find it to be of great value. Let's begin with the module.

Table of Contents

1. Review of ASP and database (DB) interaction.
2. Review of ASP Query Page
3. Review of ASP Insert Page
4. Introduction of UPDATE Page.
 - 4.1 Page Two ASP UPDATE Code
 - 4.2 Page Three ASP UPDATE CODE
5. Introduction to DELETE Page
6. Introduction to the State Concept
7. Conclusion

Deliverables



Assignment 6

produce a successful asp query page

produce a successful page with includes

create an additional query page that only returns one or two field values

→ **Assignment Seven**

produce a successful asp insert page

→ **Assignment Eight**

produce a successful asp update

produce a successful asp delete

1. Review of ASP and DB Interaction

One great feature of Active Server Pages is its ability to interact with databases. However in order for this to occur we need to have six main items. These items are established in the beginning and after they are completed we almost never need to revisit them. Unfortunately sometimes problems arise and they usually are a result of something being wrong with these six items. These items are:

1. Need to have an IIS server.
2. Need to have Frontpage extensions installed on your web, which resides in the IIS Server.
3. Need to have a database created. Note, while we are using an Access database, the DB can be in another format like SQL or Oracle.
4. Need to have permissions explicitly declared for the Internet Guest Account (IUSR).
5. Need to create an ODBC Connection for the ASP pages to interact with the server.
6. Need to have a global.asa file in the root directory of your web.

You may also recall that I mentioned that you need to have an adovbs.inc file with a specific include call on your ASP pages. This is important for some ASP pages, but not for all pages—unfortunately if you fail to include this file some of your pages will not work—so it is best to have the file included.

2. Review of the ASP Query Page

In this example we reviewed the code that produced the result of displaying all the fields and records in a database. It was a simple client to server interaction, yet it holds importance as it provides the foundation for allowing very dynamic queries. For example, in the cool online music store we use queries to allow visitors to our website to see either the entire collection or just a small subset. Let's take a look at the code again.

```

<%@ LANGUAGE="VBSCRIPT" %> DECLARING THE LANGUAGE—although by default it is
VBSCRIPT
<!--#include file="adovbs.inc" --> --telling the server to find and read this
file before executing the rest of the page

<% --indicates the start of the script

    Set oConn=Server.CreateObject("ADODB.Connection")—creates a variable which
then creates an ASP object —specifically a database connection
    oConn.Open "carldb"—this variable object now references the ODBC connection
created and opens the Database associated with this system data source name
(DSN)
    strSQLStatement = "SELECT * FROM TEST " a string variable is created and
contains the SQL syntax to execute the query that shows all the fields and
records in a table called TEST from your database

Set oRs = oConn.Execute(strSQLStatement) a new variable is created that will
hold the result of the executed SQL syntax

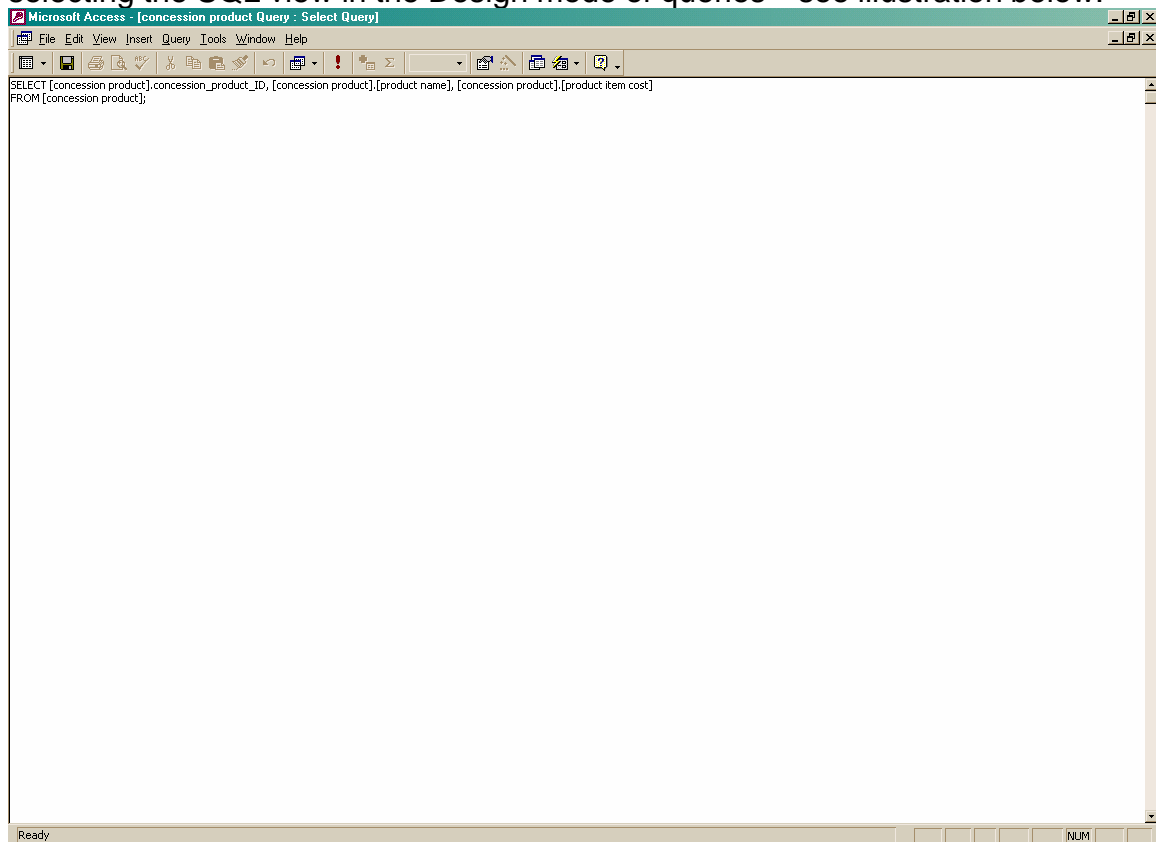
    oRs.MoveFirst %> this syntax tells the server to locate the very first
record that was retrieved

</p>
<table border="1" cols="<% = oRs.Fields.Count%>" width="100%"> this syntax
determines the number of fields and creates a table which number of columns
equals the number of fields
    <tr>
        <% For Each oField In oRs.Fields %> a simple for statement that tells
the server to continue until it runs out of fields—in the next line it tells
the server to print out every field name that it finds
        <th bgcolor="#000080"><font color="#FFFFFF"><% = oField.Name %>
        </font></th>
        <% Next %> once completed with fields, the server is instructed to
move to the records
    </tr>
    <% Do While Not oRs.EOF %> tells the server to continue until there are no
more records
    <tr>
        <% For Each oField In oRs.Fields %> as with the field columns these
lines tell the server to print the value of the field for the record and if
none exist print a space
        <td align="RIGHT"><% If IsNull(oField) Then
            Response.Write "&nbsp;"
        Else
            Response.Write oField.Value
        End If %> <br>
        </td>
        <% Next this instruction tells the server to continue and move to the
next field within the same record
            oRs.MoveNext %>
    </tr>
    <% Loop %> this loops tells the server to move to the next record
</table>
<% oRs.Close
    Set oRs = Nothing %> this command signifies the end of all the records found
and then deletes the information from memory for the server to use elsewhere

