Integration

How do I get my FRONT END

(for example, an HTML web page displayed by a browser)

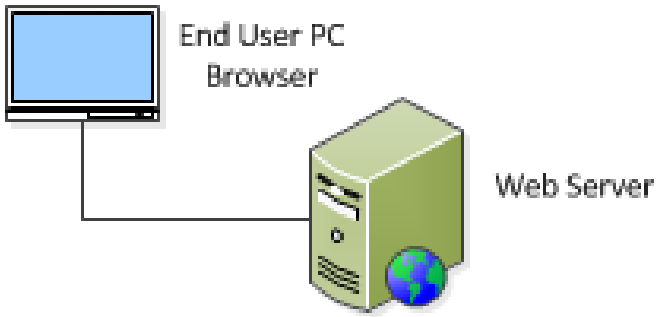
To talk to my BACK END ?

(for example, a MySQL database)

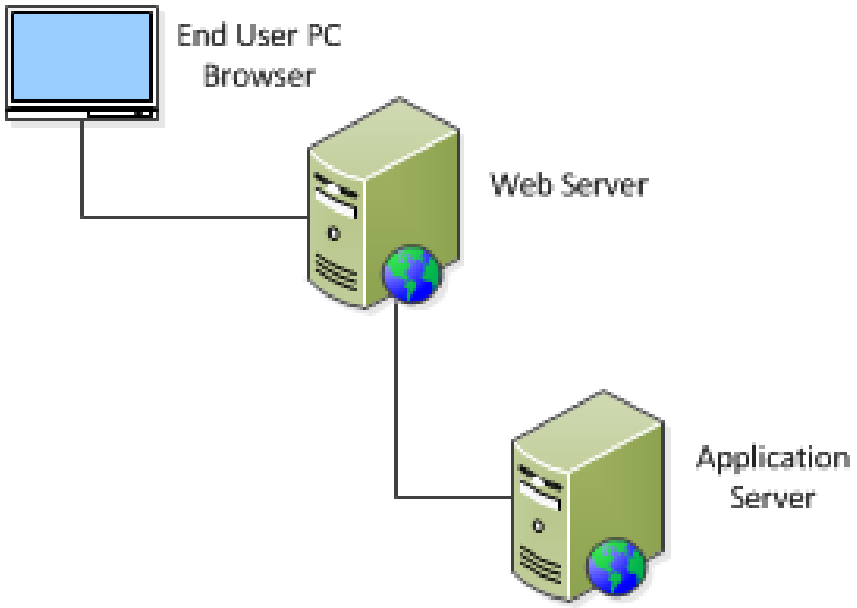
Suppose my application must

- display a form and collect user data entry
- find and display data from the database
- update data in the database

