Chapter 11
The Web meets your Database
The Server-side of the Web

Chapter Outline

Chapter 11
The Web meets your Database
The Server-side of the Web

Plans for our online dating service are moving along nicely. We have a great web page designed to attract clients – and "*The Meet Market*" is a catchy name, given the youngish audience we expect to attract. In order to make any money, however, we need to move past the static pages we designed in the HTML chapter. The web form we created in chapter 10 was a step in the right direction. It will help collect data from new clients. Now we need a way of getting their data passed along to the server housing our database. Once we have customers in the database, we can build a web form that lets people search for that special someone.

To pull this system together, we'll need to use a programming language called PHP. This is the crucial glue that connects our web pages to the database. Each line of PHP code we write will be executed on the web server. There will be commands to open connections to the database, commands to execute SQL statements that insert data into the database and commands to run SQL statements that search the database. There will also be lines of HTML code that help us organize any response we get from the database – trying to make the results look good before we send search results back to the user's web browser. For our dating service, those results might be a list of dating possibilities that we find in the database!

11.1 The Big Picture – overview of the online Dating Service

The dating system described in this chapter is purposefully simple – we want you to follow the basic approach to building a web site that can interact with a database. Once you see how the process works you can apply the techniques to other systems. Chapter 12 (Making it Real) builds a more complete version of the dating system, complete with images of the customers, checkboxes and comment areas where the new dating clients can describe themselves. Chapter 12 also describes the modifications necessary for the "Meet Market" to establish itself on the actual internet.

Let's start with an overview of what's happening in the dating system. It all begins with a static web page that describes the service we are providing. This initial page has two

links on it – one links to a page that collects data about new customers and the second links to a page that helps people search for a date.

The page that collects customer data includes a web FORM that customers will use to supply us with their name, gender and age. The web FORM is part of the activity taking place on the client-side of our dating system. Keep in mind that while a web FORM is a technique for collecting data – we haven't moved that data into the database yet. Moving data to and from the database is the responsibility of the server-side of this dating system. There will need to be a page sitting on the server that can accept the data being handed it by the web FORM located back at the client's browser. The page located on the server will handle all interaction with the database. In the case of a new customer, we will need the page on the server to execute a SQL statement to insert customer data into the database.

When dating customers click the link to our dating *search* page, they will find themselves on another web FORM. This FORM will ask for the age and gender of the people we will search for. Once the customer clicks "Submit", the browser will hand the search criteria to the server. These criteria might ask us to find females between 25 and 30 years old. In this case, we expect the page on the server to execute a SQL SELECT statement to find the appropriate records in the database. Once the records are retrieved from the database, the page on the server will need to format the answers and send the response back to the client's browser. How can the page at the server talk to the database? Well, that's all handled by PHP.

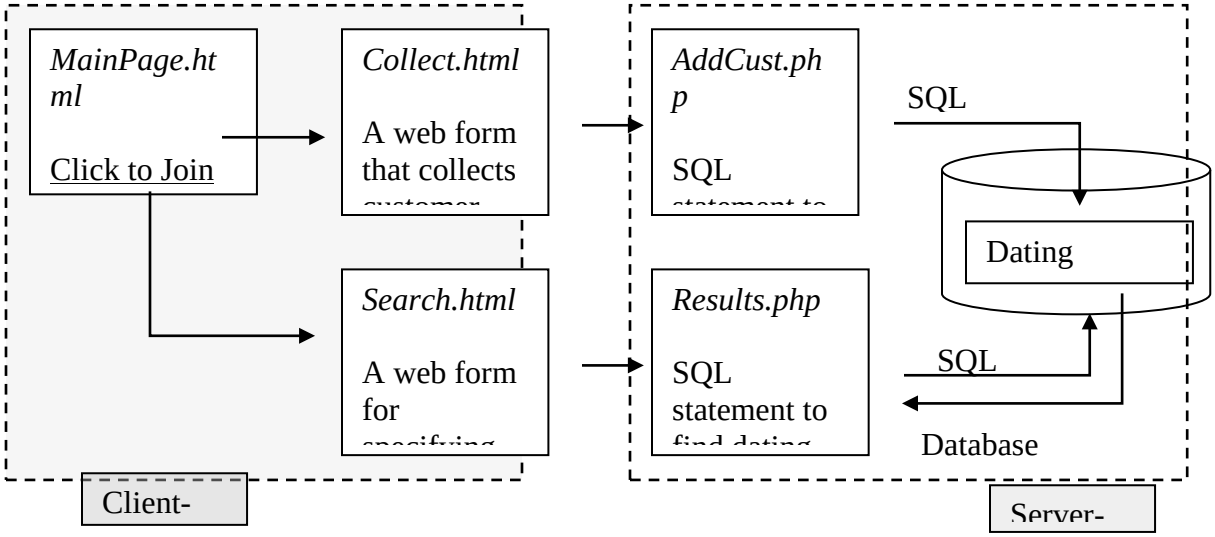


Figure 11-1. Web/database systems involve a division of labor between a client

Notice that each page that interacts with the database is a PHP file. The PHP pages can look like normal html files – with activity related to the database being handled by PHP. Those PHP sections are located within delimiters…simply codes that mark where the

PHP begins and ends. Our commands to connect to the database and execute SQL statements will be within those ASP delimiters. In PHP the delimiters are **<?php** and **?>**

11.2 Getting data from a web FORM to a PHP page

Let's begin by describing how data gets from a Web FORM to your ASP page and ultimately into your database. The story focuses on the pages we've called Collect.html and AddCust.php. Collect.html is a pretty extraordinary web page. What makes it special is the <FORM> tag. You know from Chapter 10 that the FORM defines the collection of inputs we are offering to the dating customer. In the case of Collect.html, the FORM has inputs for name, age and gender.

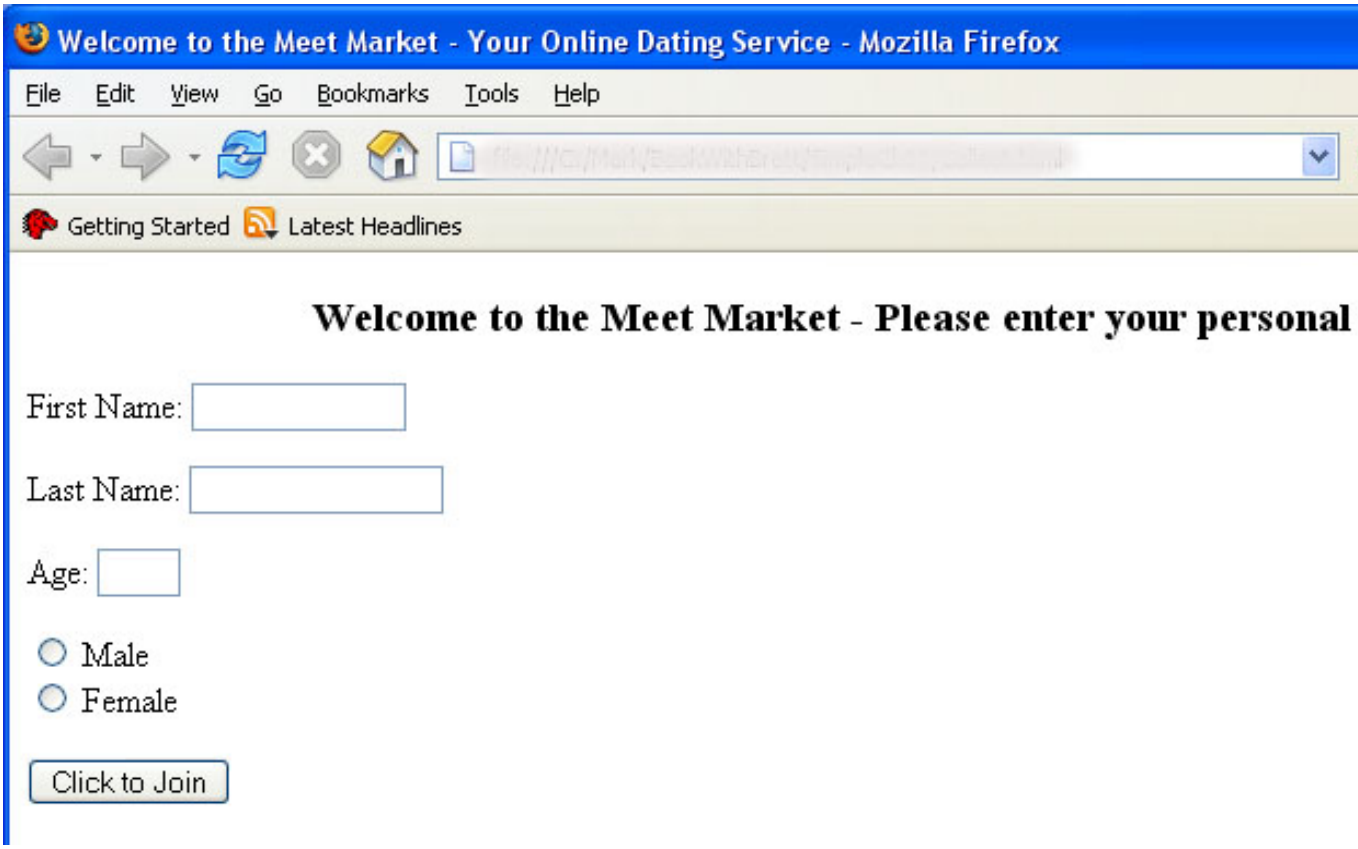


Figure 11-2. The Collect.html page is a "Web FORM"

Collect.html also has a Submit button. I'm sure the layman assumes the Submit somehow gets the data into the database. As you might suspect, the real story is a bit more involved.