```

Note that at this point you were directed to replicate this code and successfully produce an ASP page that resulted in displaying the entire contents of a table in your database.

The last part of the query assignment asked you to experiment with the SQL syntax that queried the database. There were a couple of directions that one could have executed. One method was to remove the "*" and instead to have listed specific field names. The second was to be more specific by using a WHERE state to produce records that matched a specified criteria. While the second direction is a little more complex, you can find help within Access by selecting the SQL view in the Design mode of queries—see illustration below:



Deliverable required from review section:



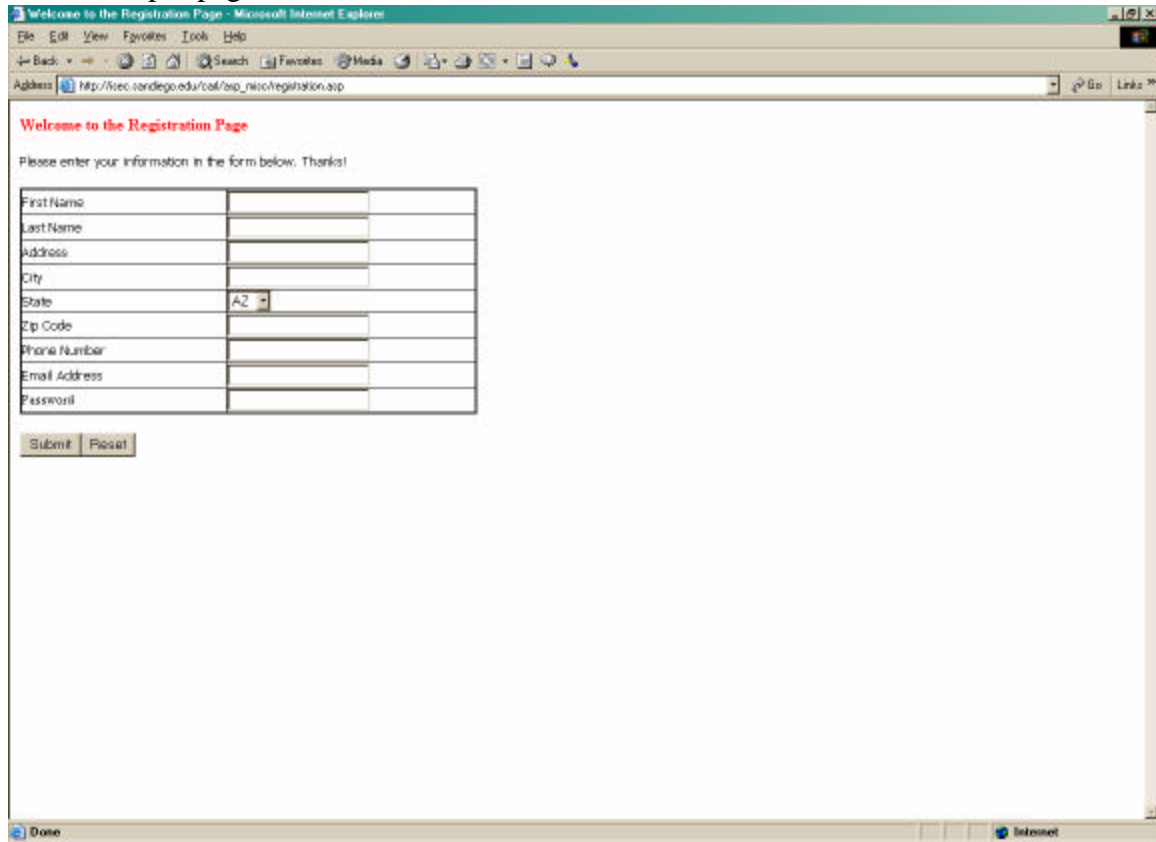
Assignment 6

- produce a successful asp query page*
- produce a successful page with includes*
- create an additional query page that only returns one or two field values*

3. Review of the ASP Insert Page

