Class: - B.Com, B.Com (Hons), BBA & BAJMC II Year

Subject: - Web Development Using PHP & MySQL

Unit	Contents
UNIT – I	**BASICS OF PHP:** Introduction to PHP, PHP features, installation of XAMPP/WAMP, Benefits of using PHP MYSQL, Server Client Environment, Web Browser, Web Server Installation & Configuration Files. OOPS with PHP, language basics, syntax, comments, variables, constants and data types, expressions and operators, flow control statements, looping structures, Arrays Including html code in PHP, Embedding PHP in web pages.
UNIT – II	**FUNCTIONS & STRINGS in PHP:** Defining a function, Calling a function, variable scope, function parameters, return values, User Defined Function, System Defined Function, Parameterized Function, Date & Time Function, Hash Function, Mail Function, predefined functions. Strings: Creating & accessing string, searching and replacing strings, encoding and escaping, comparing strings, formatting strings, regular expression.
UNIT – III	Data & File Handling: PHP Forms: $_GET, $_POST, $_REQUEST, S_FILES, $_SERVER, SGLOBALS, $ ENV, input/output controls, validation, Cookies and Sessions. File Handling: File and directory, open, close, read, write, append, delete, uploading and downloading files. File exists, File Size, Rename. Reading and display all/selected files present in a directory.
UNIT – IV	**MySQL an Overview:** Introduction, What is a Database, Understanding an RDBMS, Tables, Record & Fields, SQL Language. **Working with phpmyadmin:** Creating and using a database, Selecting a database, creating/dropping a table, loading data into a table, Retrieving information from a table, selecting all data, selecting particular rows, selecting particular columns, writing queries, sorting, date, calculations, working with NULL values, pattern matching, counting rows, using more than one tables, using table and column aliases.
UNIT – V	**MySQL DATABASES IN PHP:** Introduction, connecting to a MySQL database, querying the database, Retrieving and displaying the results, modifying data and deleting data through front end. Designing applications using PHP & MySQL.

Class: - B.Com, B.Com (Hons), BBA & BAJMC II Year Subject: Web Development Using PHP & MySQL

UNIT -1

Introduction to PHP & Features

PHP is a server scripting language, and a powerful tool for making dynamic and interactiveWeb pages.

PHP is a widely-used, free, and efficient alternative to competitors such as Microsoft's ASP.

What is PHP?

- PHP is an acronym for "PHP: Hypertext Preprocessor"
- PHP is a widely-used, open source scripting language
- PHP scripts are executed on the server
- PHP is free to download and use

What is a PHP File?

- PHP files can contain text, HTML, CSS, JavaScript, and PHP code
- PHP code are executed on the server, and the result is returned to the browseras plain HTML
- PHP files have extension ".php"

What Can PHP Do?

- PHP can generate dynamic page content
- PHP can create, open, read, write, delete, and close files on the server
- PHP can collect form data
- PHP can send and receive cookies
- PHP can add, delete, modify data in your database
- PHP can be used to control user-access
- PHP can encrypt data

With PHP you are not limited to output HTML. You can output images, PDF files, and even Flash movies. You can also output any text, such as XHTML and XML.

Why PHP?

- PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP supports a wide range of databases
- PHP is free. Download it from the official PHP resource: www.php.net
- PHP is easy to learn and runs efficiently on the server side

To start using PHP, you can:

- Find a web host with PHP and MySQL support
- Install a web server on your own PC, and then install PHP and MySQL

Use a Web Host with PHP Support

- If your server has activated support for PHP you do not need to do anything.
- Just create some .php files, place them in your web directory, and the server will

automatically parse them for you.

- You do not need to compile anything or install any extra tools.
- Because PHP is free, most web hosts offer PHP support.
- Set Up PHP on Your Own PC

However, if your server does not support PHP, you must:

- install a web server
- install PHP
- install a database, such as MySQL

PHP Scripts Syntax

- A PHP script can be placed anywhere in the document.
- A PHP script starts with **<?php** and ends with **?>**:
- PHP statements end with a semicolon (;)

Example

```
<?php

// PHP code goes
here ?>
```

```
<html>
<body>
<h1>My first PHP page</h1>
<?php
echo "Hello World!";
?>
</body>

</html>
```

Comments in PHP

A comment in PHP code is a line that is not read/executed as part of the program. Its only purpose is to be read by someone who is looking at the code.

Comments can be used to:

- Let others understand what you are doing
- Remind yourself of what you did - Most programmers have experienced coming back to their own work a year or two later and having to re-figure out what they did. Comments can remind you of what you were thinking when you wrote the code

Example

```
<html>
<body>
<?php
// This is a single-line comment
# This is also a single-line
comment /*
This is a multiple-lines comment
block that spans over multiple
```

```
lines
*/
// You can also use comments to leave out parts of a code line
$x = 5 /* + 15 */ + 5;
echo
$x; ?>
</body>
</html>
```

Variables

- Variables are "containers" for storing information.
- Creating (Declaring) PHP Variables
- In PHP, a variable starts with the $ sign, followed by the name of the variable:

Rules for PHP variables:

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

- A variable starts with the $ sign, followed by the name of the variable
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive ($age and $AGE are two different variables)

Output Variables

- The PHP echo statement is often used to output data to the screen.

PHP Variables Scope

- In PHP, variables can be declared anywhere in the script.
- The scope of a variable is the part of the script where the variable can be referenced/used.

PHP has three different variable scopes:

- local
- global
- static

Global and Local Scope

A variable declared **outside** a function has a GLOBAL SCOPE and can only be accessed outside a function:

A variable declared **within** a function has a LOCAL SCOPE and can only be accessedwithin that function:

The global Keyword

- The global keyword is used to access a global variable from within a function.
- To do this, use the global keyword before the variables (inside the function):

The static Keyword

Normally, when a function is completed/executed, all of its variables are deleted. However,

sometimes we want a local variable NOT to be deleted. We need it for a further job.

echo and print Statements

- In PHP there are two basic ways to get output: echo and print.
- In this tutorial we use echo (and print) in almost every example. So, this chapter contains a little more info about those two output statements.
- echo and print are more or less the same. They are both used to output data tothe screen.
- The differences are small: echo has no return value while print has a return value of 1 so it can be used in expressions. echo can take multiple parameters (although such usage is rare) while print can take one argument. echo is marginally faster than print.

Echo Statement

The echo statement can be used with or without parentheses: echo or echo().

Data Types

- Variables can store data of different types, and different data types can do different things.