The web FORM interacts with the web server through two critical entries that determine HOW the data will be sent and TO WHERE it will be sent. The details of this data transfer are described by two entries in the FORM tag; ***METHOD*** and ***ACTION***.

Here's the html code (Figure 11-3) *that generated the Web Form in Figure 11-2.* Pay close attention to the FORM tag. That tag describes HOW the data will be sent to the web server and where it will land once it gets there. In this case, the data will be POSTed to a PHP page named AddCust.php, where all the **action** will take place.

```
<html>

<head>
    <title>Welcome to the Meet Market - Your Online Dating Service
</title>
</head>

<body>

<center> <h3> Welcome to the Meet Market - Please enter your personal
data </h3> </center>

<p>
<FORM method = "POST"  action = "AddCust.php" >

First Name: <input type="text" size="12" name="frmFname"> <p>

Last Name:  <input type="text" size="15" name="frmLname"> <p>

Age:       <input type="number" size="2" name="frmAge"> <p>

<input type="radio" name="frmGender" value="Male">  Male  <br>
<input type="radio" name="frmGender" value="Female">  Female <p>

<input type="submit" name="submit"  value="Click to Join">
```

Figure 11-3. HTML code for the web FORM in

POSTing data from a Web FORM to the ASP page happens when someone clicks the

The recipient of the data is always the page mentioned in the "ACTION" entry of the FORM tag. Notice (Figure 11-4) that the POSTed data is available to the *AddCust.php* page – but that *no data will be written to the database until the PHP page establishes a connection to the database and executes a SQL INSERT statement*.

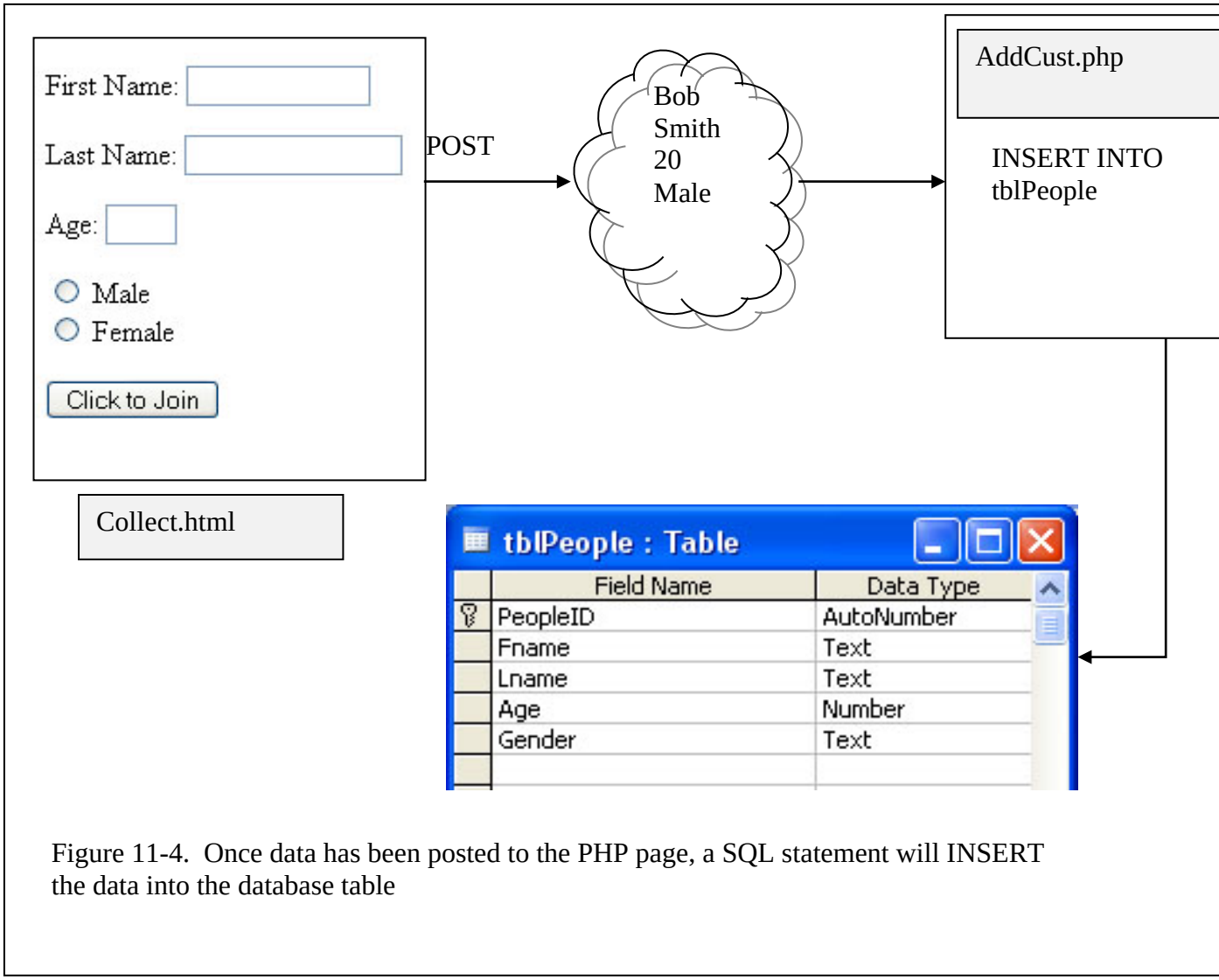


Figure 11-4. Once data has been posted to the PHP page, a SQL statement will INSERT the data into the database table

Every example in this book will use POST as our Method. You should be aware that another option exists. You've seen evidence of this second approach whenever you use a search engine on the web – this second option is called GET.

GET packages the data as part of a URL. You've seen evidence of this many times. The browser uses a question mark to separate the data from the name of the page that is being sent the data. For our dating system, it might look like:

http://www.MeetMarket/AddCust.php ? frmFname = "Sally" &

GET is often restricted to querying a database – and is also limited in the amount of data it can handle. You'll often see it used in web search engines. GET also requires slightly different server code. To keep things simple (we want to avoid two different systems if

we can get away with it) we'll stick to POST. Just be aware that many times you'll see evidence that someone is using GET – it's kind of cool to be doing a web search in ***Google*** and to see the URL of the search with a question mark followed by your search criteria…and to realize what is going on!

POST collects the data into a bundle and moves it as part of the HTTP header. Here is the POST from our Collect.html web FORM. Notice how the browser has listed each field and the value being sent from the web FORM. Each entry on the web FORM has been concatenated together to form one long string of data that will be passed to the AddCust.php page sitting on the web server.

- Page:
  POST 25 bytes to /MeetMarket/AddCust.php

- POST Data:
  frmFname=Mark&frmLname=Brickley&frmAge=55&frmGender=

Would you like to check out the POST from your web forms? Sometimes it can really help debug a problem with your system if you can see what data is being sent to your PHP page. In chapter 10 we viewed the data being sent by the form by submitting the data to an email – in this chapter I'm forcing the browser to disclose the POST by deliberately making an error on the AddCust.php page. The browser revealed the POST when it found that the data couldn't be handed to the AddCust page. I'll show you how to force this error once we start building the AddCust page.

11.3 Steps involved in developing an PHP page:

Now that you have some sense of how the data will get from the *web form* (processed by the browser) to the *PHP page* (located on the web server), it's time to prepare the database for a connection and get started on our AddCust.php page.

The steps involved in creating our first PHP page may seem somewhat complex at first, but there is a pattern you can follow. As an overview,

- you will need to create a database, a "user" and give that user the clearance to access the database. Most of the steps are simple.

- you need to write a few lines of PHP to connect it to your database

- finally, you can retrieve the data from the POST and execute a SQL statement. With SQL you can insert data into the database, search for data and update or delete data. These functions are critical to populating our dating database,

searching it for potential dates and updating the database as people decide to leave the system or change their personal data.