The specific example of this used in class was a simple online registration page. Overall, when we are utilizing an insert page—we first are creating a format for which a user can input data, and then we parse that data so it can be entered into a database. Just as in java—we cannot use the same variable name, so we have to create new variables that will take the same value as the original input page and then we can put them into a database. Let's take a look at this visually to see if we can get it to make more sense.

The first input page



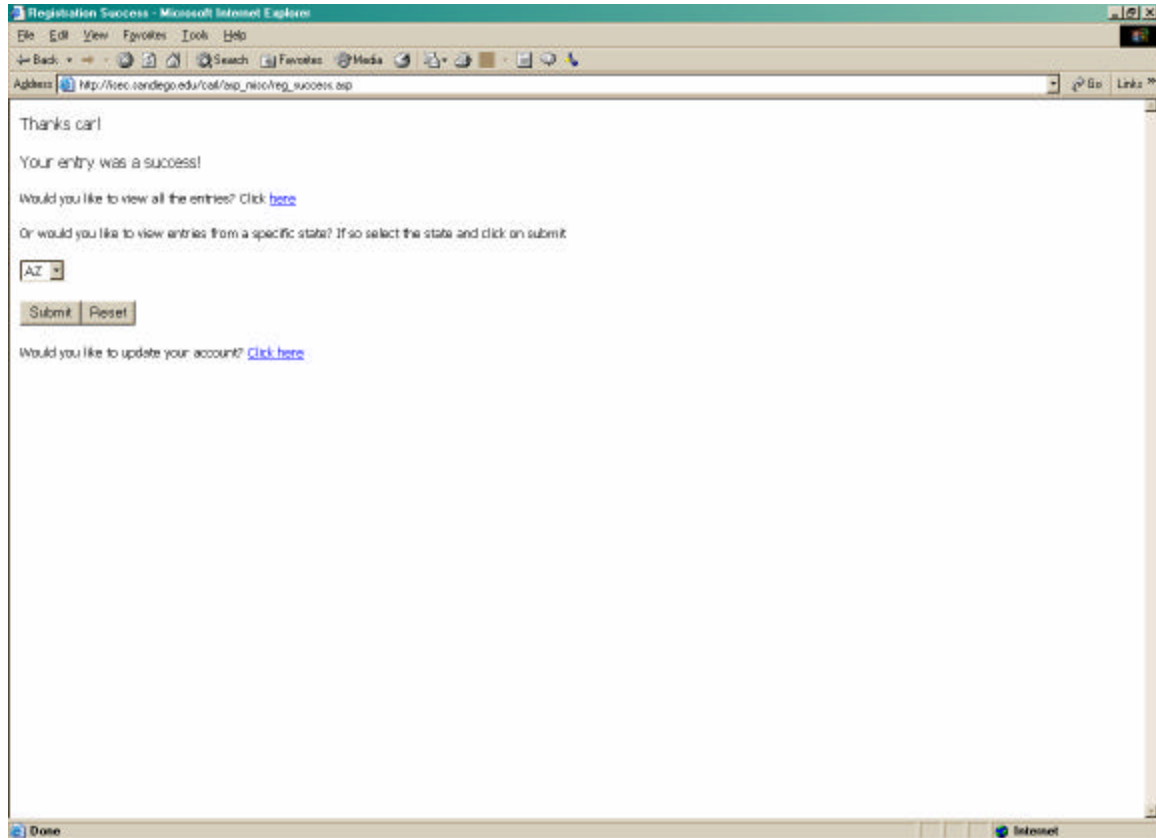
The screenshot shows a Microsoft Internet Explorer browser window displaying a registration page. The page title is "Welcome to the Registration Page" and the URL is "http://sec.sandiego.edu/cal/isp_netc/registration.asp". The page contains a form with the following fields:

First Name	<input type="text"/>
Last Name	<input type="text"/>
Address	<input type="text"/>
City	<input type="text"/>
State	<input type="text" value="AZ"/>
Zip Code	<input type="text"/>
Phone Number	<input type="text"/>
Email Address	<input type="text"/>
Password	<input type="text"/>

Below the form are two buttons: "Submit" and "Reset".