PHP supports the following data types:

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- Array
- Object
- NULL
- Resource

PHP String

- A string is a sequence of characters, like "Hello world!".
- A string can be any text inside quotes. You can use single or double quotes:

PHP Integer

An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.

Rules for integers:

- An integer must have at least one digit
- An integer must not have a decimal point
- An integer can be either positive or negative
- Integers can be specified in three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with 0)
- In the following example $x is an integer. The PHP var_dump() function returns the data type and value:

PHP Float

- A float (floating point number) is a number with a decimal point or a number in exponential form.
- In the following example $x is a float. The PHP var_dump() function returns the

data type and value:

PHP Boolean

A Boolean represents two possible states: TRUE or FALSE.

```
$x = true;
$y = false;
```

Booleans are often used in conditional testing. You will learn more about conditional testing in a later chapter of this tutorial.

PHP Array

- An array stores multiple values in one single variable:
- An array is a special variable, which can hold more than one value at a time.
- An array can hold many values under a single name, and you can access the values by referring to an index number.

Create an Array in PHP

- In PHP, the array() function is used to create an array:

```
array();
```

In PHP, there are three types of arrays:

- **Indexed arrays** - Arrays with a numeric index
- **Associative arrays** - Arrays with named keys
- **Multidimensional arrays** - Arrays containing one or more arrays

PHP Operators

Operators are used to perform operations on variables and values.

PHP divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Increment/Decrement operators
- Logical operators
- String operators
- Array operators

PHP Arithmetic Operators

- The PHP arithmetic operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.

Operator	Name	Example	Result
+	Addition	$x + $y	Sum of $x and $y

-	Subtraction	$x - $y	Difference of $x and $y
*	Multiplication	$x * $y	Product of $x and $y
/	Division	$x / $y	Quotient of $x and $y
%	Modulus	$x % $y	Remainder of $x divided by $y
**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power (Introduced in PHP 5.6)

PHP Assignment Operators

- The PHP assignment operators are used with numeric values to write a valueto a variable.
- The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the assignment expression on the right.

Assignment	Same as...	Description
x = y	x = y	The left operand gets set to the value of the expression on the right
x += y	x = x + y	Addition
x -= y	x = x - y	Subtraction
x *= y	x = x * y	Multiplication
x /= y	x = x / y	Division
x %= y	x = x % y	Modulus

PHP Comparison Operators

- The PHP comparison operators are used to compare two values (numberor string):

Operator	Name	Example	Result
==	Equal	$x == $y	Returns true if $x is equal to $y

===	Identical	$x === $y	Returns true if $x is equal to $y, and they are of the same type
!=	Not equal	$x != $y	Returns true if $x is not equal to $y
<>	Not equal	$x <> $y	Returns true if $x is not equal to $y
!==	Not identical	$x !== $y	Returns true if $x is not equal to $y, or they are not of the same type
>	Greater than	$x > $y	Returns true if $x is greater than $y
<	Less than	$x < $y	Returns true if $x is less than $y
>=	Greater than or equal to	$x >= $y	Returns true if $x is greater than or equal to $y
<=	Less than or equal to	$x <= $y	Returns true if $x is less than or equal to $y

PHP Increment / Decrement Operators

- The PHP increment operators are used to increment a variable's value.
- The PHP decrement operators are used to decrement a variable's value.

Operator	Name	Description
++$x	Pre-increment	Increments $x by one, then returns $x
$x++	Post-increment	Returns $x, then increments $x by one
--$x	Pre-decrement	Decrements $x by one, then returns $x
$x--	Post-decrement	Returns $x, then decrements $x by one

PHP Logical Operators

- The PHP logical operators are used to combine conditional statements.

Operator	Name	Example	Result
And	And	$x and $y	True if both $x and $y are true
Or	Or	$x or $y	True if either $x or $y is true
Xor	Xor	$x xor $y	True if either $x or $y is true, but not both

&&	And	$x && $y	True if both $x and $y are true
||	Or	$x || $y	True if either $x or $y is true
!	Not	!$x	True if $x is not true

PHP String Operators

- PHP has two operators that are specially designed for strings.

Operator	Name	Example	Result
.	Concatenation	$txt1 . $txt2	Concatenation of $txt1 and $txt2
.=	Concatenation assignment	$txt1 .= $txt2	Appends $txt2 to $txt1

PHP Array Operators

- The PHP array operators are used to compare arrays.

Operator	Name	Example	Result
+	Union	$x + $y	Union of $x and $y

==	Equality	$x == $y	Returns true if $x and $y have the same key/valuepairs
===	Identity	$x === $y	Returns true if $x and $y have the same key/valuepairs in the same order and of the same types
!=	Inequality	$x != $y	Returns true if $x is not equal to $y
<>	Inequality	$x <> $y	Returns true if $x is not equal to $y
!==	Non-identity	$x !== $y	Returns true if $x is not identical to $y

1. PHP Flow Control Statements

- Sequential (discussed till now)
- Selection(if, if..else,switch ...etc)
- Iterative(Loops)

- Very often when you write code, you want to perform different actions for different conditions. You can use conditional statements in your code to do this.

In PHP we have the following conditional statements (selection)

- if statement - executes some code if one condition is true
- if...else statement - executes some code if a condition is true and another code if that condition is false
- if...elseif ..... else statement - executes different codes for more than two conditions
- switch statement - selects one of many blocks of code to be executed

The if Statement

- The if statement executes some code if one condition is true.

Syntax

if (*condition*) {
code to be executed if condition is true;}

The if...else Statement

The if..... else statement executes some code if a condition is true and another code if that condition is false.

Syntax

```
if (condition) {
    code to be executed if condition is true;
} else {
    code to be executed if condition is false;
}
```

The if... elseif ....else Statement

The if.... elseif... else statement executes different codes for more than two conditions.

Syntax

```
if (condition) {
    code to be executed if this condition is true;
} elseif (condition) {
    code to be executed if this condition is true;
} else {
    code to be executed if all conditions are false;
}
```

The switch Statement

- The switch statement is used to perform different actions based ondifferent conditions.

Use the switch statement to **select one of many blocks of code to be executed**.

Syntax

```
switch (n)
    { case
    label1:
        code to be executed if n=label1;
        break
    ; case
    label2:
        code to be executed if n=label2;
        break
    ; case
    label3:
        code to be executed if n=label3;
        break;
    ...
    default:
        code to be executed if n is different from all labels;
```

```
}
```

This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use break to prevent the code from running into the next case automatically. The default statement is used if no match is found.

PHP Loops

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal code-lines in a script, we can use loops to perform a task like this.