11.3.1 Creating the database – we will use a new database system – not Access.

We spent a month learning the basics of database earlier this semester. The database we used was Microsoft Access – primarily a desktop database. PHP often works with a database named MySQL; a popular database used for online businesses. We will learn the basics of mysql…creating a database and a simple table.

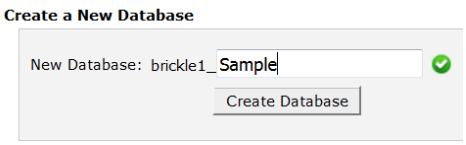
Follow the steps carefully:

FIRST…you DO need a web form. Each field needs a name – that means you need names for the data represented by your text inputs, radio buttons, dropdowns and textareas. MAKE A LIST OF THE NAMES YOU GAVE THOSE FIELDS. My recommendation is to name every field starting with "frm"; so the names might be frmFname, frmLname etc.

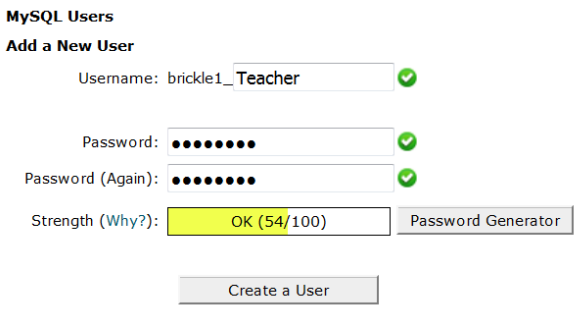
SECOND…you need a database to eventually HOLD the values posted by the web page. You will create the database using TWO tools on the cPanel at CuttingEdgeSpace. The first tool is **MySQL Databases**. It will create the database 'shell', establish a user and their password and finally – connect that user with the database.

From the cPanel click on MySQL Databases

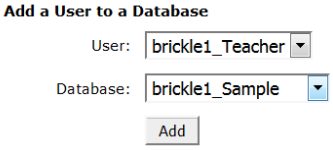
Create a database by providing a name. Be sure to write down the entire name including the 'prefix' that is already filled in for you. In my case the database might be named brickle1_Sample



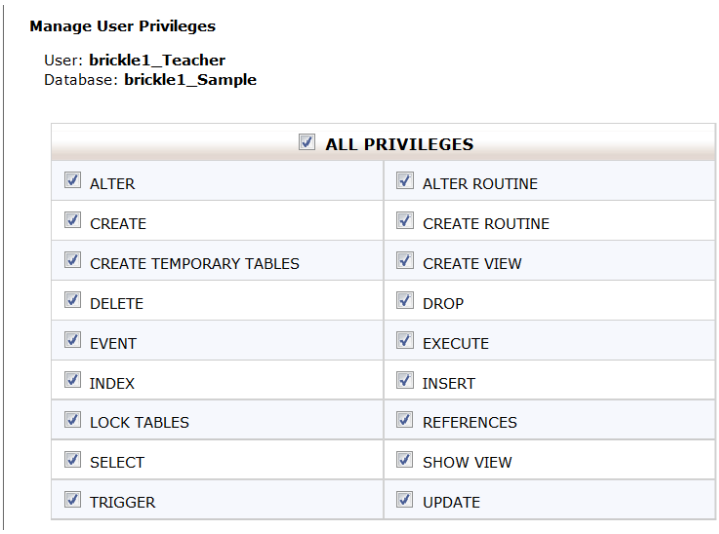
Create a User; again, be sure to write down the ENTIRE name of that user. Write down the password since you will need it when you write your PHP.



Assign that new user to the database you just created.



Give that user a full set of permissions to use your database. This includes the ability to execute all the various SQL statements. At the very bottom of this screenshot, I missed grabbing the button that says “Make Changes”. Be sure you click “All Priviliges”, then hit that “Make Changes” button

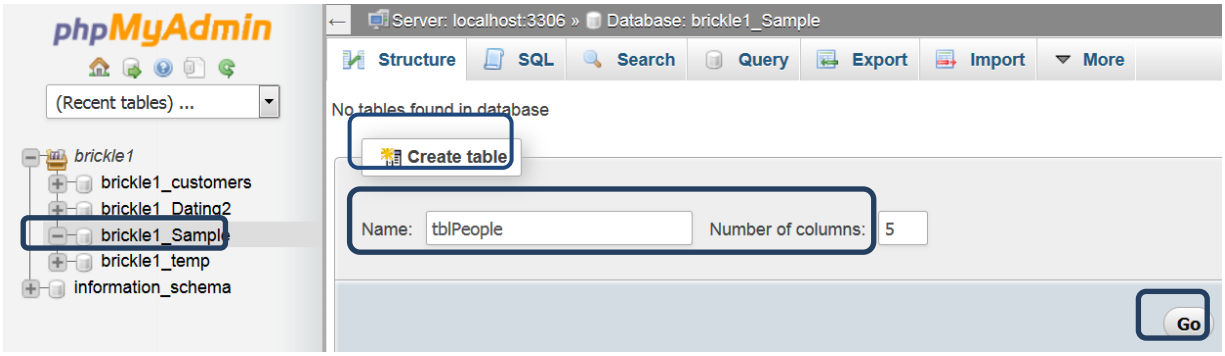


In the upper left corner, click "Home" to get back into your control panel.

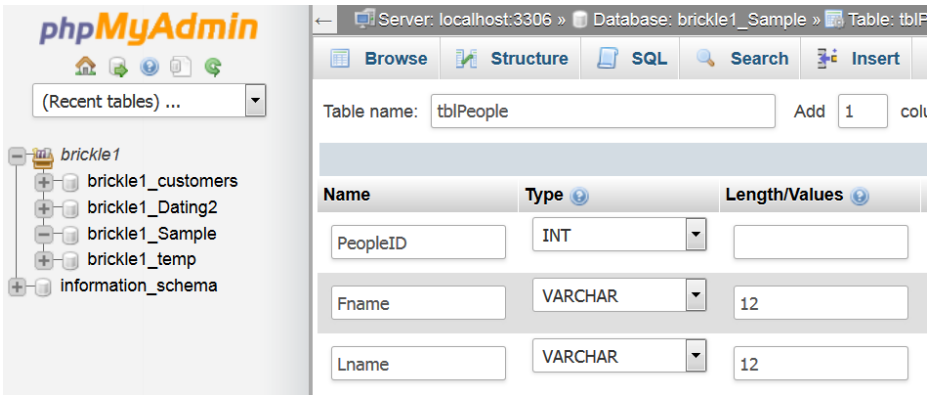
THIRD…use **PHPmyAdmin** to build your database table. This isn't that different from the way you created tables in Access. You make a list of the fields you need, choose a data type for each field and a length for the text fields. You will define one of those fields as your Primary Key and set it to be an Autonumber.

Once in PHPmyAdmin, click on your database, then click the CREATE TABLE button. I recommend you naming the table with "tbl", so it might be tblPeople, tblCustomers, tblStudents or something along those lines.

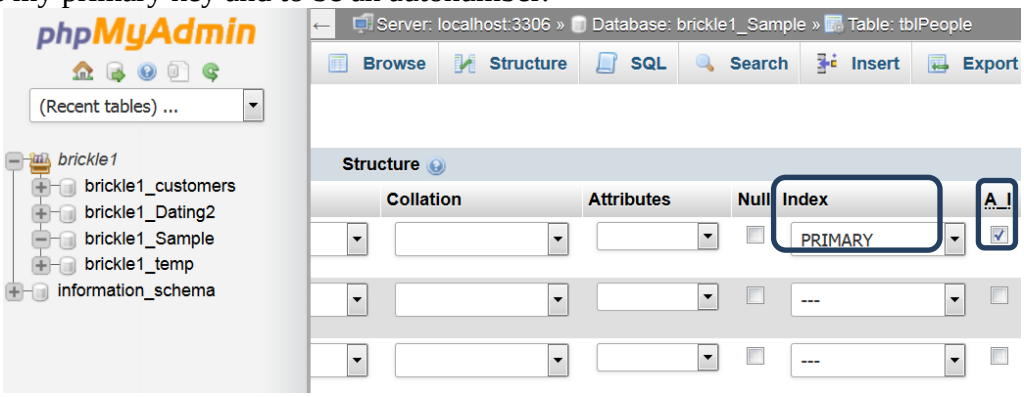
Enter the number of fields / columns you need



List the fields you need, select VARCHAR as the data type for your text fields and assign a length. If you want a primary key to use an AutoNumber, select INT as the data type, set the INDEX to be PRIMARY and check off the AI (AutoIndex) checkbox.