In this page a quick little table is created that contains row titles next to text boxes. Each text box is given a name—for example First Name is listed left of a text box that is titled “txtfname”, Last Name is listed left of a text box that is titled “txtlname”, and so forth. The submit button of this page sends the values entered by the user to a page called `reg_success.asp`. This redirection is caused by right-clicking within the form, selecting form properties, selecting the option button, and finally by typing the name of the page into the box.

The redirect or second page called `reg_success.asp` produces the following result:



This page provides a confirmation message in addition to allowing the user to select more options. At this point we are not as concerned with the future options as we are with the successful insertion of the user's information. Let's take a look at the code that made this happen:

```
<%@ LANGUAGE="VBSCRIPT" %>
<!--#include file="adovbs.inc" -->

<html>

<%

    strfname = cStr(Request.Form("txtfname"))
    strlname = cStr(Request.Form("txtlname"))
    stradd = cStr(Request.Form("txtadd"))
    strcity = cStr(Request.Form("txtcity"))
    strstate = cStr(Request.Form("txtstate"))
    strzip = cStr(Request.Form("txtzip"))
    strphone = cStr(Request.Form("txtphone"))
    stremail = cStr(Request.Form("txtemail"))
    strpword = cStr(Request.Form("txtpword"))

Set oConn=Server.CreateObject("ADODB.Connection")
oConn.Open "carldb"
strSQLStatement = "INSERT INTO REGISTRATION" _
    & "( FNAME, LNAME, ADDRESS, CITY, STATE, ZIP, PHONE, EMAIL, PASSWORD" _
) " _
    & "SELECT '" & strfname & "' AS FNAME, '"_
```

```
& strlname & "' AS LNAME, '"_
& stradd & "' AS ADDRESS, '"_
& strcity & "' AS CITY, '"_
& strstate & "' AS STATE, '"_
& strzip & "' AS ZIP, '"_
& strphone & "' AS PHONE, '"_
& stremail & "' AS EMAIL, '"_
& strpword & "' AS PASSWORD ;"
```

```
oConn.Execute(strSQLStatement)
```

```
%>
```

Notice that much of the code for the insert page looks very similar to that of the query page. We are still declaring the language and including the `adovbs.inc` file. Once this is completed you'll notice something new—that is we are starting off by declaring variables to capture the information contained in the previous page. This method is one that refers to 'state', which is a topic that will be further discussed later. For now, think of it like this, in ASP information is passed from one page to another—BUT if you want to do something else with the data-like DB insertion or passing it to another page you have to "revive its state." Perhaps another way of thinking about it is like this—your friend tells you a story and you understand it perfectly—but if you try to tell it to a third person it might take a little more explanation. The same holds true for ASP. This is why we create new variables to capture the data from the registration table page.

To continue, once we have created new variables to capture all of the textbox data, we recreate our DB connection. After we recreate our connection we need to rewrite the syntax to allow for DB insertion. The SQL syntax is changed to inform the DB as to which table we wish to insert a new record, followed by which fields, and concluded by which values to enter for each field in the record. Once this is completed we then executed the SQL request. The last item that we utilize in this page is the `Response.Write` object in ASP. This `Response.Write` command tells the server to print out the value of the variable in the parenthesis. Again, this is one example of manipulating variable state for our purposes. However, this method only works in a A-B page communication process. Most common errors in attempting to execute this process reside in failure to match up correct new variable names with the textbox variable name with the fields. I know that sounds a little confusing, but remember you are creating an intermediary that is basically "the person in the middle". For example, you know that your registration page contains a text box called `txtfname` and your database has a field name call `FNAME`. [Remember that I use all CAPS to designate my field names and lowercase for variables]—now all you have to keep track is that you have created a temporary variable called `strfname` that is able to capture the 'baton or value' from the registration textbox field and transfer it to the DB field name.