In PHP, we have the following looping statements:

- while- loops through a block of code as long as the specified condition is true
- do...while - loops through a block of code once, and then repeats the loop as long as the specified condition is true
- for- loops through a block of code a specified number of times
- foreach- loops through a block of code for each element in an array

The PHP while Loop

- The while loop executes a block of code as long as the specified condition is true.

Syntax

```
while (condition is true) {
    code to be executed;
}
```

The PHP do...while Loop

The do...while loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true.

Syntax

```
do {
    code to be
executed; }while
(condition is true);
```

for Loops

- PHP for loops execute a block of code a specified number of times.

The PHP for Loop

- The for loop is used when you know in advance how many times the scriptshould run.

Syntax

```
for (init counter; test counter; increment counter) {
    code to be executed;
}
```

Parameters:

- *init counter*: Initialize the loop counter value
- *test counter*: Evaluated for each loop iteration. If it evaluates to TRUE, theloop continues. If it evaluates to FALSE, the loop ends.
- *increment counter*: Increases the loop counter value

The PHP foreach Loop

- The foreach loop works only on arrays, and is used to loop through eachkey/value pair in an array.

Syntax

```
foreach ($array as $value)
{
    code to be executed;
}
```

- For every loop iteration, the value of the current array element is assigned to $value and the array pointer is moved by one, until it reaches the last array element.

Class: - B.Com, B.Com (Hons), BBA & BAJMC II Year Subject: Web Development Using PHP & MySQL

Unit-2

Functions

The real power of PHP comes from its functions; it has more than 1000 built-in functions. Functions are blocks of codes that perform specific task in php. It makes the codes reusable and shorthand. it can be categorized into two groups.

- Predefined functions
- User defined functions

Predefined Functions

Predefined function (plural predefined functions) (computing) **Any of a set of subroutines that perform standard mathematical functions included in a programming language**; either included in a program at compilation time, or called when a program is executed. Predefined functions are the inbuilt functions of php. These functions can be subdivided into multiple categories as stated below.

- string functions
- numeric functions
- date and time functions
- array functions
- directory functions

String Function

strtolower(); -> converts all characters of the string to lower case
strtoupper(); -> converts all characters of the string to upper case
ucfirst(); -> converts first letter to upper case
ucwords(); -> converts first letter of each word to upper case
strlen(); -> counts number of characters in a string and returns integer value
trim(); -> trims unnecessary spaces
ltrim(); -> trims unnecessary spaces from left
rtrim(); -> trims unnecessary spaces from right
sprintf("%s,%s,%s",$a,$b,$c); -> placeholder to keep the value
str_word_count() -> count the number of words
$count = str_word_count($x,1) -> returns array
strstr(); echo strstr($str,U,true);
stristr(); case insensitive strstr
str_replace() -> search and replace characters from the string
str_repeat() -> repeats the string
substr(int,int) -> prints a string from defined initial character number to defined last number
strpos() -> finds position of the string

Numeric Function

abs() -> returns positive value of a number
sqrt() -> returns square root of a number
round() -> rounds a floating number
floor() -> rounds a number down to a nearest integer
ceil() -> rounds a number up to a nearest integer
rand() -> generates a random integer
mt_rand() -> generates random number between defined inital and end number
pow() -> returns x raised to the power of y
pi() -> returns the value of pi
min() -> returns the lowest value from an array
max() -> returns the highest value from an array
fmod() -> returns the remainder from x/y {%}
bindec() -> converts a binary number to a decimal number
decbin() -> converts a decimal number to a binary number
deg2rad() -> converts a degree value to a radian value

Class: - B.Com, B.Com (Hons), BBA & BAJMC II Year Subject: Web Development Using PHP & MySQL

Array Functions

array() -> creates an array
sizeof() -> counts the number of values in an array
sort() -> sorts an indexed array in ascending order
in_array() -> checks if a specified value is in array
list() -> keep array values in variable
compact() -> keeps variable values in associative array
arsort() -> sorts an associative array in descending order according to the value
array_unique() -> removes duplicate values from an array
explode() -> converts string to array
implode() -> converts array to string
extract() -> converts array to variable
array_sum() -> returns the sum of values in an array
array_shift() -> removes the first element of an array and returns the first value from the array
array_pop() -> deletes the last element from an array
array_merge() -> merges multiple arrays
array_search() -> searches for a defined value in an array and returns the key for that value
array_reverse() -> returns an array in reverse order
array_keys() -> returns all the keys from an array

PHP Date/Time Functions

PHP date() Function: The PHP date() function converts timestamp to a more readable date and time format.

Syntax:

date(format, timestamp)

- The format parameter in the date() function specifies the format of returned date and time.
- The timestamp is an optional parameter, if it is not included then the current date and time will be used.

Formatting options available in date() function: The format parameter of the date() function is a string that can contain multiple characters allowing to generate the dates in various formats. Date-related formatting characters that are commonly used in the format string:

- d: Represents day of the month; two digits with leading zeros (01 or 31).
- D: Represents day of the week in the text as an abbreviation (Mon to Sun).
- m: Represents month in numbers with leading zeros (01 or 12).
- M: Represents month in text, abbreviated (Jan to Dec).
- y: Represents year in two digits (08 or 14).
- Y: Represents year in four digits (2008 or 2014).

The parts of the date can be separated by inserting other characters, like hyphens (-), dots (.), slashes (/), or spaces to add additional visual formatting.

PHP time() Function: The time() function is used to get the current time as a Unix timestamp (the number of seconds since the beginning of the Unix epoch: January 1, 1970, 00:00:00 GMT).
The following characters can be used to format the time string:

- h: Represents hour in 12-hour format with leading zeros (01 to 12).
- H: Represents hour in 24-hour format with leading zeros (00 to 23).
- i: Represents minutes with leading zeros (00 to 59).
- s: Represents seconds with leading zeros (00 to 59).
- a: Represents lowercase antemeridian and post meridian (am or pm).

- A: Represents uppercase antemeridian and post meridian (AM or PM)

checkdate()	Validates a Gregorian date
date_diff()	Returns the difference between two dates
date_format()	Returns a date formatted according to a specified format
date_time_set()	Sets the time
getdate()	Returns date/time information of a timestamp or the current local date/time

PHP User Defined Functions

- Besides the built-in PHP functions, we can create our own functions.
- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute immediately when a page loads.
- A function will be executed by a call to the function.

Defining a Function in PHP

A user-defined function declaration starts with the word function:

Syntax

```
function functionName() {
code to be executed;
}
```

- Function names are NOT case-sensitive.