Notice that I set my primary key to have a data type of INT (integer) and did not assign a length to it. Here's a shot of that screen just to the right of what you are currently looking at. Notice I have set the Index to "Primary" and checked off "AI" to make the PeopleID field be my primary key and to be an autonumber.



FINALLY, there is a SAVE button just below the list of fields you just built.

Congratulations – you have built a new database in MySQL, created a user who has privileges to access that database and finally created a table within the database. With the details about that database, the user and their password, you can easily create a connection between your PHP page and the database. At some point (a few lines later in the code) you will have PHP grab the data from the POST and issue a SQL insert statement to move that data into the database.

Defining the DSN (Data Source Name)

Programmers have a name for the connection between the PHP page and the database; they call it a Data Source Name. The DSN contains the answers to three questions required to establish the database connection (Figure 11-5):

1. What kind of database package are you using? In our case, it will be **mysql**.

2. Where is the database located? We will use the term **localhost** since our database is on the same computer the PHP page is on. We don't necessarily know where that is…could be in France for all we know – but it doesn't matter as long as we tell PHP that the database is on the same computer as the PHP is on.

Assuming you have an empty HTML document – just a shell with a head and body, put your php code in the body (it can go pretty much anywhere in the body). Here is the code I need to write to connect my PHP page to my Dating database…

```
<?php
        $dsn = "mysql:host=localhost;dbname=Dating";
        $username = "brickle1_Teacher";
        $password = "pa55word";

        $db = new PDO($dsn, $username, $password);
```

The **"<?php**" and "**?>**" symbols are there to mark this area as PHP code. Since they mark the edges of our code, they are often referred to as "**delimiters**". Remember that any HTML on the page will be read by a browser – your server will have no interest in it. The web server is only interested in the PHP sections.

Notice that I'm starting every variable name with a dollar sign and I'm ending every line of code with a semicolon.

I've named my first variable $dsn because it reminds me of Data Source Name. We just mentioned that our DSN needs to know what kind of database we are using, where it's located and what the database is called. The answers to those three questions are in that first line of code.

Since we will need a valid user with permission to access the database, our next two lines store the name of the user and their password.

The final line establishes the actual connection to the database using the three variables we just created. I've decided to name the connection $db, but you'll often see code written by programmers who have named the connection $conn, since it reminds them of

the connection. The point is that it is not important what the connection is named. Don't worry about the PDO part. This PHP Data Object is a bit like a function that handles the details of setting up the connection. Since I want a new connection, I ask for a new PDO. If you happen to have some background in programming, you recognize that I've created something called an Object – for the rest of us, just think of that last line as a way to finalize the connection.

NOW GET THE DATA FROM THE POST

Once the database exists and we have a connection to it – the rest of the process is easy. We need to grab the data from the post and put a SQL statement together.

Pulling data from the post is pretty simple. Before you look at the code, you need to remember that our form put four pieces of data into the post. It named the data frmFname, frmLname, frmAge and frmGender. As we reach into the post for each piece of data, we need to store the data on our PHP page. We can store the data by creating variables – as before, the variables will all start with a dollar sign.

Check out the code…

```
$fn =
$_POST['frmFname'];
$ln =
$  POST['frmLname'];
```

Let's think about the first line. PHP uses $_POST to pull data from the post. Notice how we refer to the data in the post by enclosing the name of the data in both square brackets and single quotes. Notice the underscore after the dollar sign and note how the word POST is uppercase – yes…it actually matters! Once the data has been collected from the post, I'm storing it in a variable named $fn. When I put my SQL statement together, I'll refer to $fn when I need to know the first name of the person who filled out my form.

Here's what the PHP code looks like so far…

```
<?php
        $dsn = "mysql:host=localhost;dbname=Dating";
        $username = "brickle1_Teacher";
        $password = "pa55word";

        $db = new PDO($dsn, $username, $password);


        $fn = $_POST['frmFname'];
        $ln = $_POST['frmLname'];
        $ag = $_POST['frmAge'];
        $gd = $_POST['frmGender'];
```

11.4 Adding records to the database

Well, it took a long time to get here, but we are now ready to add customer data to our database. We can use a SQL INSERT statement to do the job

To make this as simple as possible, let's say our database only has two field names (fname and lname) in a table named tblPeople. To add Bob Jones to the database, we can write:

```
"INSERT INTO tblPeople (fname, lname) VALUES ('Bob', 'Jones')"
```

Our SQL statement will be slightly more involved since we don't know the exact name of the person trying to join our dating service. The name, age and gender of that person are stored in variables named $fn, $ln, $ag and $gd. These variables become the VALUES in the SQL statement.

Here's what it would look like:

"INSERT INTO tblPeople (fname, lname, age, gender) VALUES ('$fn','$ln','$ag','$gd')"

Notice how we are using single quotes around the variable names and separating them with commas.

One minor complication: I'm not ready to run the SQL statement yet since I'll need $db's help to get that done. Recall that $db represents my connection between PHP and the database. Since I'm not ready to use the SQL statement until I'm on the next line, I need to store the SQL statement in a variable until I'm ready to use it. I'm going to call that variable $sql. Here's the code:

```
$sql = "INSERT INTO tblPeople (fname, lname, age, gender)
        VALUES ('$fn','$ln','$ag','$gd' )";
```

NOTE: You need this typed on ONE line…my word processor wrapped it onto a second line for formatting reasons – but the PHP code will need it on a single line!

How do we get that SQL statement to run? This might actually be the easiest line of code…we simply ask $db to execute the SQL statement.

$db->exec($sql);

That makes our code (almost done…) look like this:

```
<?php
        $dsn = "mysql:host=localhost;dbname=Dating";
        $username = "brickle1_Teacher";
        $password = "pa55word";

        $db = new PDO($dsn, $username, $password);


        $fn = $_POST['frmFname'];
        $ln = $_POST['frmLname'];
        $ag = $_POST['frmAge'];
        $gd = $_POST['frmGender'];

        $sql = "INSERT INTO tblPeople (fname, lname, age, gender)
                VALUES ('$fn','$ln','$ag','$gd' )";
```

Notice the strange hyphen and right angle symbol after $db. This is PHP's way of telling $db what we want done. If you have a programming background, $db represents an object (a new PDO object). The "exec" is a method of the database object.

Congratulations! We have built a PHP page that collects data from a web form, establishes a connection to a database and has executed a SQL Insert statement that pushes the data on our new dating customer into our database.

11.7 Talking back to the Customer!

If you would like to welcome our new member, we can "echo" their name back to them as part of our welcoming message. PHP can echo just about anything to the page. If it echoes something that looks like html, our browser will interpret that code and display it.

Here's a simple paragraph that PHP might echo back to the browser:

echo " Thanks for joining our dating service ";

It would be better if we could include the name of the new customer as part of the paragraph. There are several ways of getting that accomplished…but one way that's easy to follow is to concatenate the pieces together.

Let's say I want the browser to display "Hey Mark, thanks for joining!" That sentence will be the same for each new customer – except for the name. That name will need to be slipped into the sentence. The pieces of the sentence will become:

" Hey "
$fn
", thanks for joining

Notice how the parts that don't change are enclosed in quotes, but the variable holding the person's name is left out in the open for the PHP page to see as a variable.

To tie the pieces together (something called **concatenation**) we use a dot.

Here's the final code:

echo "Hey " . $fn . ", thanks for joining ";

IMPORTANT TO READ

These echoes may seem like a minor addition to the page – but they play a huge role in the quality of your web experience. In fact, without them, Facebook, Twitter, Amazon and most other sites on the net would not be possible. When you log onto Facebook, there is no way they will have built a custom page just for you. The details of your page are sitting in a database. Once you log in, PHP (or something like it) pulls your profile image out of the database and echoes an image tag onto a blank page. Then it gets the list of comments made by you and others and echoes those to the page as well. The echoes are painting a page specific to you…once it is complete, the web server sends it off

Our complete AddCust.php page now looks like this:

```
<html>
    <head>
        <title> The Meet Market </title>
   </head>
<body>

   <?php
        $dsn = “mysql:host=localhost;dbname=Dating”;
        $username = “brickle1_Teacher”;
        $password = “pa55word”;

        $db = new PDO($dsn, $username, $password);


        $fn = $_POST[‘frmFname’];
        $ln = $_POST[‘frmLname’];
        $ag = $_POST[‘frmAge’];
        $gd = $_POST[‘frmGender’];

        $sql = “INSERT INTO tblPeople (fname, lname, age, gender)
                VALUES (‘$fn’,‘$ln’,‘$ag’,‘$gd’ )”;

        $db->exec($sql);

        echo “Hey “   .   $fn   .  “, thanks for joining</p>”;
   ?>
```