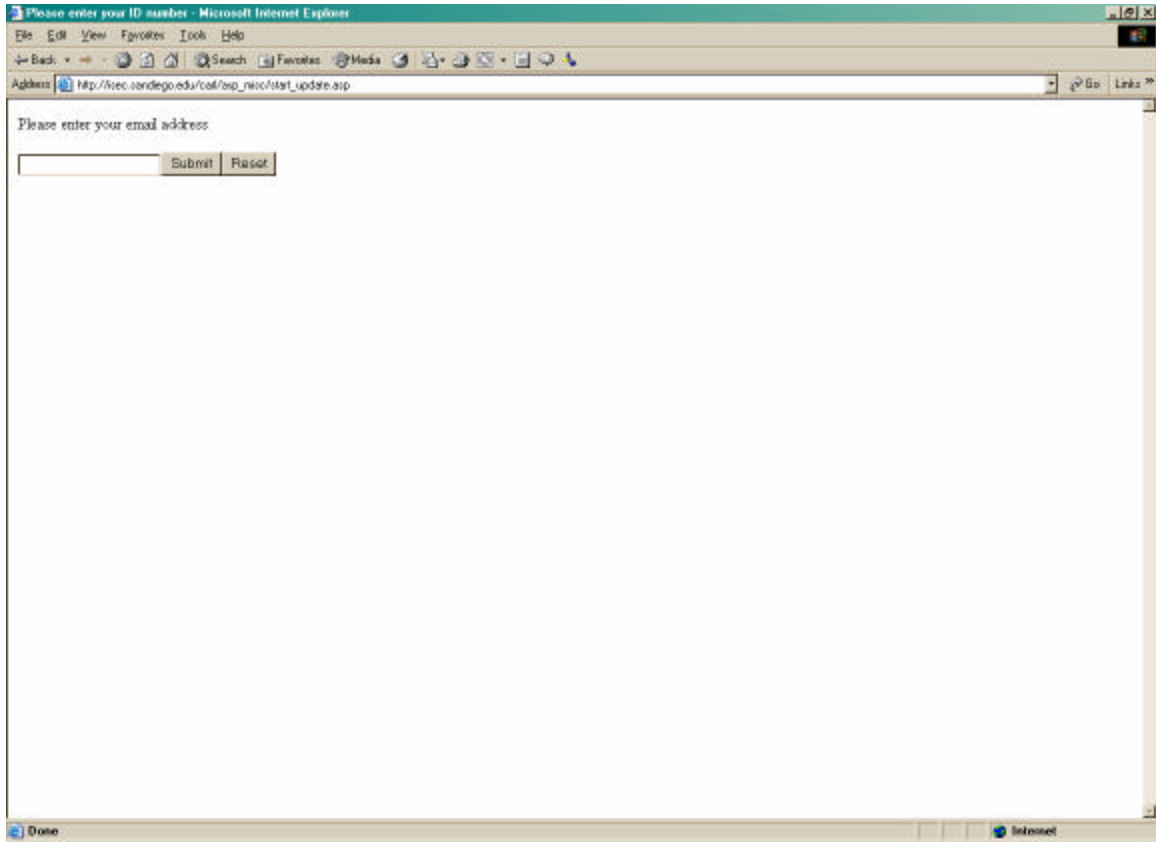
Assignment Seven

produce a successful asp insert page

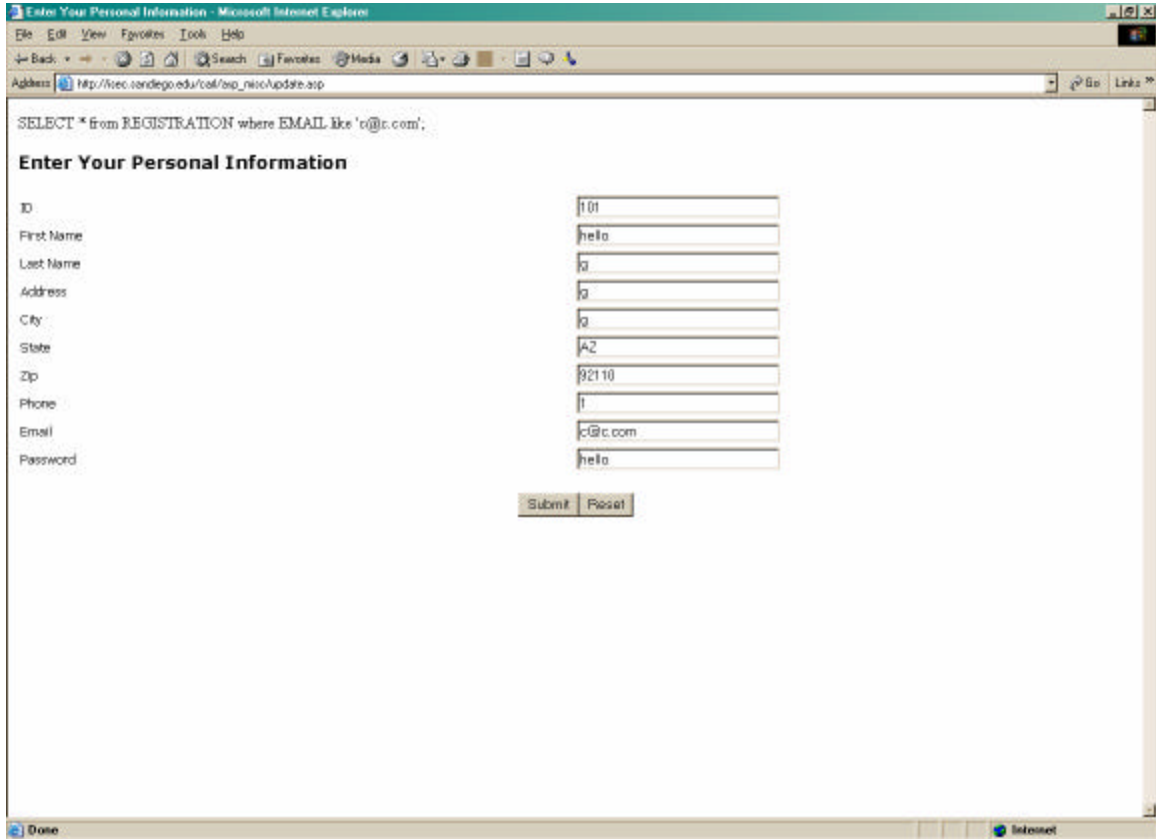
4. Introduction to the UPDATE Page

This is the point where we left off in our last class session. The UPDATE concept is actually not much more than the combination of the query and insert pages—except with a little twist. That is, in our first step we need to query our database to find a specific record [NOTE: It is true that we could use our UPDATE code to modify multiple records—but for purposes of our examples we will deal with only one specific record to keep things simple]. Once we have located this record via our initial page request through our ASP DB Connection, we need to display the results—AND also in a form that we can make modifications. The final third page is where we make the changes to the database. Let's take a look at how this works visually:

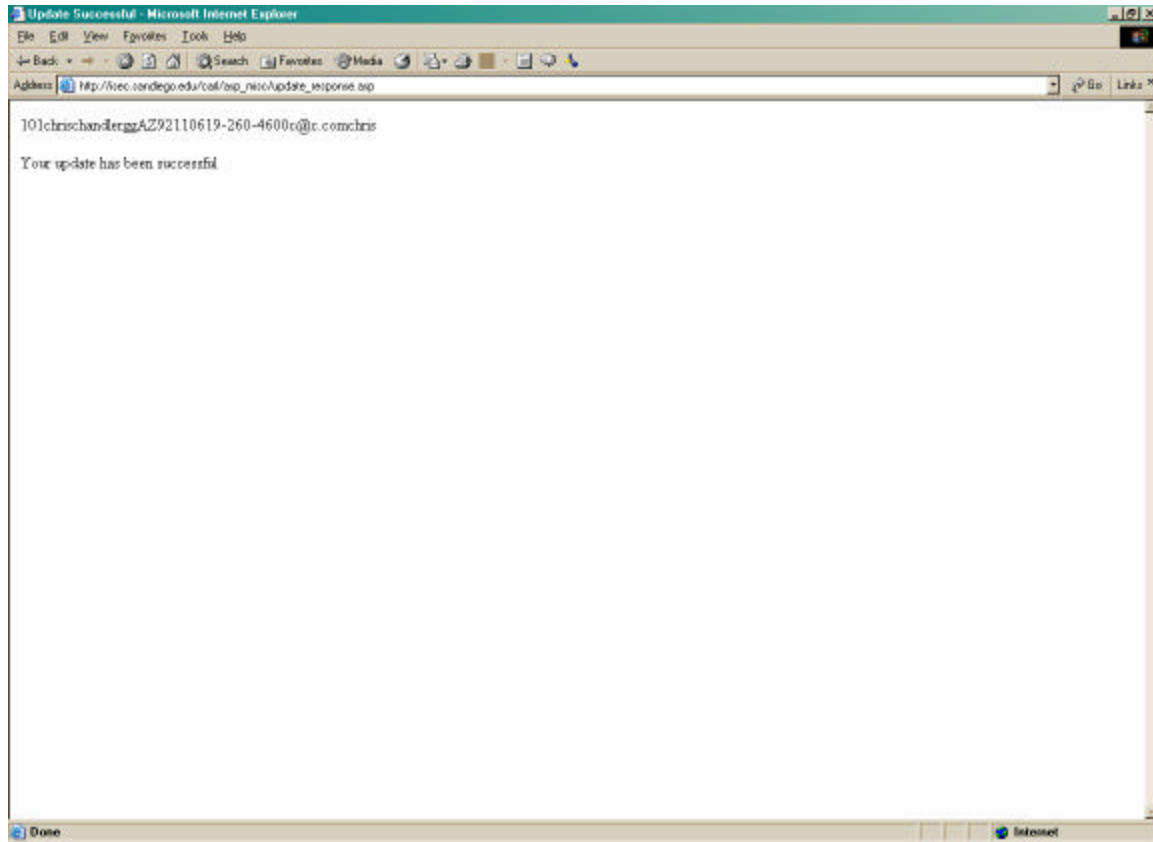
Page One—the start of the query page—which begins by our request of SOME UNIQUE FIELD IDENTIFIER—in this case an email address of which there, can only be one single entry. Note that is page is a simple form with one textbox. The textbox gets assigned a unique name, which gets passed to Page Two.



Page Two- The successful execution of our query and reproduction of all the fields associated with the record. At this point we are allowed the capability to modify the fields as indicated by the textboxes in the page result provided.



Page Three—Once we made modifications we are provided with a confirmation page as shown below. Note in this page illustration we have utilized the *Response.Write* command to show on the screen the results of the changed data. (It might be a little hard to notice but in this case the first name was changed from *hello* to *chris*, and the last name was changed from *g* to *chandler*)



Hopefully, this illustration provides a clear picture of the UPDATE page process. Let's take a look at the code for this process—for which we will only discuss pages two and three.