In the example below, we create a function named "writeMsg()". The opening curly brace ({) indicates the beginning of the function code and the closing curly brace (}) indicates the end of the function. The function outputs "Hello world!". To call the function, just write its name:

Example

```
<html>
<body>
<?php
function
writeMsg()echo
"Hello world!";
}
writeMsg();
?>
</body>
```

```
</html>
```

PHP Function Arguments

- Information can be passed to functions through arguments. An argument is just like a variable.
- Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

PHP Default Argument Value

The following example shows how to use a default parameter. If we call the function setHeight() without arguments it takes the default value as argument:

Example

```
<html>
<body>
<?php
function setHeight($minheight = 50) {
   echo "The height is : $minheight
   <br>";
}
setHeight(3
50);
setHeight();
setHeight(1
35);
setHeight(8
0);
?>
</body>
</html>
```

PHP Functions - Returning values

To let a function return a value, use the return statement:

Unit-3

PHP Forms

PHP forms allow users to input data, which is processed and handled by the PHP server. PHP handles form submissions using **superglobals** like `$_GET`, `$_POST`, and `$_REQUEST`

HTML Form Basics

A basic HTML form uses the `<form>` tag. The `action` attribute specifies the script where the form data is sent, and the `method` attribute specifies how the data is sent.

Example:

```
html
Copy code
<form action="process_form.php" method="POST">
    <label for="name">Name:</label>
    <input type="text" id="name" name="name">

    <label for="email">Email:</label>
    <input type="email" id="email" name="email">

    <input type="submit" value="Submit">
</form>
```

- **`action`**: Specifies the target PHP script (e.g., `process_form.php`).
- **`method`**:
  - `GET`: Appends data to the URL.
  - `POST`: Sends data securely in the HTTP body.

PHP Form Processing

`$_GET`

- Retrieves data sent via the HTTP GET method.
- Appends data to the URL (e.g., `example.com/form.php?name=John`).
- Example:

```
php
Copy code
$name = $_GET['name']; // Access 'name' from URL query string
```

`$_POST`

- Retrieves data sent via the HTTP POST method.
- Data is sent in the request body, making it more secure than `GET`.
- Example:

```
php
```

```
Copy code
$email = $_POST['email']; // Access 'email' from form data
```

$_REQUEST

- Retrieves data from both `$_GET` and `$_POST` arrays, as well as `$_COOKIE`.
- Not recommended due to potential security concerns (ambiguity about the source).
- Example:

```
php
Copy code
$data = $_REQUEST['data'];
```

$_FILES

- Handles file uploads.
- Provides details such as file name, size, and type.
- Example:

```
php
Copy code
$fileName = $_FILES['uploadedFile']['name'];
move_uploaded_file($_FILES['uploadedFile']['tmp_name'], 'uploads/' . $fileName);
```

2. Server and Global Variables

$_SERVER

- Contains server and environment information (e.g., headers, paths, and script locations).
- Example:

```
php
Copy code
$scriptName = $_SERVER['SCRIPT_NAME']; // Current script name
```

$GLOBALS

- A superglobal array containing all global variables.
- Rarely needed; better practices involve using function arguments or class properties.

$_ENV

- Stores environment variables.
- Example:

```
php
Copy code
$envVar = $_ENV['MY_ENV_VAR']; // Access an environment variable
```

3. Input/Output Controls

Sanitizing and Validating Input

- Use PHP filters to validate and sanitize input:

```
php
Copy code
$email = filter_input(INPUT_POST, 'email', FILTER_SANITIZE_EMAIL);
if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
    echo "Valid email.";
} else {
    echo "Invalid email.";
}
```

Output Escaping

- Prevent Cross-Site Scripting (XSS) attacks:

```
php
Copy code
echo htmlspecialchars($userInput, ENT_QUOTES, 'UTF-8');
```

4. Validation Techniques

- **Client-side validation**: Use JavaScript for quick feedback.
- **Server-side validation**: Always validate on the server to ensure security.

5. Cookies

- Store small amounts of data on the user's browser.
- Example:

```
php
Copy code
setcookie("username", "John", time() + (86400 * 30), "/"); // Set a cookie for
30 days
if (isset($_COOKIE['username'])) {
    echo "Welcome " . $_COOKIE['username'];
}
```

6. Sessions

- Store user data across multiple pages on the server.
- Example:

```
php
Copy code
session_start(); // Start a session
$_SESSION['user'] = "John"; // Store session data
echo $_SESSION['user']; // Access session data
```

Class: - B.Com, B.Com (Hons), BBA & BAJMC II Year Subject: Web Development Using PHP & MySQL

File Handling in PHP

File handling in PHP allows you to create, open, read, write, append, and manage files and directories.

Opening and Closing Files

- **`fopen()`**: Opens a file.

```
php
Copy code
$file = fopen("example.txt", "r"); // Open file in read mode
fclose($file); // Close the file
```

- Modes for `fopen()`:
  - `"r"`: Read only.
  - `"w"`: Write only (truncates file or creates a new one).
  - `"a"`: Append (write at the end of file or create a new one).
  - `"x"`: Create a new file for writing (fails if file exists).

Reading Files

- **`fread()`**: Reads content from a file.

```
php
Copy code
$file = fopen("example.txt", "r");
$content = fread($file, filesize("example.txt"));
fclose($file);
echo $content;
```

- **`file_get_contents()`**: Reads the entire file into a string.

```
php
Copy code
$content = file_get_contents("example.txt");
echo $content;
```

Writing to Files

- **`fwrite()`**: Writes content to a file.

```
php
Copy code
$file = fopen("example.txt", "w");
fwrite($file, "Hello, World!");
fclose($file);
```

- **`file_put_contents()`**: Writes a string to a file.

```
php
Copy code
file_put_contents("example.txt", "Hello, PHP!");
```

Appending to Files

- Open the file in append mode (`"a"`) and use `fwrite()`:

```
php
Copy code
$file = fopen("example.txt", "a");
fwrite($file, " This is an additional line.");
fclose($file);
```

Deleting Files

- **`unlink()`**: Deletes a file.

```
php
Copy code
if (file_exists("example.txt")) {
    unlink("example.txt");
    echo "File deleted.";
}
```

2. Directory Handling

Creating a Directory

- **`mkdir()`**: Creates a directory.

```
php
Copy code
mkdir("my_directory");
```

Reading Files in a Directory

- **`scandir()`**: Retrieves an array of all files and directories.

```
php
Copy code
$files = scandir("my_directory");
foreach ($files as $file) {
    if ($file !== "." && $file !== "..") {
        echo $file . "<br>";
    }
}
```