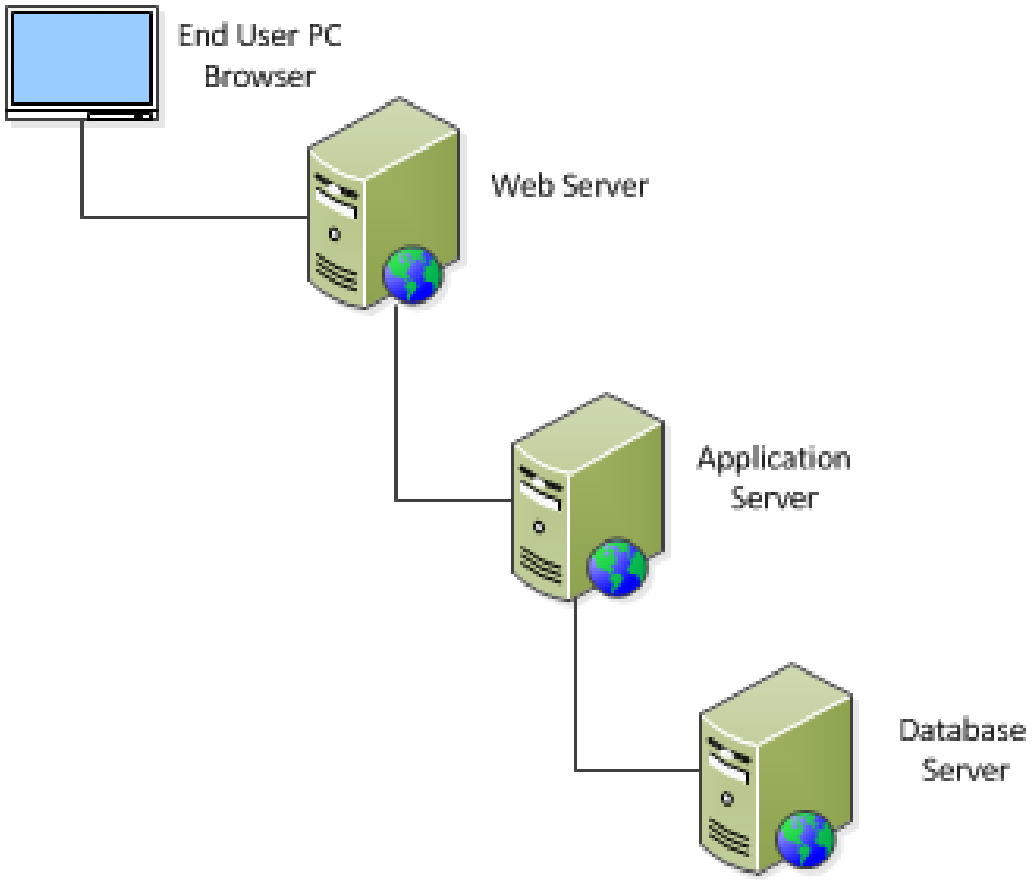
Application Architecture



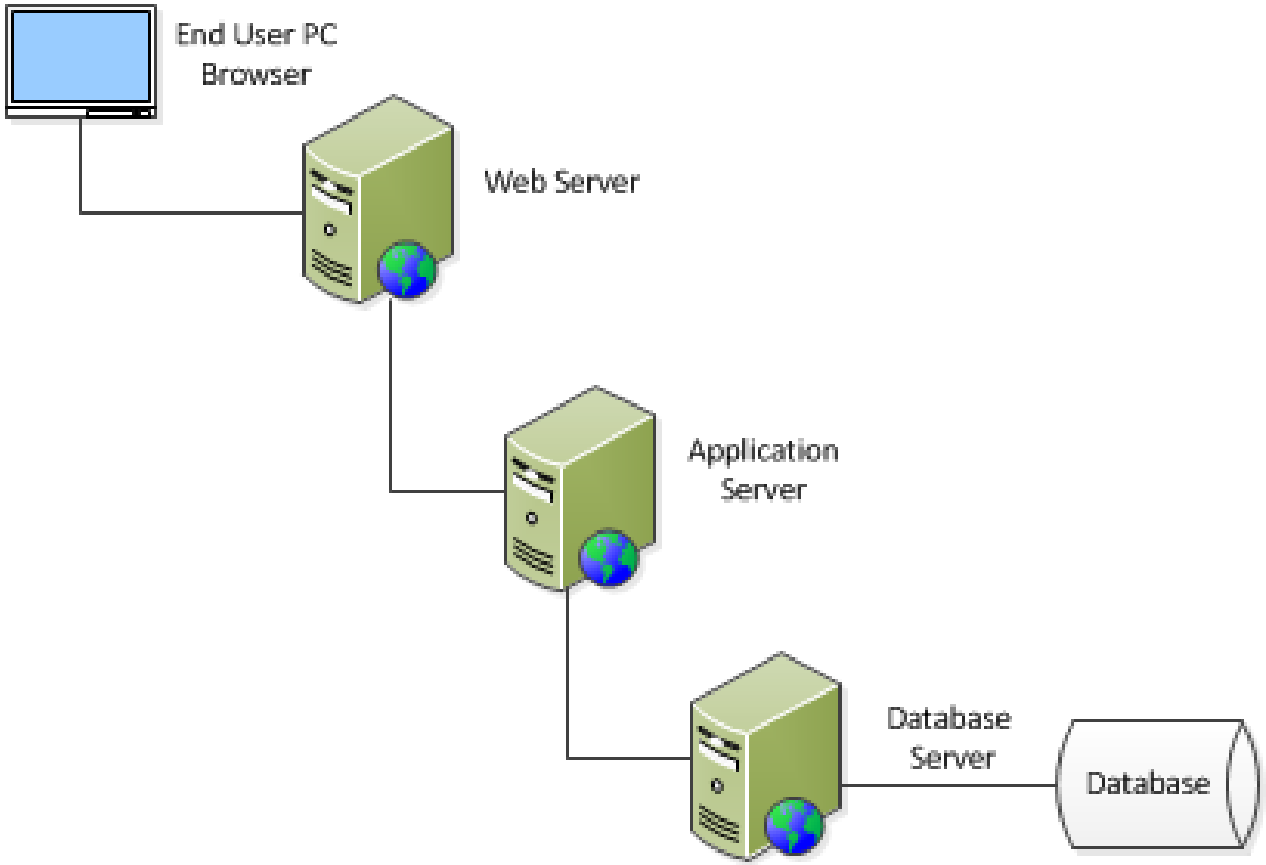
Application Architecture



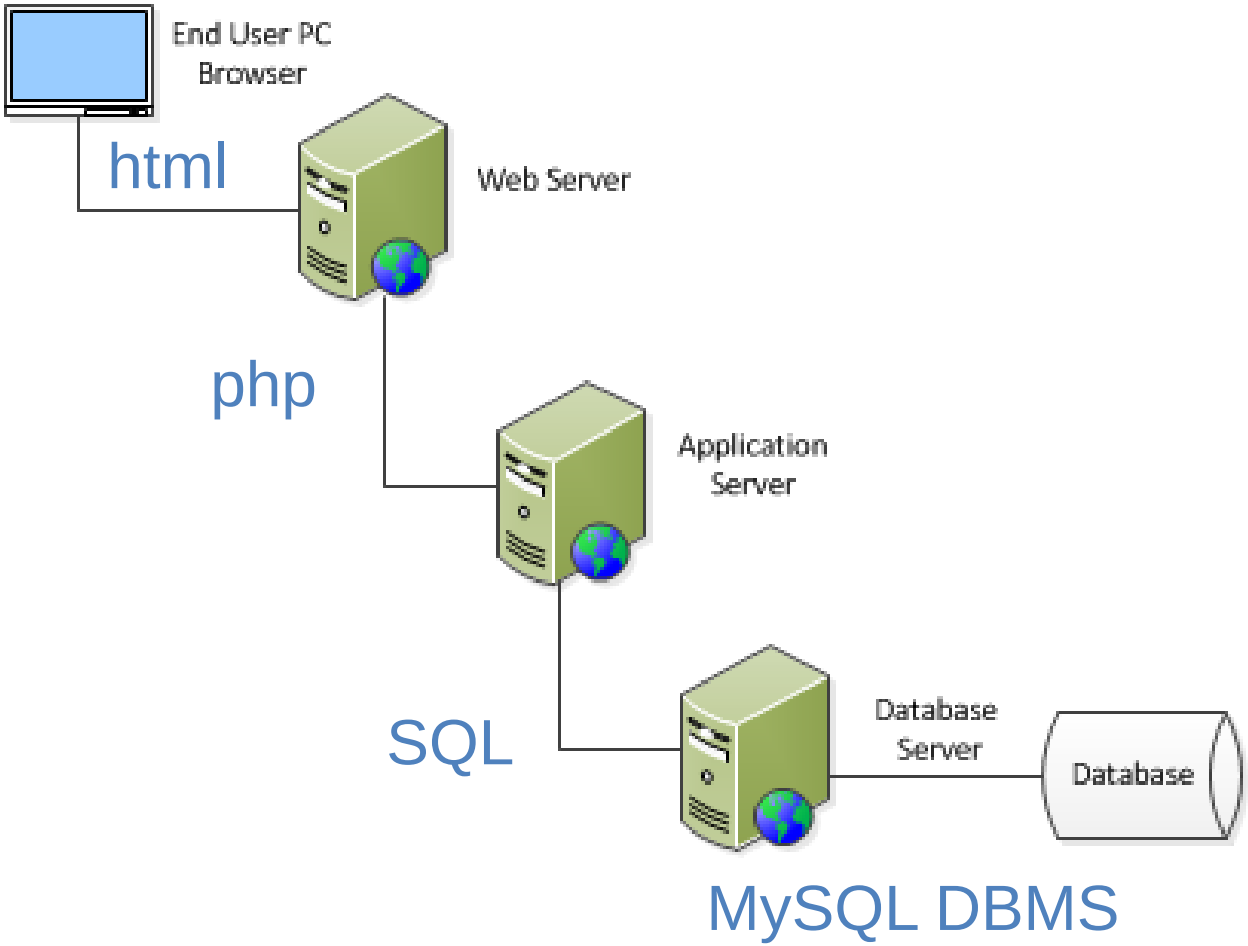
Application Architecture



Application Architecture



Application Architecture



Integration

Three Components:

- HTML – Takes a marked up file and renders it in the browser
- PHP – A server-side scripting language
- SQL – Communicates with the database server

HTML Basics

A "markup" language –

- marks text for displaying in browser,
- embeds images & links,
- displays forms,
- invokes HTTP calls to the web server

Integration

- Tags

```
<tag attributename= "attributevalue">
</endtag>
```

- there are 4 basic attributes for every tag

`id="xxxx"` – identifies it

`class="classname"` – ties it into a group for style

`style="xxxxx"` – where xxx is a list of style elements

`title="xxxxx"` – adds misc info to the tag

- HTML is NOT case sensitive.
  Rule = always enter tags in lower case.
- Anything in quotes *might be* case sensitive
  Like attribute values: `<img src="picture.jpg">`