4.1 Page Two ASP UPDATE CODE

```
%@ LANGUAGE="VBSCRIPT" %>--again we declare the code and include the adovbs.inc file
```

```
<!--#include file="adovbs.inc" -->
<%
```

```
stremail = cStr(Request.Form("txtemail"))here we create a new variable that captures the value from the textbox from the previous page
```

```
%>
<html>
```

```
<% Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open "carldb"
strsql = "SELECT * from REGISTRATION where EMAIL like '&stremail&'";

response.write strsql<br>"
set oRs = Conn.Execute(strsql)
oRs.MoveFirst %> --Notice this is the same ASP code from the query page, where we create a DB connection, open it, and then use SQL syntax to find a match to the input from the textbox from page one
```

```
<head>
<meta http-equiv="Content-Language" content="en-us">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<meta name="GENERATOR" content="Microsoft FrontPage 4.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<title>Enter Your Personal Information</title>
</head>
```

<body> the following code creates a table within a form to produce the results of the query in the form of a table with textboxes. The only difference is that we are not executing a FOR LOOP through the records, we are explicitly indicating which Field and values to display to the user screen. This is done by the <%=oRs("FIELD NAME" %>. -There is a very easy way to create this as opposed to hard coding the page—simply create a table and insert blank textboxes. Once the textboxes are created, double click and type the preceding code (<%=oRs("FIELD NAME" %>.) under the space title INITIAL VALUE—also FIELD NAME will vary according to the fields in your DB.

```
<p><font size="4" face="Verdana"><b>Enter Your Personal Information</b></font></p>
<form method="POST" action="update_response.asp">
  <table border="0" width="100%">
    <tr>
      <td width="50%"><font size="2" face="Tahoma">ID</font></td>
      <td width="50%"><input type="text" name="txtID" size="30" value="<%=oRs("ID")%>"></td>
    </tr>
    <tr>
      <td width="50%"><font size="2" face="Tahoma">First Name</font></td>
      <td width="50%"><input type="text" name="txtfname" size="30" value="<%=oRs("FNAME")%>"></td>
    </tr>
    <tr>
      <td width="50%"><font size="2" face="Tahoma">Last Name</font></td>
      <td width="50%"><input type="text" name="txtlname" size="30" value="<%=oRs("LNAME")%>"></td>
```

```

        </tr>
        <tr>
            <td width="50%"><font size="2" face="Tahoma">Address</font></td>
            <td width="50%"><input type="text" name="txtadd" size="30"
value=" <%=oRs( "ADDRESS" )%> "></td>
        </tr>
        <tr>
            <td width="50%"><font size="2" face="Tahoma">City</font></td>
            <td width="50%"><input type="text" name="txtcity" size="30"
value=" <%=oRs( "CITY" )%> "></td>
        </tr>
        <tr>
            <td width="50%"><font size="2" face="Tahoma">State</font></td>
            <td width="50%"><input type="text" name="txtstate" size="30"
value=" <%=oRs( "STATE" )%> "></td>
        </tr>
        <tr>
            <td width="50%"><font size="2" face="Tahoma">Zip</font></td>
            <td width="50%"><input type="text" name="txtzip" size="30"
value=" <%=oRs( "ZIP" )%> "></td>
        </tr>
        <tr>
            <td width="50%"><font size="2" face="Tahoma">Phone</font></td>
            <td width="50%"><input type="text" name="txtphone" size="30"
value=" <%=oRs( "PHONE" )%> "></td>
        </tr>
        <tr>
            <td width="50%"><font size="2" face="Tahoma">Email</font></td>
            <td width="50%"><input type="text" name="txtemail" size="30"
value=" <%=oRs( "EMAIL" )%> "></td>
        </tr>
        <tr>
            <td width="50%"><font size="2"
face="Tahoma">Password</font></td>
            <td width="50%"><input type="text" name="txtpwd" size="30"
value=" <%=oRs( "PASSWORD" )%> "></td>
        </tr>
    </table>
    <p align="center"><input type="submit" value="Submit" name="B1"><input
type="reset" value="Reset" name="B2"></p>

</form>

</body>
<% oRs.Close
    Set oRs = Nothing %>

</html>

```

4. 2 PAGE THREE ASP UPDATE CODE

As you can tell, the ending UPDATE code looks very similar to the INSERT code discussed earlier. We are still creating new variables and assigning them the values from the textbox from the preceding page. The only real change is the SQL syntax—which says UPDATE as opposed to INSERT, but also includes the WHERE statement. This works exactly like it sounds. You are telling the DB to change all of the record fields for one specific entry that matches your UNIQUE identifier—EMAIL. So be careful you don't allow the user to make changes to that field—otherwise your page will not be successful.

```
<%@ LANGUAGE="VBSCRIPT" %>
<!-- #include file="adovbs.inc" -->
<html>

<head>
<title>Update Successful</title>
<meta name="GENERATOR" content="Microsoft FrontPage 4.0" >
<meta name="ProgId" content="FrontPage.Editor.Document" >
</head>

<body>

<%

    strfname = cStr(Request.Form("txtfname"))
    strlname = cStr(Request.Form("txtlname"))
    straddress = cStr(Request.Form("txtadd"))
    strcity = cStr(Request.Form("txtcity"))
    strstate = cStr(Request.Form("txtstate"))
    strzip = cStr(Request.Form("txtzip"))
    strphone = cStr(Request.Form("txtphone"))
    stremail = cStr(Request.Form("txtemail"))
    strpword = cStr(Request.Form("txtpword"))

Response.Write(strid)
Response.Write(strfname)
Response.Write(strlname)
Response.Write(straddress)
Response.Write(strcity)
Response.Write(strstate)
Response.Write(strzip)
Response.Write(strphone)
Response.Write(stremail)
Response.Write(strpword)

Set oConn=Server.CreateObject("ADODB.Connection")
oConn.Open "carldb"
strSQLStatement = "UPDATE REGISTRATION " _
& "SET FNAME = '"&strfname&"'," _
    & "LNAME = '"&strlname&"'," _
    & "ADDRESS = '"&straddress&"'," _
    & "CITY = '"&strcity&"'," _
    & "STATE = '"&strstate&"'," _
    & "ZIP = '"&strzip&"'," _
    & "PHONE = '"&strphone&"'," _
    & "EMAIL = '"&stremail&"'," _
```

```
        & "PASSWORD = '"&strpword&"'_  
        & "WHERE EMAIL = '"&stremail&"';"  
oConn.Execute(strSQLStatement)  
%>
```