Checking if a File Exists

- **`file_exists()`**: Checks if a file or directory exists.

```
php
Copy code
if (file_exists("example.txt")) {
    echo "File exists.";
} else {
```

```
        echo "File does not exist.";
    }
```

Getting File Size

- **`filesize()`**: Returns the size of a file in bytes.

```
php
Copy code
$size = filesize("example.txt");
echo "File size: $size bytes";
```

Renaming Files or Directories

- **`rename()`**: Renames a file or directory.

```
php
Copy code
rename("old_name.txt", "new_name.txt");
```

3. File Uploading

HTML Form

```
html
Copy code
<form action="upload.php" method="POST" enctype="multipart/form-data">
    <input type="file" name="fileToUpload">
    <input type="submit" value="Upload">
</form>
```

PHP Script (`upload.php`)

```
php
Copy code
if ($_FILES['fileToUpload']['error'] == 0) {
    $targetDir = "uploads/";
    $targetFile = $targetDir . basename($_FILES["fileToUpload"]["name"]);

    if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $targetFile)) {
        echo "File uploaded successfully.";
    } else {
        echo "File upload failed.";
    }
}
```

4. File Downloading

```
php
Copy code
$file = "example.txt";
if (file_exists($file)) {
    header('Content-Description: File Transfer');
    header('Content-Type: application/octet-stream');
```

```
    header('Content-Disposition: attachment; filename="' . basename($file) . '"');
    header('Content-Length: ' . filesize($file));
    readfile($file);
    exit;
}
```

5. Reading Selected Files

To read and display specific files based on user input:

```
php
Copy code
$selectedFile = "example.txt"; // Assume this is selected by the user
if (file_exists($selectedFile)) {
                        echo nl2br(file_get_contents($selectedFile));
```

Unit -4

Introduction

Sure the idea of dynamic web pages is cool, but you can only go far with what's built into PHP, like changing the page based on the day of the week. What you'd really like to do is make a web page unique for each visitor, and that's where databases come in.

We will begin this chapter assuming that the reader has absolutely no knowledge of MySQL or databases. First, we'll explain databases, then we'll create one the easy way

— using phpMyAdmin. Then we'll cover how to create databases and tables using SQL, and in the next chapter we'll show how all this can be done using PHP.

What are Databases?

Databases help to organize and track things. Databases allow you to use creativity to group things together in meaningful ways, and to present the same set of information in different ways to different audiences.

Database is simply an organized collection of data stored in a computer.

Databases are composed of one or more "tables". Tables are composed of parts called "rows" and "columns" similar to what you would see in a spreadsheet. The columns section of each table declares the characteristics of each table while each row contains unique data for each element in the table.

It may sound complicated but actually it is quite simple. Take the example below

Table: Cars

ID	VIN	Make	Model	Style	Year	Price
1	1328237824	Ford	Explorer	SUV	2005	5995
2	4797834923	Dodge	RAM	Pickup	2008	7200
3	2394923724	Mazda	6	Passenger	2010	9995
4	2342323634	Subaru	Outback	Passenger	2007	4500

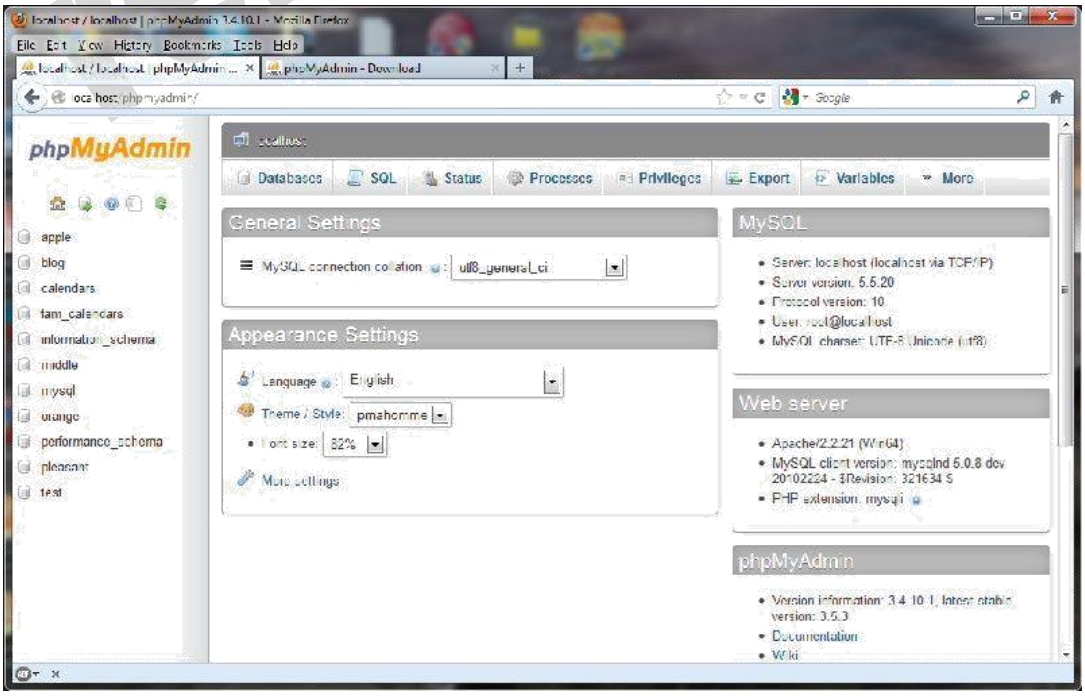
We can clearly see that the elements in this table has seven **columns** defined as ID, VIN, Make, Model, Style, Year, and Price. The table has four **rows** that describe four different cars—a Ford Explorer, Dodge RAM, Mazda 6, and a Subaru Outback.

Here is a quick review of what we have learned.

- **Tables** are just a collection of things that you want to keep track of.
- **Tables** consist of rows and columns.
- **Columns** hold the different attributes of each element in that table. Rows in a table hold different instances uniquely defined by the table's columns. **Databases** are a collection of tables.