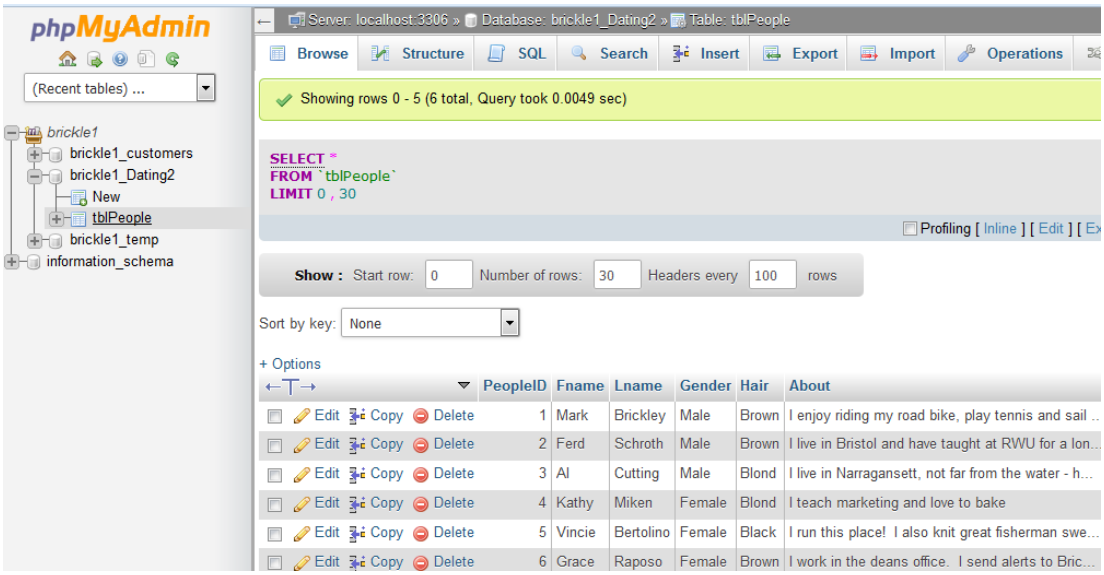
Think about the code for a moment. You should be able to follow it. Let me recap…
1) We established a connection to our database
2) We grabbed data out of the POST
3) We created a SQL statement and executed it
4) and, finally, built a personalized response to our customer

11.8 Testing the Dating System

The most important thing to keep in mind as we try out the system is that PHP lives on a web server. That means our form and php page need to be uploaded to the Public folder on your CuttingEdge account. There is no way for it to work by having Notepad run the page – it must be on the actual internet.
Once the pages have been uploaded to your hosting account, use a browser to display your web form, fill it out and hit submit. If you get the "Hey Mark, thanks for joining" message (obviously the name matches the name you entered on the form), then all is probably well. Go back to the form and enter another person and keep doing that until you have a few customers entered.

On the Control Pancel of your account, get back into PHPmyAdmin, click on the name of your database and the name of your table. That should display any data sitting in the table. I doubt it will be necessary, but you may need to click on "Browse" to see the data.

Here's what my table looked like after entering a few customers:



11.8.2 What can go wrong?

It can be frustrating when a PHP page doesn't work. Aside from typos, the most common issues are:

1. You didn't end a line with a semicolon

2. You forgot the name of the database, the user or their password and tried entering them from memory – be sure to go back to MySQL Databases and check on the correct names. Do you know the name of your table>

3. You probably have a primary key that is set to use an autonumber. That field should NOT be in the SQL statement.

4. Be careful with the field names in the database and the variable names on the PHP page! You have names, ages and genders on the web Form, on the PHP page and in the database table. I've found that people often get lost and forget what they called fields in each of these places. Here's a simple grid that I complete when I'm building one of these systems. For the dating system we just completed, it might look like this:

Collect.html	**AddCust.asp**	**tblPeople**
frmFname	Fn	Fname
frmLname	Ln	Lname
frmAge	Age	Age
frmGender	Sex	Gender

11.9 Searching a database with PHP

The second major topic in this chapter revolves around web driven searches for data. It would be great to be able to search for a date. Sure, we use this example to hold your interest, but the techniques apply to any web system that let's customers search for clothes, books or other items.

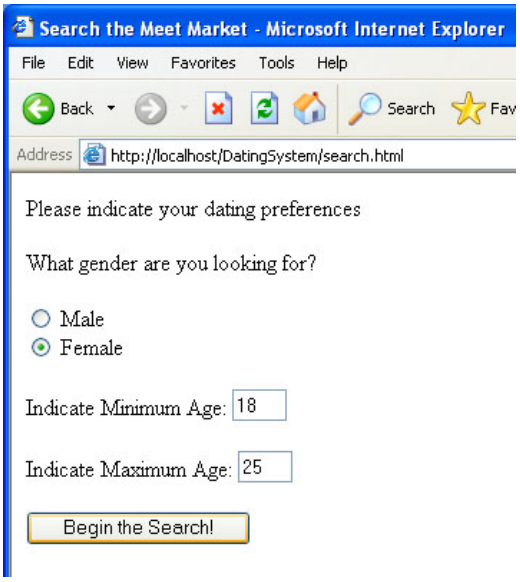
To pull this off we'll need two pages, one that asks what they are looking for and a second page (this one is PHP) that does a SQL SELECT statement.

Let's start with a page that asks for their "requirements" in a date. To keep things simple, I'll just ask about the gender and an age range. The customer can use this page to indicate they are looking for a male or female between one age and another. Once the user clicks the "Begin the Search" button (our 'submit' button), the web form will post the user's search criteria to a PHP page called Results.php that will then send a SQL SELECT statement to the Dating database and hand the results back to the browser.

You can probably write the code for the web Form by yourself. The whole page is normal html and uses a Form to determine the customer's dating requirements.

```
<html>
  <head>
     <title> Search the Meet Market  </title>
  </head>

<body>
<center> Please indicate your dating requirements </center>
<form     method = "POST"       action = "Results.asp">

I'm looking for a: <p>
   <input    type = "radio"   name = "frmGender"   value = "Male">
Male <br>
   <input    type = "radio"   name = "frmGender"   value = "Female">
Female <p>

Minimum Age:  <input type = "number" size = "2" name =
"frmMinAge" >
Maximum Age:  <input type = "number" size = "2" name =
```

Here's the form in action. It lacks graphics to give it that pizzazz that will give the site a professional look – you can add that later. It also lacks any error-trapping code to make sure the Minimum age is actually lower than the Maximum age. Let's not worry about that here (you learned how to check the data if you read the Geeks installment for Chapter 10).



The customer's dating preference will get passed to the PHP page named *Results.php*. That page will create a connection to our database. We can then use the connection to execute a SQL statement to find all records that relate to the client's dating preference.

Let's get that much done before we worry about how to display a long list of data on the web page that the customer will see.

Finding potential dates in the Dating database
As with all forms, the Search.html Form handed its data off to the server for processing. Thinking back to the Form tag, you know that the *Action* was going to be handled by an PHP page by the name of *Results.php*. That's the page we need to build next.

11.10 Building the Results.php page

Results.php needs to execute a SQL query based on the data our web form handed it. The hand-off included the gender as well as a minimum and maximum age. Only people meeting those conditions will be in the list that we send back to the browser.

You may not be comfortable with PHP code yet – but check out how much of this code you've already seen. I'll focus on the code between the PHP delimiters.

```
<?php
        $dsn = "mysql:host=localhost;dbname=Dating";
        $username = "brickle1_Teacher";
        $password = "pa55word";

        $db = new PDO($dsn, $username, $password);

        $gd = $_POST['frmGender'];
        $min = $_POST['frmMinAge'];
        $max = $_POST['frmMaxAge'];

        $sql = "SELECT * FROM tblPeople WHERE gender = '$gd'
                AND age >= $min
```

As with the previous PHP page that collected data from a form and inserted it into a table, this new page starts off the same way.

1) Establish a connection to the database

2) Retrieve data from the POST

3) Build a SQL statement…this time we are SELECTING data from the table

4) Run the SQL statement

Before we move on, let me explain the SQL statement. I'm sure you can read the basic statement – Get all the data from the People table where the Gender matches what they asked for and the person's age is at least the minimum they asked for and no more than the maximum they asked for – and make sure the list is sorted by age.

One minor issue to discuss: when we write formulas in any computer system (Excel, Access etc.) we already know that words get quotes around them and numbers don't. That's why I have single quotes around the gender, but no quotes around the ages. If I were to write this SQL statement using actual data, it might make more sense to you:

e.g.

SELECT * FROM tblPeople WHERE gender = **'**Female**'** and age >=20 AND age <=25

Pay attention to the line that runs the SQL statement. I'm asking $db to QUERY the table rather than EXEC something. When a SQL statement changes a database – as it would with an INSERT, UPDATE or DELETE - that would be an "EXEC". We use

QUERY when the command is simply looking for data and making no changes to the database.