- Spaces: Many spaces = one space.
- Always use end tags.
- Nesting elements: From `<tag>` to `</endtag>`
- Good habit: Quote all attribute values.

Basic Document Structure:

```
<html>
  <head>
     <title>
     <meta> - stuff for web crawlers
  </head>
  <body>
     your content goes here
  </body>
</html>
```

`<h1>, <h2>` through `<h6>` = Headings

`<p>` = Paragraph -- causes a line break

`<div>` = Division -- for grouping `<p>`s for alignment or style

`<span>` does the same grouping as `<div>` but without the line break

`Align="right", "left" or "center" or "justify"`

`<blockquote>` causes a return, indents

`<pre>` for displaying text exactly as you entered it

- Lists

`<ol>` = Ordered List , attribute type=A,a,I,I,1 start=xx
`<ul>` = Unordered List (bullets) attr type=disc, circle, square
`<li>` = identifies list items

- Horizontal Lines

`<hr>` attributes: `size="10"` pixels (height)
`width="xx"` in pixels or percentage
`align="xxx"`
`noshade="noshade"`

- Blank Lines

`<br>` break
` ` – non-breaking space (embedded blanks)

- Comments

`<! -- xxxx -- >`

Linking

- Hyperlink
  - Tells the browser to display another document
  - Can be on the same site, or ANYWHERE
  - Implemented via `<a>` tag ("a" is for "anchor")
  - Uses the `href="xxxxx"` attribute to identify where the link takes you
  - Browser identifies a Link via underscore and color
    - Avoid `<u>` tag
  - Cursor changes shape on "mouseover" of the link
  - Status Bar shows URL of the link on "mouseover"

Linking

- Managing Hyperlink Colors
  - `<body link="xxxx", alink="xxxx", vlink="xxxx">`
  - `xxxx` = valid color name
- Absolute versus Relative URL
  - `Protocol://host.domain.tld/fullpath/file.htm`
  - `File.htm`
  - Relative location set by current page OR `<base>` tag in `<head>` section
  - `<base href=http://host.domain.tld/path>`
  - No file name ➔ uses `"index.htm"` or `"default.htm"`

Linking

- URL
  - Directory path – filename plus extension
- Linking to a "marker"
  - Defined by `name="xxxx"` or `id="xxxx"` attribute
  - **Name** is older. **Id** is XHTML compliant. Use both.