Getting Started with phpMyAdmin

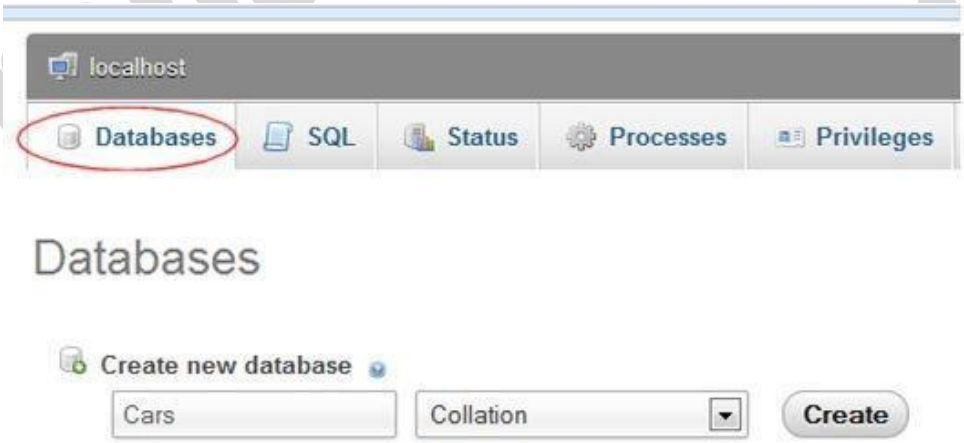
just enter this address into your browser or click on this link: http://localhost/phpmyadmin/ Clicking that link should take you to a page that is similar to this:



What is phpMyAdmin?

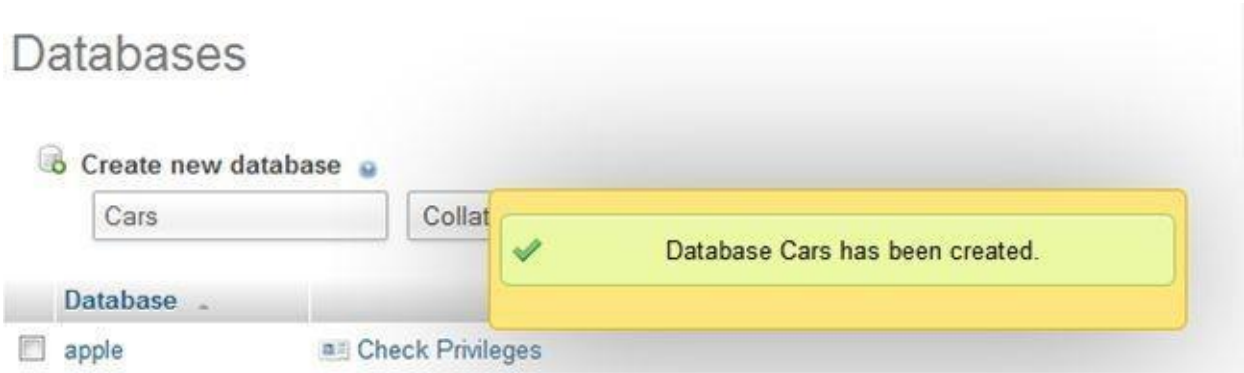
phpMyAdmin is a free software tool—that just happens to be written in PHP itself— that is intended to handle many common administration tasks of MySQL using a browser. phpMyAdmin supports a wide range of operations with MySQL. The most frequently used operations are supported by the user interface (managing databases, tables, fields, relations, indexes, users, permissions, etc), and you still have the ability to directly execute a SQL statement if you prefer.

Using phpMyAdmin to create a database



First navigate such that you have phpMyAdmin on the screen. Click on the link that says Databases:

In the box that says Create new database, type the word ‘Cars’, then click on the Create button. If it worked properly, you should see a yellow confirmation box appear on the screen briefly, as below:



Introduction to SQL

This is equivalent to issuing the SQL command

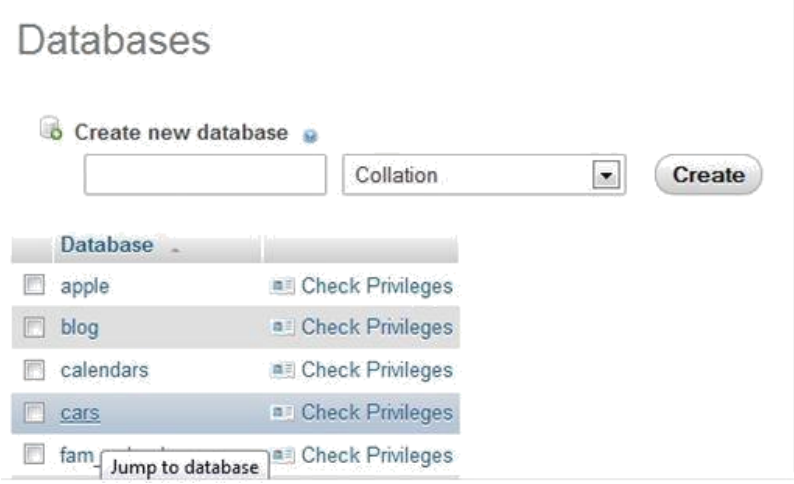
```
CREATE DATABASE Cars;
```

and, in fact, phpMyAdmin actually executed that **exact** SQL command in the background for you when you clicked on the button. In other words, you can think of phpMyAdmin as a tool that builds SQL commands for you.

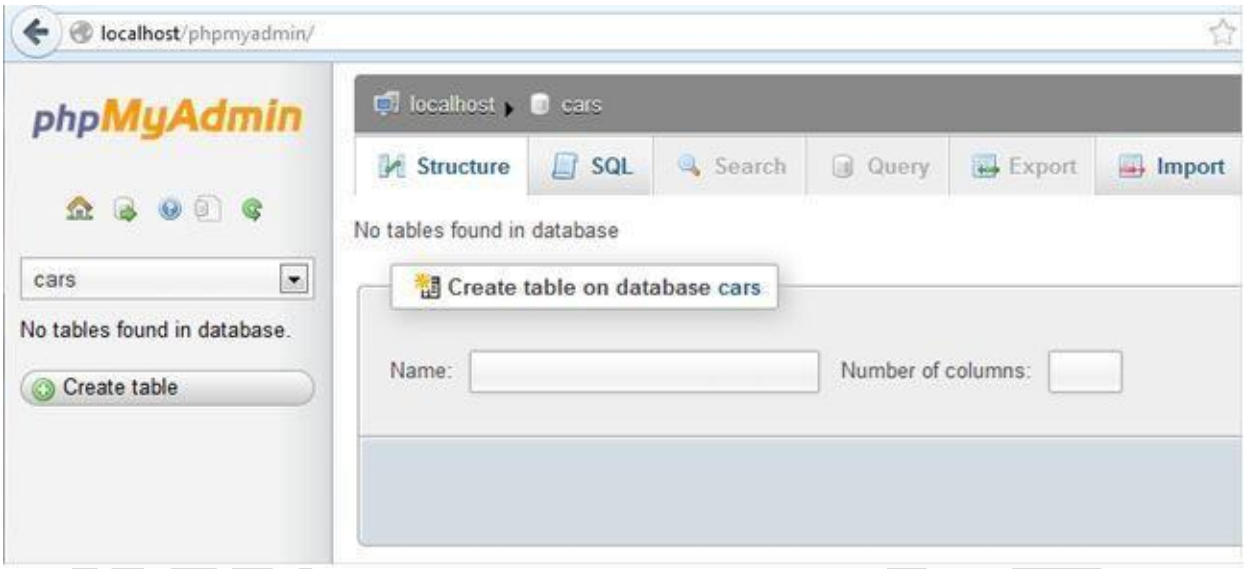
SQL (pronounced "sequel") or Structured Query Language) is a language all its own. SQL is a special-purpose programming language designed for managing data in relational database management systems, such as MySQL. SQL can be used to create databases, create tables, and insert, update, and delete data into tables.