The only other thing to keep in mind is that I've spaced the SQL out in hopes it is easier for you to see what's going on. When this is written for real, the SQL statement must appear on a single line…

11.11 Using a RecordSet to store search results

At this stage we have asked for a list of records that match the dating interests of our customer – and that creates a bit of a problem. We actually need a place to hold that list until we can get around to writing some code to display it! I'm going to create a variable named $answer to hold the results from our database query.

```
$answer = $db-
```

Now, when the query runs, the results will fall into $answer. The results may include hundreds of potential dates. We need a way to display these results one-by-one. If you have ever programmed before, we are about to write a loop. For those who haven't programmed, we are going to grab the first line of results, echo them to the page, then repeat that process until $answer is empty.

Note that the field names mentioned in the ResultSet are actually the names of fields located in the database table – that's why I'm using the names "***Fname***" and "***Lname***" rather than frmFname and frmLname (those are the names of fields located on the web Form. The code to display the results is nearly English-like!

```
foreach ($answer as $ans) {

        echo $ans['Fname'] . "<br>" ;
        echo $ans['Lname'] . "<br>" ;
        echo $ans['Age']   . "<br>" ;
        echo "<hr><br />";
}
```

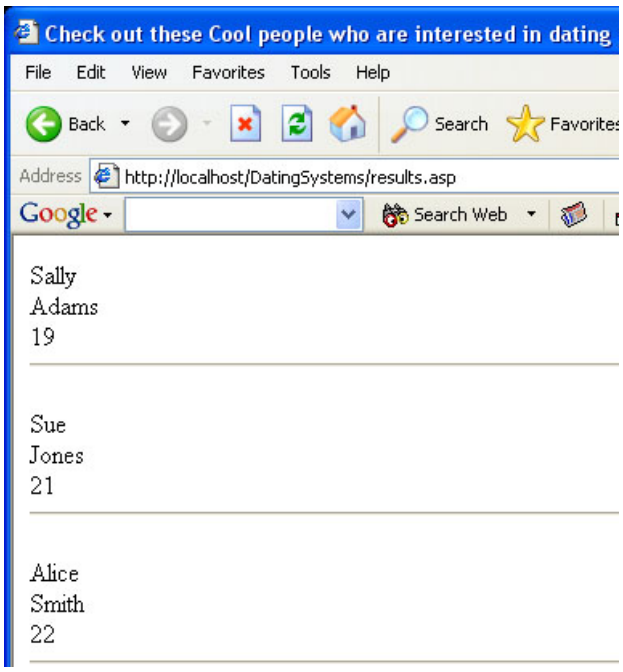
There are a number of ways to build a loop in PHP. The "foreach" loop grabs one line of data from $answer, displays that data and repeats the process until $answer is empty. The curly braces mark the upper and lower edges of the loop. One weird aspect of the "Foreach" loop is that we need to set up an alias for use within the loop…think of it this way **"for each line in $answer, known as $ans within this loop…do what is inside the curly braces"**

Here's the complete code for the Results.php page:

```
<html>

<head>
<title> Check out these Cool people who are interested in dating  </title>
</head>

<body>

<?php
        $dsn = "mysql:host=localhost;dbname=Dating";
        $username = "brickle1_Teacher";
        $password = "pa55word";

        $db = new PDO($dsn, $username, $password);

        $gd =    $_POST['frmGender'];
        $min =  $_POST['frmMinAge'];
        $max = $_POST['frmMaxAge'];

        $sql = "SELECT * FROM tblPeople WHERE gender = '$gd'
                AND age >= $min
                        AND age <= $max
                                ORDER BY age";

        $answer = $db->query($sql)

        foreach ($answer as $ans) {
          echo $ans['Fname'] . "<br>" ;
          echo $ans['Lname'] . "<br>" ;
          echo $ans['Age']    . "<br>" ;
          echo "<br><br>";
        }
```

11.12 Testing the Dating System's search capabilities

It's time to try this system out. Once again, remember that we need to enter a URL in the browser's address bar. This time the page we need is the Search.html form that asks the user about the person they want to date. Here's what it looks like when I search for Females between 18 and 24. Notice that our SQL statement has sorted the results by age. The lines that separate each potential date are simply HTML horizontal rule tags (<hr>).



Just for Geeks
An Introduction to Cookies

I buy books from a fairly popular online store. When I reach their site, the opening page says "Welcome Back Mark!" You've probably experienced the same thing and might have wondered how they knew your name. Well, the answer is simple – the company dropped a ***cookie*** onto your computer when you first visited. That cookie is nothing more than a small text file that can be used to store anything from your name to your password, your music or book preference and possibly the date of your most recent visit.

Each time you visit a site your browser checks to see if a cookie exists and sends it along with the request for the web page you asked for (it travels as part of the HTTP header). The web server can then read the value of the cookie and insert your name into the page before sending the requested page back to the browser.

Why the need for cookies?

Why bother using cookies? The answer is pretty simple – the Web (HTTP) has no memory!

Let's say that Harry is shopping for his wife's birthday present and has asked your site to display the sweaters you carry. You have a clever ASP page that responds with descriptions, prices and great color photos. Unfortunately, once that page gets sent, your web site forgets who asked for the sweaters. When the next request arrives, there is no way for the clothing store to know whether this is still Harry – or whether some new customer has entered the web site. Imagine doing business in a real store at which salesmen immediately forget which customers they are dealing with! Computer people have a fancy term for this short term memory – the word is "***stateless***". A system that *does* remember things is referred to as a "stateful" system.

A stateless system means that an internet store can't follow a customer as they move from page to page within your web site. If Harry decides that a watch would be a better gift than a sweater, there should be a way to follow Harry to the jewelry department and know that you are still dealing with the Harry from Rhode Island!

The solution has been to use cookies. Even if we don't know the customer's name, it would be nice to know that this customer is the same person we were dealing with a moment ago. Many web sites send a simple cookie to your browser that contains a "Session ID" – the web site might not know your name, but they can at least assign you a number that will last as long as you are browsing their web site. The browser destroys these session cookies the moment you close the browser. For longer term relationships,

web sites can write a cookie to your computer and tell the cookie to self destruct ten years from now.

The basic mechanics of reading & writing cookies

The code required to drop a cookie onto a client machine is very simple. Since the cookie is being written by the web server, we will use the ***Response*** 'object'.

It is important to remember that
Request is used when data is coming *from the client's web browser*

To set a cookie named "fname" to the value "Mark", you could write

```
Response.Cookies("fname") =
```

In the following example, I have placed the code into the AddCust.asp page. Keep in mind, the "Collect" page (a web form) has just passed a collection of data to AddCust (including the customer's name). Since AddCust now knows the name of the new dating client, it can drop that name onto the client's machine. **The code actually goes above the opening HTML tag.**

Notice how the cookie is being given the name "fname" and is set to the value that came in from the web form – request.Form("frmFname"). The cookie is given an expiration date, after which it will be deleted. Without an expiration date, the cookie lasts only as long as your current browser session. A cookie that is stored on your hard drive until some expiration date is said to be ***persistent***. A cookie that lasts only as long as you are on the web, is said to be a ***session*** cookie – since it lasts only as long as your browsing session.

```
<%
Response.Cookies("fname")=
Request.Form("frmFname")
```

To read an existing cookie, you use ***Request.Cookies*** (remember, *request* is used when data is coming from the client's machine). Here's a line I placed in the Results.asp page. That's the page that finds potential dates for clients. To make things a bit more personal, I make sure I display the value of the cookie that 'knows' your name. You could put this code just above the loop that displays the contents of the recordset.

```
Thanks for using our service <% =request.cookies("Fname")%> Check out
```