Images

- Two basic types
  - .gif –Graphics Interchange Format
  - .jpg – Joint Photographic Experts Group

.gif

- For illustrations
- Good with large areas of contiguous color
- Compresses nicely.
- "lossless"
- Limited to 256 colors
- Uses dithering to simulate other colors
- Allows transparency
- Supports animation

.jpg

- For photos
- Good with large numbers of various colors
- "lossy" compression.
- Millions of colors
- No transparency
- No animation

Images

- `<img>` tag attributes
  - `Alt="xxxxx"` ➔ text to display when image is not available
  - `Align="xxxxx"` ➔ flows text around the image
  - `Hspace="xx", Vspace="xx"` ➔ puts a buffer of space around the image
  - `Height="xx", Width="xx"` ➔ resizes the image
    - Reserves the space to speed in page loading
- `<img>` as "link" (i.e. a "button")

```
<a href="week3_ex4.htm">
<img src="Button1_on.gif" /> </a>
```

Other Formatting Tags

- `<font>` attributes
  - `color="xxxx", size="xx", face="xxxx"`
  - Size ranges from 1 to 7. 3 is default.
  - Can use "+1"
  - Colors – RGB values or Name
- `<body>` attributes
  - `Bgcolor="xxxx"`
  - `Text="xxxx"`
  - `Link, alink, vlink`
  - `Background="image.gif"`
  - `Topmargin="xx", leftmargin="xx"`

Tables

- Used Primarily for LAYOUT options
- Rows and Columns, "cells"
- The browser will SIZE the table large enough to hold the cells' contents
- Every row gets the same number of "cell positions", whether or not you define or use them
- Table Tags:
  - `<table>` defines the table
  - `<tr>` defines a table row
  - `<th>` defines the table's header cell content
  - `<td>` defines the table's data cell content

Tables

- `<table>` attributes
  - `bgcolor` – just like <body>
  - `border="xx"` size in pixels. Default = no border
  - `cellspacing="xx"` size of cell space in pixels (space between cells)
  - `cellpadding="size"` of cell pad in pixels (space around contents within cell)
  - `width="nn"` (pixels or percent) Size of table
  - `align="xxx"` left, right, center – aligns the table on the page

Tables

- `<tr>` defines a table ROW
- `<tr>` attributes
  - `align="xxx"` (left, right, center, justify) aligns cell contents
  - `bgcolor` – just like <body>
- `<td>` defines a table cell
- `<td>` attributes
  - `align="xxx"` (left, right, center, justify) aligns cell contents
  - `bgcolor` – just like <body>
  - `Colspan="nn"` – number of columns cell spans
  - `Rowspan="nn"` – number of rows cell spans

php Basics

- php pages MUST be invoked via the URL rather than simply opening the file in your browser. Why?
  - They are ***SERVER SIDE SCRIPTS***
  - They are EXECUTED by the ***web server***
  - Not just PARSED by the ***browser***

- Your web server has a default location/path where it expects to find executable php scripts.
- Execute the php files (via the Apache web server) by entering a URL into your browser: `localhost/filename.php`

php Basics

- php code is typically embedded within an HTML page
- php code is embedded within special tags:

```
<?php
    php code…
    php code…
?>
```

- You may also see:

```
<?
    php code…
    php code…
?>
```

- Or:

```
<script language="php">
    php code…
    php code…
</script>
```

php Basics

- To send text to the browser

```
<?php
    echo "Hello World";
    echo 'Hello World';
    print 'Hello World';
    print "Hello World";
?>
```

Note: all statements end with " ; "

- To send a quote to the browser

```
<?php
   echo "Hello Wayne's World";
?>
```

- This won't work

```
<?php
   echo 'Hello Wayne's World';
?>
```

- Use " \ " (backslash) as *escape* character

```
<?php
   echo "Hello Wayne\'s World";
?>
```

- To send text + html tags to the browser

```
<?php
    echo "<b>Hello</b><em>World</em>";
?>
```

- Multiple lines (demo)

```
<?php
    echo "this text is spread across
          multiple lines in php";
?>
```

- Yields multiple lines in HTML, but not when rendered

- Commenting your code (for the sake of human readers)
  - HTML comments `<!-- xxxxxx -->`
  - php comments
    - Use # or // for single line comments
    - Use /* through */ for multi line comments

- Comments

```
# Created August 27, 2007
# Created by Larry E. Ullman
# This script does nothing much.
echo '<p>This is a line of text.
      <br>This is another line of text.</p>';
/*echo 'This line will not be executed.';*/
echo "<p>Now I'm done.</p>"; // End of PHP code.
```