Using phpMyAdmin to create a Table in a database

Now that the database is created, we would like to use it. Find the cars database in the list of databases, then click on the database name.



phpMyAdmin will provide a page similar to:



This is the equivalent to the SQL command:

```
USE cars;
```

This tells the MySQL database that you are going to work in the database *cars* until you say otherwise.

You have just created the database for our fictional used car lot. We will develop this database more as we go along.

Defining our first table

So far, you have created your database, and figured out the general structure of PHPMyAdmin. Now you will need to put a table inside of the database you have created. In the case of our cars database, we will need to define the table to describe the cars and trucks that Sam has for sale on his used car lot.

Before creating your table, think about what you are going to put into the table and what are the various attributes that might distinguish one row (car) from another.

What defines an automobile?

I can think of a number of properties or attributes that distinguish one car from another on a used car lot.

- Vehicle ID Number (VIN)
- Year
- Make
- Model
- Trim
- Exterior color
- Interior color
- Asking Price
- Purchase Price
- Mileage
- Transmission
- Purchase Date
- Sale Date
- Sale Price

That should be enough to at least let us get started. Now we have to figure out what kind of data we are going to put in these categories.

Datatypes

For learning purposes, there are really only three types of data you will need to use.

They are:

1. Numbers
2. Characters
3. Dates

Numbers

Numbers, as the name probably gives away, are any kind of numeric information. Will you need to use any kind of decimals for the data that you are going to store? In that case, you will need to use the datatype decimal or float. If not, you can use the datatype int (short for integer) or bigint (a big integer—which takes up more space, but can handle bigger numbers).

Characters

The character type in MySQL is the data type you use to store Strings. Characters are used to store the representation of a letter, word, or series of words. For example the letter A and the phrase 'Hello World' would both be of a character type. MySQL calls this a VARCHAR, short for variable characters It is variable because you only set the maximum number of characters that the field can hold, and if you put in a value with fewer characters, the shorter value will be stored. Other databases, such as Microsoft SQL Server, offer the CHAR datatype, which will fill in any unused characters with spaces. Why anyone would want that I can't imagine, so for simplicity we'll stick to VARCHAR for now.

Use the datatype varchar(n) to define a column that you would like to represent with a character. Substitute the n in varchar(n) with the maximum amount of letters a column

in your table can have (up to 255). Spaces, tabs, and newlines in a paragraph all count as characters.

Dates

Dates are a way to store dates in the database. Do you just want to store the date and not the time? Use the datatype date. Do you want to store the time and not the date? Use the datatype time. Want to store the date and the time? Use the datatype datetime.

Let's look back at our characteristics of cars to decide what kind of datatype they should be.

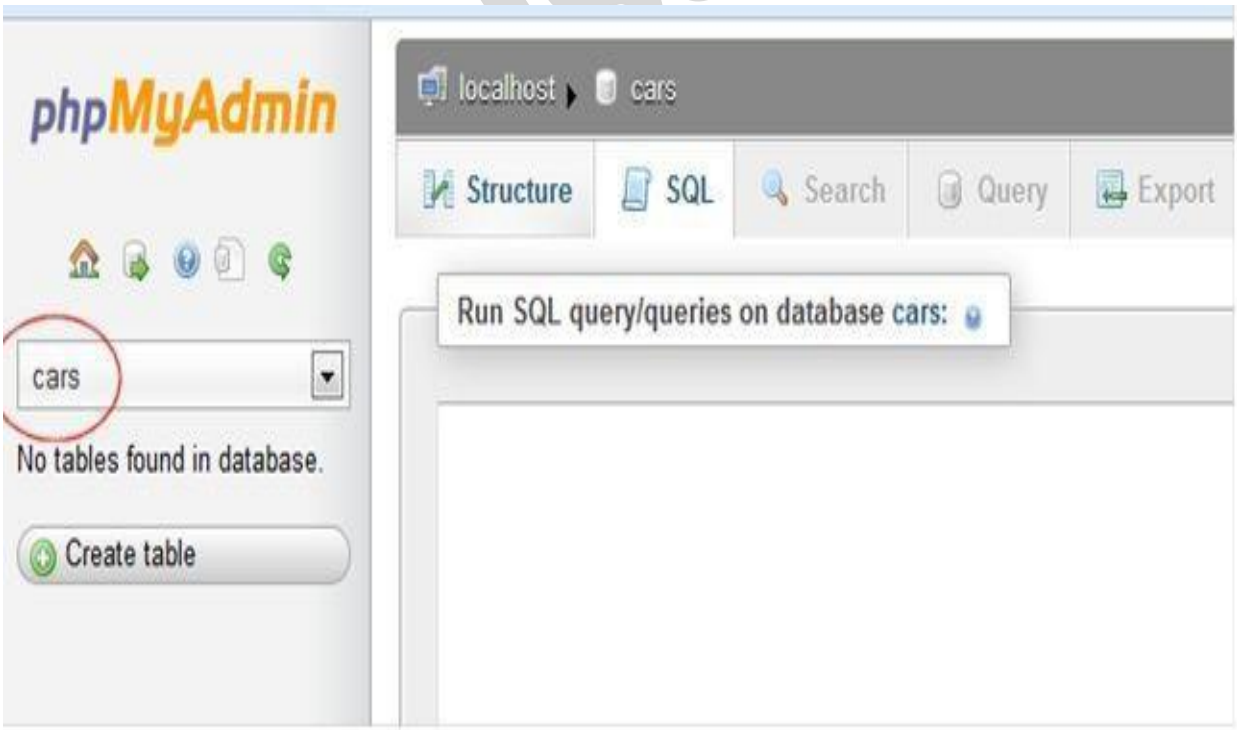
- Vehicle ID Number (VIN) – All over-the-road-vehicles have a 17-character VIN, which does not include the letters I (i), O (o), or Q (q) (to avoid confusion with numerals 1 and 0). Varchar(17)
- Year - Consists of numbers without a decimal point. Int
- Make – Consists of text. Varchar(25)
- Model – Consists of text and the occasional number. Varchar(25)
- Trim – Consists of text. Varchar(25)
- Exterior color – Consists of text. Varchar(25)
- Interior color – Consists of text. Varchar(25)
- Asking Price - Consists of numbers with decimal point. Decimal
- Purchase Price - Consists of numbers with a decimal point. Decimal
- Mileage - Consists of numbers without a decimal point. Int
- Transmission – Consists of text. Varchar(25) Purchase (Acquisition) Date - Date
- Sale Date - Date

- Sale Price - Consists of numbers without a decimal point. Int

That about sums up the table that we need to create to track our cars. Since the VIN is the only truly unique element in the list, we will make this the "Primary Key".