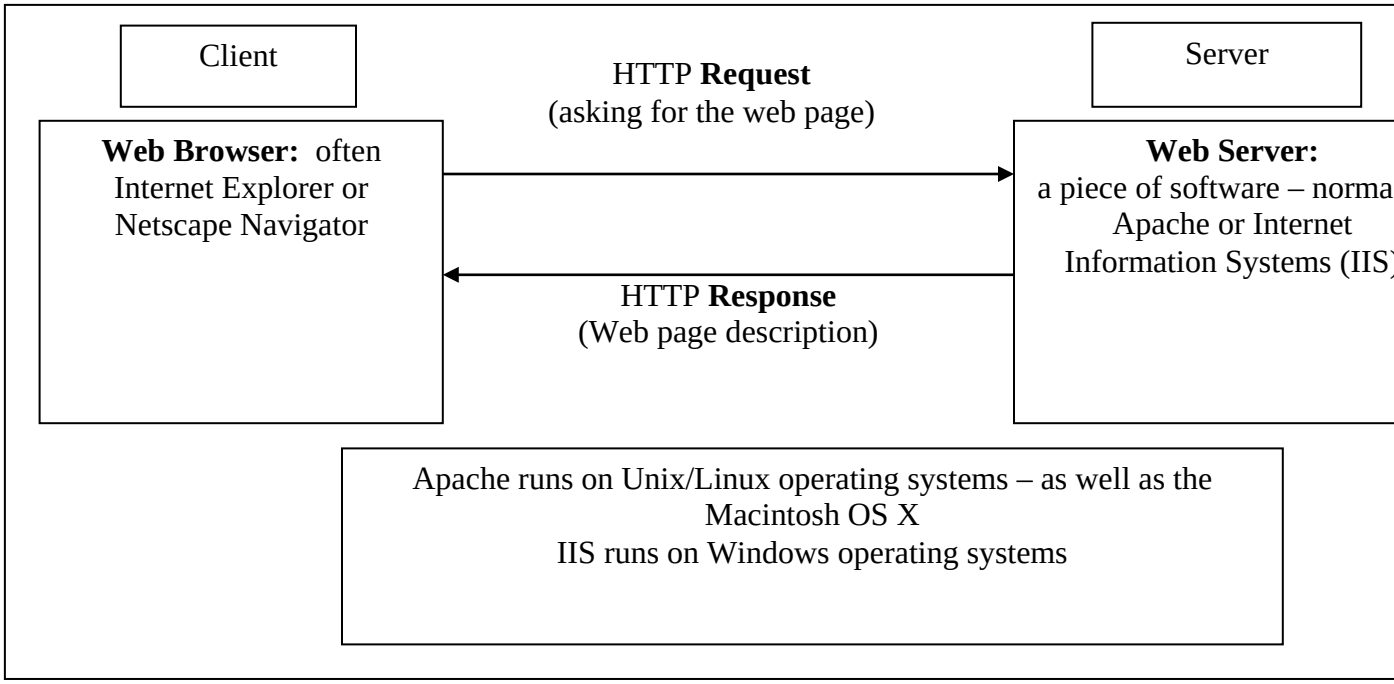
Sometimes cookies contain several values. When that happens, the cookie is said to have Keys (each entry is a Key). To pull this off, the cookie is given a name and then the Key is given a name. Check out the following example of a cookie named Customer that has two Keys – one for Fname and one for Lname.

```
Response.Cookies ("Customer") ("Fname") =
"Bob"
```

How it works

The word Request and Response might seem familiar to you. In chapter One we introduced the client / server model of the web and used a diagram that depicted web page Requests from the browser and web page Responses by the web server. The Request and Response are part of HTTP and are used to transport data from one machine to another. Each package of data included a Header – and that's where the browser places the cookie for its trip to the server. The header of the Request package is also where data from a web form is placed. This is why we refer to the cookie as Request.Cookies and why we refer to data from a web form as Request.Form.

The web server can also utilize the header for sending things (this time the header of a Response package). The header of the Response is where the Cookie travels on its way from the web server to the web browser. This explains why we write a Cookie by saying Response.Cookies.



The Browser's Role

The browser manages the collection of cookies. When you start the browser it reads the collection of cookies into memory and prepares to send those cookies off to sites you decide to browse to. When you close the browser, the cookie collection is written back to the hard drive. The browser checks the expiration date of each cookie and deletes those that have reached their expiration.

Each browser has its own place for storing cookies. In the case of Internet Explorer, you can check for cookies by looking in…

```
C:\Documents and Settings\Some User
```

Cookie Security

As a security precaution, a browser will only record a cookie for the domain trying to set it and will only send a cookie for the domain you asked for. In other words, if you browse for www.amazon.com, the browser will only send the cookie that was originally written by Amazon. The browser will also make sure that www.amazon.com can only write a cookie that mentions Amazon.com – Amazon can't read or write cookies for other web sites…

Summary

Web/Database integration is where the action is in the world of internet commerce. It's a complex topic involving real computer code – for the most part, we've left the world of wizards behind.

So what have you learned? Hopefully you understand the roles played by web browsers and web servers and the division of labor that occurs in client/server systems. The browser plays a pivotal role in collecting data and search criteria from customers through the use of web forms. These forms are able to *post* their data to asp pages sitting on web servers. Code on the ASP pages helps to establish connections to databases, using *DSN*s defined in the Windows Control Panel. Once a connection exists, SQL statements can be used to interact with the database. INSERT statements are used to place new data into the database, while SELECT statements retrieve data requested by users. We used the concept of a RecordSet to hold results of database searches and a programming technique called a loop to work our way through the list of results.

For the Geeks among you, we focused on the use of Cookies. Not only do these cookies provide some personalization (Welcome Back Sally!), but they also provide a way to track customers as they move from page to page within your web site.

Our goal throughout this chapter has been to relate our activities to the world of internet commerce. We collected data about new customers for our online dating service – but we could as easily have been collecting shipping and billing data for an online store. Our attempts to search for a date resemble many of the searches you have done online – either through search engines or at online bookstores!

Keywords

Active Server Pages (ASP)
Dim
Date Source Name (DSN)
EOF
GET
Localhost
Loops
RecordSet
Request
Response
Server Components
- ADODB
- AdRotator
- ContentRotator

VBscript

Geeks only…

Cookie
Persistent
Session
Stateless vs. Stateful

Review Questions

1. Quite often you'll notice criteria being sent as part of a URL. A Google search for Pontiac Solstice might be displayed as

http://www.google.com/search?hl=en&q=Pontiac+Solstice This approach to passing data to the web server is an example of

a) the POST Method
b) the GET Method
c) the Method being set to "Action"
d) the Method set to Google.asp

2. Of the following, what does a DSN store?
   a) the path to the database
   b) the name of the database
   c) the type of database (Access, SQLserver etc.)
   d) all of the above
   e) only choice a and b

3. Which symbols are 'delimiters' for asp code?
   a) <% and %>
   b) < and />
   c) <* and />
   d) %> and *>

4. Request is used to refer to data sent by a browser. For example, Request.Form refers to data being Posted by a web form. What would you use to let a server send things to the user (browser)?
   a) Execute
   b) Send
   c) Push
   d) Response

5. Given the previous question, which command would send a cookie from the server to the user?
   a) Execute.Cookies
   b) Send.Cookies
   c) Push.Cookies
   d) Response.Cookies

6. A RecordSet is used to
   a) hold the results of a SQL statement
   b) tell you how large the data collection is
   c) help you record data to a database table (i.e. write to a table)
   d) define the criteria you are searching for

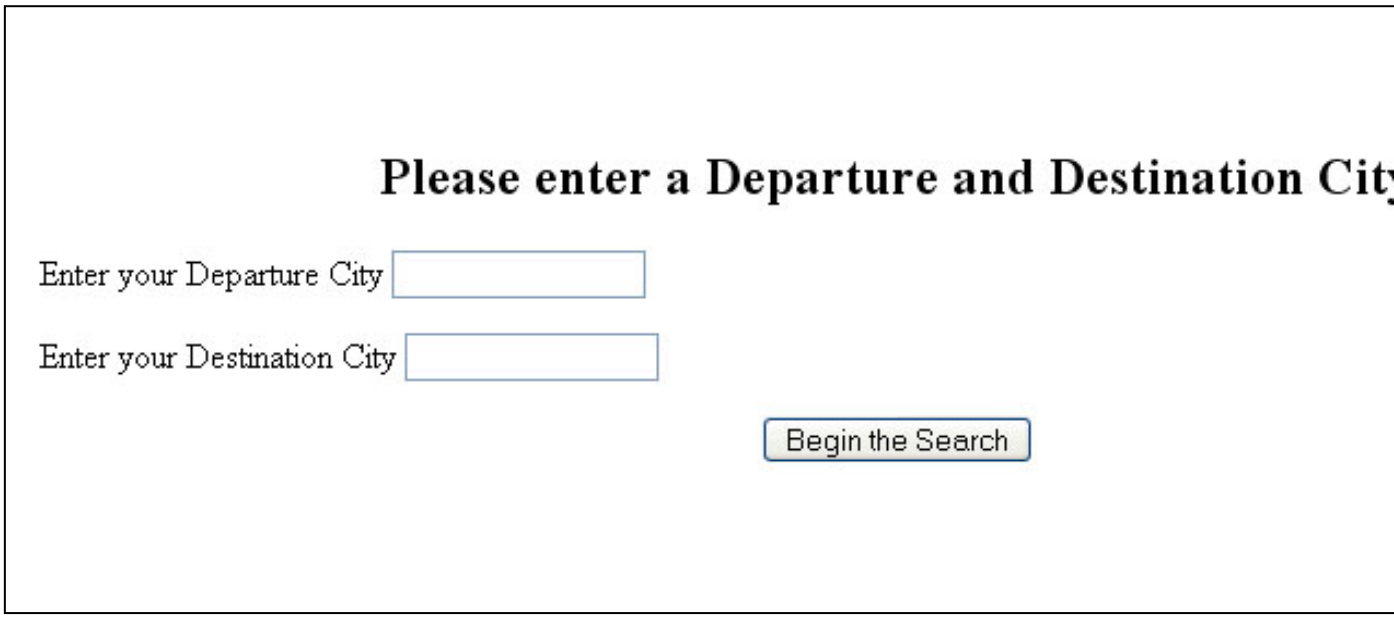
7. A web form lists the Action as "xxx.asp". This setting tells the browser that
   a) it needs to pass the form's data to a page named xxx.asp
   b) xxx.asp needs to be created by the browser
   c) xxx.asp must be loaded by the browser before data can be transferred