- Note on debugging
  - **HTML with php can be very unforgiving and difficult to debug**
  - **Tips:**
    - **Comment out stuff to see what's not working**
    - **Use notepad++**
      - **It matches tags with end tags**
      - **Color codes your HTML & PHP**

- Variables
  - Must be named
  - Name must start with a $
  - Names may contain letters, numbers, and "_"
  - First character after $ must be a letter or "_"
  - Names ARE case sensitive
    - `$name` does not equal `$Name`
  - Assigned values with "="

```
$name = "Desmond";
```

- Php comes with some pre-defined variables
  - $GLOBALS — References all variables available in global scope
  - $_SERVER — Server and execution environment information
  - $_GET — HTTP GET variables
  - $_POST — HTTP POST variables
  - $_FILES — HTTP File Upload variables
  - $_REQUEST — HTTP Request variables
  - $_SESSION — Session variables
  - $_ENV — Environment variables
  - $_COOKIE — HTTP Cookies
  - $php_errormsg — The previous error message
  - $HTTP_RAW_POST_DATA — Raw POST data
  - $http_response_header — HTTP response headers
  - $argc — The number of arguments passed to script
  - $argv — Array of arguments passed to script

- Variables
  - Get in the habit of using a CONSISTENT naming scheme
    - Lower case, underscores, mixed caps
  - No need to initialize
  - No need to declare the variable type
  - Easy to switch types

- String Variables
  - Variable is assigned any value in quotes (single or double)
  - Any new value assigned overwrites the old value
  - No strict limit on length
  - Concatenated with " . "

```
$first_name = "Kate";
$last_name = "Austen";
$full_name = $first_name." ".$last_name;
echo "$full_name";
```

- Numeric Variables
  - Variable is assigned any numeric value without quotes
  - Any new value assigned overwrites the old value
  - Don’t use commas for thousands
  - Assumed positive
  - Arithmetic operators

    `+  -  *  /  ++  --`
  - Arithmetic Functions
    - `round(xxx, yyy)`
    - `number_format(xxx, yyy)`

      Where xxx is the number and yyy is number of decimal place

Using Quotes

- Double quoted strings resolve values
- Single quoted strings do not resolve values (quote_demo.php)
- `$var = 'test';`

  `echo "var equals $var";` //yields `var equals test`

  `echo 'var equals $var';` //yields `var equals $var`

  `echo "\$var equals $var";` //yields `$var equals test`

  `echo '\$var equals $var';` //yields `\$var equals $var`

- Use double quotes to echo (or print) the value of a variable
- Use single quotes to echo (or print) HTML

Using HTML Forms

- HTML forms
- How are they used?
  - Use the browser's window as a data entry screen
  - Collect information from the user
  - Pass it to the web server via http
  - Invoke a server-side script
  - Passes ***form data*** as input to the script

Using HTML Forms

- `<form>` tag has several attributes – two are required
- ACTION
  - `<form action="http://URL">` name of a program on the web server
    - URL specifies the location of the executable file on the web server
  - `<form action="mailto:mailrecipient">` sends an email
- METHOD
  - `<form method="POST" >` or `<form method="GET">`
    - **POST** when you have large amount of data being sent, encryption available, a two-step process
    - **GET** for small amounts, no security – all in one step

      `<form enctype=`
      - » multipart/form-data (default)
      - » text/plain (used only for mailto)

Using HTML Forms

- the <input> tag
  - Specifies an input field on a form
- type attribute – tells us what kind of control
  - **text**
  - **radio**
  - **checkbox**
  - **submit button**
  - **reset button**

- <form> examples
- Text Box

```
<input type="text" name="Name" size="20" maxlength="30">
```

- Radio Button(s)

```
<input type="radio" name="Gender" value="M" /> Male
<input type="radio" name="Gender" value="F" /> Female
```

- Check Box(es)

```
<input type="checkbox" name= "size" value="S"
  checked="checked" />Small
<input type="checkbox" name="size" value="M" />Medium
<input type="checkbox" name="size" value="L" />Large
<input type="checkbox" name="size" value="XL" />X-Large
```