5. Introduction to the DELETE Page

If you have made it successful to this part, then you will enjoy how easy the DELETE feature is—as it follows the same process as the UPDATE page—the only difference is simply just a little bit of code—mostly in the form of a word DELETE.

You should still create an initial page that prompts the user for a specific record to delete. This can be done by using the same UNIQUE IDENTIFIER and then displaying the results of that record (same as page two in UPDATE). Now on Page Three you will want to change your ASP DB code to read as such:

```
<%  
stremail = cStr(Request.Form("txtemail"))  
  
Set oConn=Server.CreateObject("ADODB.Connection")  
oConn.Open "carldb"  
strSQLStatement = "DELETE from REGISTRATION where EMAIL = '"&stremail&"';"  
oConn.Execute(strSQLStatement)  
%>
```

See its that simple! The good news is that many of the other items we will investigate will turn out to be almost just as easy—as from here on out we will continue to return to the main ASP functions of retrieving information from a database and then manipulating the values for our unique specific purposes.

Assignment Eight

produce a successful asp update
produce a successful asp delete

6. Introduction to the State Concept

Now that you have down the main ASP DB interaction tools its time to understand how we can utilize information drawn from one page throughout the individual's experience in all of our website pages. This concept is referred to as 'state' and was mentioned earlier. Let's delve down into a deeper discussion.

It's a Fact: The Web Is Stateless

Have you ever wondered what, exactly, happens when you type in a URL into your browser's Address bar? The Internet is based on a client-server model, where your Web browser is the client, and the Web server is the server. The

figure on the left provides a graphical representation of the client-server model.



In the client-server model, the client opens up a channel of communication with the server and requests a resource. The server receives the request, locates the resource being requested, and sends it to the client, closing the channel of communication.

This is an impersonal transaction

between the client and server. The server does not keep open its channel of communication with the client. The server is not concerned with whom it has talked to recently, what it has sent recently, or what it thinks a particular client will request next. The server does one job, and it does it well: wait for a request from the client and then process that request.

Due to this impersonal communication between a Web browser and a Web server, user information is not persisted from one request to another. Imagine that you wanted to create a form that would query the user for his or her name. After the user entered his or her name, it would be nice to display a personal greeting on each Web page on your Web site. To display a personalized welcome message, you would need some way of remembering each user's name. Saving such information is referred to as maintaining state, or persisting state.

Ways to Maintain State

Because the client-server model does not make maintaining state inherently easy, you must examine some advanced techniques to maintain state. No doubt you'll find that some of the methods that can be used to persist state seem rather obtuse.

Later we'll look at how to maintain state by sending state information through the querystring. Using this approach can lead to a syntactical headache, and you may find yourself wondering why maintaining state through the querystring appears to be so confusing. Keep in mind that the client-server model does not lend itself to state persistence; therefore, some methods of maintaining state can be unwieldy. Thankfully, ASP provides some built-in objects and collections to help maintain state such as cookies and sessions. We will discuss these in our next meeting.

7. Conclusion

By now you should have created a successful ASP Query, INSERT, UPDATE, and DELETE page. You should also have an understanding of the concept of state so that when we meet again you'll be ready to hit the ground running to tackle the concepts of querystrings, cookies, and sessions—the basic underpinnings of the 'shopping cart' which fuels the electronic commerce world. At this point let's address the deliverables. The original intention was that each person would hard code the first four tools, QUERY, INSERT, UPDATE, and DELETE. After successful completion of these tasks, each person would then replicate these pages towards their final project. In terms of assignment submission, you can select to either just replicate the pages provided or manipulate them towards your final project. It is your decision, however, you are better served by selecting the latter. Either way please email me the exact URL to check your assignment submissions.

I would also like to remind you to take advantage of the files that I have given to you—as you can gain a great deal by unzipping the files and placing them into your web. The only changes you will have to make is to copy the DB files from my database into yours, and then of course changing the DSN call. This way you will have a full working ASP site in which to emulate your final project from. I understand that I presented you with a great deal of material but ideally when we meet again you'll have all the assignments completed. If not, no worries, we can work from there, but hopefully this module has been useful to you.

Best,

Carl

Deliverables

- **Assignment 6**
 - produce a successful asp query page*
 - produce a successful page with includes*
 - create an additional query page that only returns one or two field values*

- **Assignment Seven**
 - produce a successful asp insert page*

- **Assignment Eight**
 - produce a successful asp update page*
 - produce a successful asp delete page*