d) xxx.asp will spring into action once the user loads that page

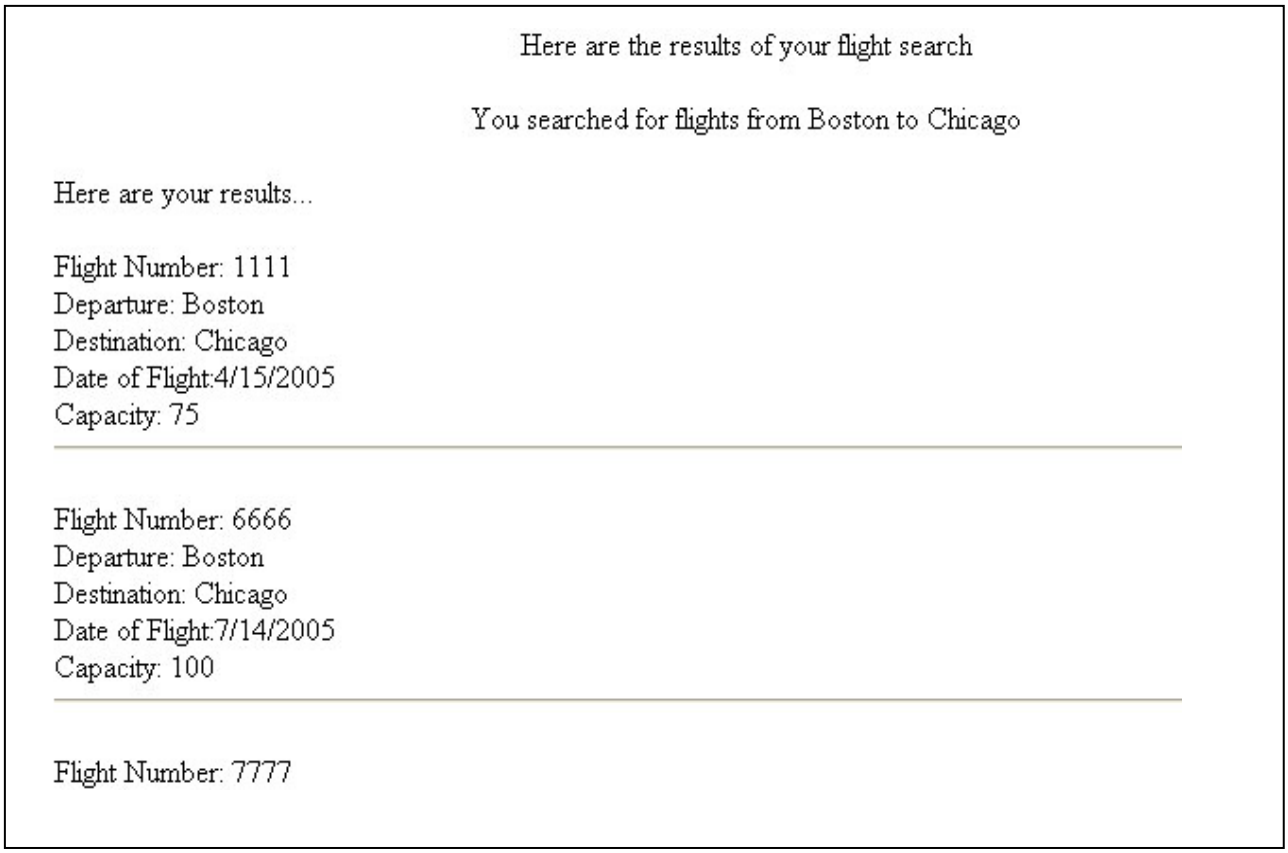
Exercises

1. You have a database named Flights that you've been working with for most of this book. Take a quick look at the tblFlights table to remind yourself what the field names are and what your data looks like.

   Build a form that will let you search let you search for flights from some Departure City to some Destination City. The form might look something like this:



Once you have the Form built, create an asp page that will find the appropriate flights for you. Here I've searched for flights from Boston to Chicago and the asp page is displaying the results of the search.

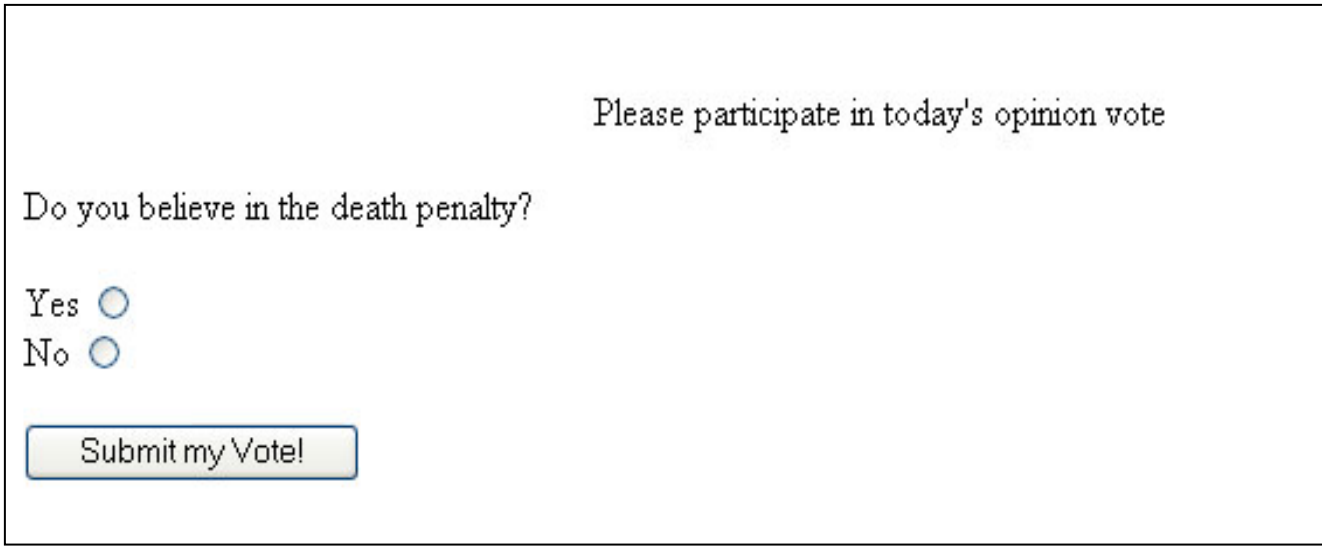


2. Have you ever participated in an online vote? There are lots of web sites that run quick opinion polls – CNN might ask if we agree with the President's new plan for taxes, ESPN might have a list of the basketball teams in the "Final Four" and ask us to pick the team we think will win.

   For this assignment, create a database named Votes.mdb and establish a table named tblVotes. Let the tblVotes table have two fields:

   VoteNum (an autonumber and your primary key)
   Vote (a text field with a length of 3 – it need to hold Yes and No

   Create a web Form to present your question and set up some radio buttons for the vote. It might look like this:



Once you have the Form, create an ASP page to INSERT the votes into your tblVotes table. Be sure to confirm that you recorded the vote and let them know what you recorded (Yes or No). Print out the notepad code for both the web Form and the ASP page. Be sure you also print the voting table to show the votes.

Something of a Challenge…

For a challenge… once the ASP page records the vote, it would be a great feature if the page then queried the database and found the number of Yes and No votes. This will take a bit of thinking.

Some hints:

You know how to write a SQL statement to count the votes, provided the vote is "Yes" and you can do the same thing with the "No" votes.

If the answer is held in a RecordSet, what will it be called? Normally a RecordSet holds the fields from your table and you already know the names of each field. How will you know the name given to the vote "count"? Think back to SQL and you'll recall the word "AS" that can be used in a SQL statement. The word "AS" creates a derived field that will be stored in the RecordSet just as any other field will be.

The final ASP page might look something like this when it runs:

Thanks for your Vote... we have you down as a No on this issue

So far, we have recorded 5 Yes votes and 3 No Votes