- List Box

```
<select name="Grade" size="3">
      <option>A
      <option>B
      <option>C
      <option>D
      <option>F
</select>
```

- List Box via <select> tag
  - **Size** attribute
    - When absent: you get a "drop down list", first item selected by default
    - When present: indicates the number of items in the list
  - **Selected** attribute: specifies selected item
  - **Multiple** attribute: when "yes", can click > 1

Using Forms

```
<input type="submit" />
<input type="reset" />

<textarea name="comments" cols="40" rows="8">
```

- FORM demo
  - NOT from URL, just opened in browser
  - Using method = GET
  - Using action = MAILTO

- Sending FORM data to a PHP program

  Requires TWO files

  - An **HTML** page with a FORM and a SUBMIT button
  - A **PHP** program invoked when the FORM is submitted, specified in ACTION attribute
  - Must be specified via **URL** in the ACTION attribute

(In the HTML Form page)

```
<form method="post"
   action="http://localhost/handleform.php"
  enctype="multipart/form-data"
  onsubmit="window.alert('Form is being posted')">
```

Using Forms

- Values passed from the HTML page to the PHP program appear in a SYSTEM VARIABLE ARRAY called "$_REQUEST"
- Entries in the "$_REQUEST" array are referenced by their HTML "name= " attribute
- Note that the CHECKBOX input type comes through as an array (multiple values)

- Use the var_dump() method to see all variable information

- Use the `isset()` function
  - To determine whether a variable has been assigned a value
- Example:

```
<?php
    If (isset($_REQUEST['Gender'])) {
          $Gender = $_REQUEST['Gender'];
    } else {
          $Gender = NULL;
    }
</select>
```

- Validating FORM data
  - You don't want to let any bad data get into your database
  - You must assume that if users are allowed to enter bad data, they will
  - **`isset()`** will test FALSE for an empty string
  - **`empty()`** will test TRUE for an empty string
  1. Did they enter ANYTHING?
  2. Is it VALID for the variable type?
  3. Is it a valid VALUE for the field?

- Example

```
<?php
// Validate the name:
if (!empty($_REQUEST['name'])) {
    $name = $_REQUEST['name'];
} else {
    $name = NULL;
    echo '<p>You forgot to enter your name!</p>';
}
?>
```

- Tips on Validating FORM fields
  - `is_numeric()` tests if a field contains valid numbers
  - It is a good idea (courtesy) to inform your users whether or not form fields are REQUIRED or OPTIONAL

php Basics

- Handling ARRAYS
  - Two types:
    - Index Keys, like $_REQUEST[1]
    - String Keys, like $_REQUEST['Name']
  - Indexes begin at 0 (default)
  - Wrap array references with string keys in { } when using `echo()` or `print()` to avoid parse errors

```
<?php
        echo "{$_REQUEST['Name']}";
?>
```

php Basics

- Creating ARRAYS
  - Declare and initialize with `array()` function

```
<?php
   $cars = array();
   $cars[0] = 'Acura';
   $states = array('AL','AK','AR',…,…,'WY');
   $days = array('M'=>'Monday','T'=>'Tuesday');
   $months = array('Jan','Feb','Mar',… …);
?>
```

  - Load them one entry at a time or all at once

- Accessing ARRAYS using `foreach()` loop

```
<?php
   foreach ($array as $value) {
       // do something with $value
   }
?>
```

- This command loops through the array `$array` and with each iteration sets `$value` equal to the value of each successive entry

- Getting KEYS and VALUES

```
<?php
   foreach ($array as $key => $value) {
       // do something with $key and $value
   }
?>
```

The symbol => maps the key to the value

php Basics

- Setting initial KEY

```
<?php
   $states = array( 1 => 'AL','AK',…,…,'WY');
?>
```

- To fill an array with numeric values use the `range()` function

```
<?php
   $months = range(1, 12);
?>
```

(Demo = arraydemo.php, 1 through 4)

php Basics

- While Loop

```
<?php
   while (condition) {
       // do something
   }
?>
```

- Checks the condition FIRST
- If the condition is TRUE, it executes "something"

php Basics

- For Loop

```
<?php
   for ($i=1, $i <=10, $i++) {
       // do something
   }
?>
```

- Sets $i to 1, checks the condition ($i <= 10)
- Then if the condition is true, it will do something
- Then it will increment the counter, check the condition, etc.
- "Do While" versus "Do Until"
- Beware infinite loops

Combining PHP and MySQL

- Three steps
  - Connect to the database
  - Run the query
  - Parse query output

Combining PHP and MySQL

- Connecting to the database
  - "mysqli" is a class that represents the connection between a php program and a database
  - We use the `mysqli_connect()` function to connect
  - Syntax:

```
$dbc = mysqli_connect(
        hostname,username,pw,db_name)
```

  - `$dbc` is used as a variable in subsequent MySQL functions

Combining PHP and MySQL

- Define the four connection parameters as CONSTANTS – prohibits them from being changed (not very secure)
- Use the "or DIE" option on the call to the function

Combining PHP and MySQL

- Running a Simple Query
  - Uses the `mysqli_query()` function
  - Syntax:

    ```
    $r = mysqli_query($dbc,$q)
    ```

    where:

    `$dbc =` database connection placeholder

    `$q =` text string containing your SQL query

    `$r =` boolean flag indicating success/failure

Combining PHP and MySQL

- Handling Query Output
  - `msyqli_assoc` – associates the column name to the array index
  - `mysqli_num` – associates a number to the column array index
    - These are “constants”
  - Uses the `mysqli_fetch_array()` function
  - Returns ONE answer set ROW at a time
  - Typically embedded in a "while" loop
  - Syntax:

    ```
    $row = mysqli_fetch_array($r)
    ```

    where:

    `$row =` an array holding the contents of one row of the answer set

    `$r =` the placeholder variable for the query answer

Code Samples - SELECT

```php
<?php # simple_select.php

include ('header.html');

// Set the database access information as constants:
DEFINE ('DB_USER', 'root');
DEFINE ('DB_PASSWORD', '');
DEFINE ('DB_HOST', 'localhost');
DEFINE ('DB_NAME', 'northwinds');

// Make the connection:
$dbc = @mysqli_connect (DB_HOST, DB_USER, DB_PASSWORD, DB_NAME)
       OR die ('Could not connect to MySQL: ' . mysqli_connect_error() );  // Connect to the db.

$q = "SELECT FirstName, LastName, Country from nwemployees;";   // Define the query.

$r =  mysqli_query($dbc,$q);  // Run the query.

// Count the number of returned rows:
$num = mysqli_num_rows($r);

if ($num > 0) { // If it ran OK, display the records.

    // Print how many rows were returned:
echo "<p>This query returned $num rows.</p>\n"; }

    // Fetch and print all the records:
    while ($row = mysqli_fetch_array($r, MYSQLI_NUM)) {
        echo $row[0]." ".$row[1]." ".$row[2]."<br>";

    }
mysqli_close($dbc); // Close the database connection.

include ('footer.html');
?>
```

Code Samples - UPDATE

```php
<?php # update.php

include ('header.html');

// Set the database access information as constants:
DEFINE ('DB_USER', 'root');
DEFINE ('DB_PASSWORD', '');
DEFINE ('DB_HOST', 'localhost');
DEFINE ('DB_NAME', 'northwinds');

// Make the connection:
$dbc = @mysqli_connect (DB_HOST, DB_USER, DB_PASSWORD, DB_NAME)
       OR die ('Could not connect to MySQL: ' . mysqli_connect_error() );  // Connect to the db.

//$q = "SELECT FirstName, LastName, Country from nwemployees;"; // Define the query.

$q = "UPDATE nwemployees set FirstName = 'Alan' where EmployeeID = 3;"; // Define update query.

$r =  mysqli_query($dbc,$q);  // Run the UPDATE query.

$q = "SELECT FirstName, LastName, Country from nwemployees;";   // Define select query to show updates.

$r =  mysqli_query($dbc,$q);  // Run the SELECT query.

        // Fetch and print all the records:
    while ($row = mysqli_fetch_array($r, MYSQLI_NUM)) {

        echo $row[0]." ".$row[1]." ".$row[2]."<br>";

    }

mysqli_close($dbc); // Close the database connection.

include ('footer.html');
?>
```