Defining a column as a primary key means that the column will only be able to have unique values (i.e. nothing can repeat itself). In the case of this specific table, it means that you can't enter two cars with the same VIN into the database, because we have just told mySQL that this isn't allowed. Some examples of this in everyday life are license plate numbers, credit card numbers, and social security numbers. All of these numbers are supposed to unique for each person. The same concept applies to tables in databases. Whenever possible, it is good practice to make sure that the table you are creating contains some form of primary key to give something to uniquely identify a row.

How do I make a table with this information? Great question. Although we created the *database* using the phpMyAdmin wizard, from now on we're just going to use SQL.



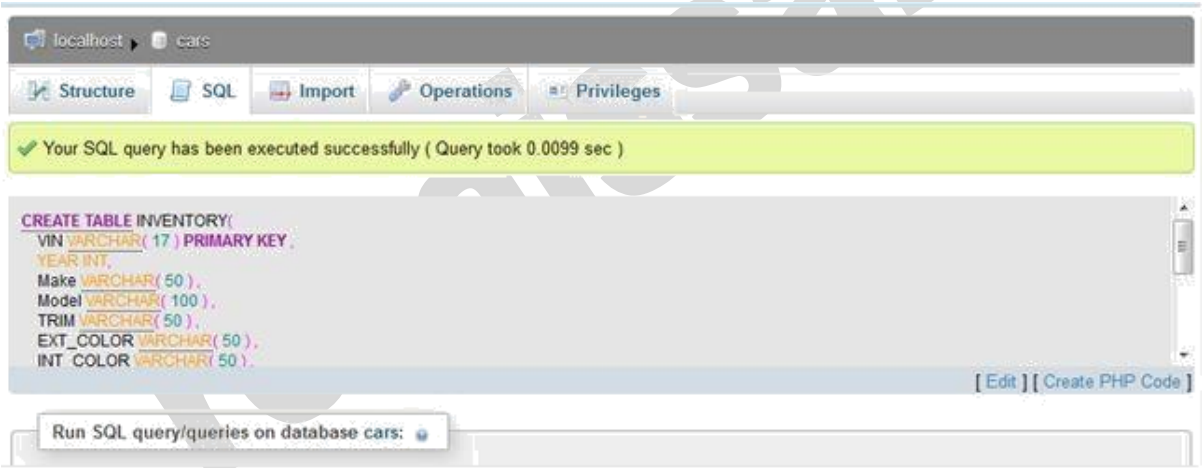
Class: - B.Com, B.Com (Hons), BBA & BAJMC II Year Subject: Web Development Using PHP & MySQL

In your window with phpMyAdmin, make sure that the cars table is selected (see it circled in red below), then click on the SQL tab to bring up the command box. Make sure that you see localhost -> cars above the box. If you do not, just click on the cars link on the right side and then the SQL tab to get yourself there.

Type the following command into the box and click go.

CREATE TABLE INVENTORY (VIN varchar(17) PRIMARY KEY, YEAR INT, Make varchar(50), Model varchar(100), TRIM varchar(50), EXT_COLOR varchar (50), INT_COLOR varchar (50), ASKING PRICE DECIMAL (10,2), SALE PRICE DECIMAL (10,2), PURCHASE_PRICE DECIMAL (10,2), MILEAGE int, TRANSMISSION varchar (50), PURCHASE_DATE DATE, SALE_DATE DATE)

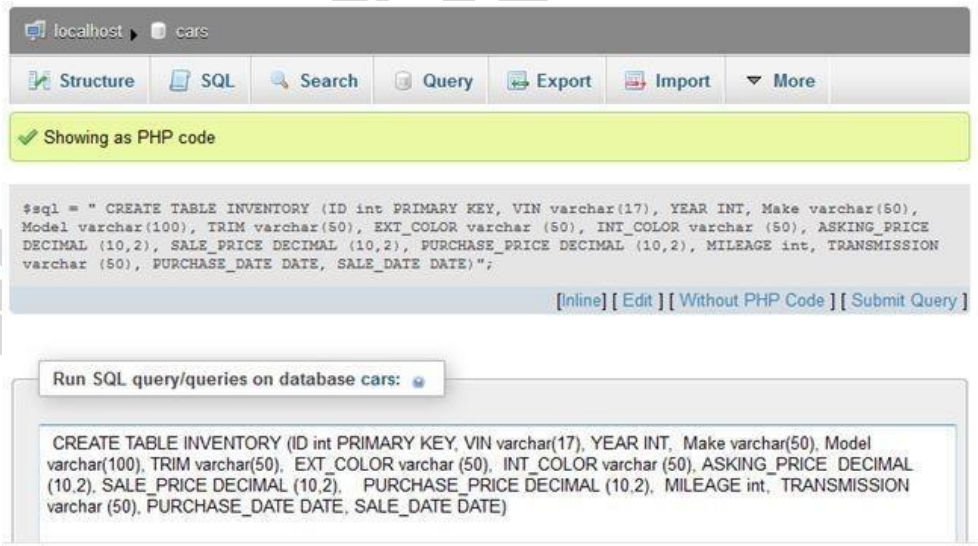
Congratulations! You have created the INVENTORY table.



Here's an incredibly useful tip: Click the link "**Create PHP Code**" located on the right side of the screen and what you'll get back is:

$sql = "CREATE TABLE INVENTORY (VIN varchar(17) PRIMARY KEY, YEAR INT, Make varchar(50), Model varchar(100), TRIM varchar(50), EXT_COLOR varchar (50), INT_COLOR varchar (50), ASKING PRICE DECIMAL (10,2), SALE PRICE DECIMAL (10,2), PURCHASE_PRICE DECIMAL (10,2), MILEAGE int, TRANSMISSION varchar (50), PURCHASE_DATE DATE, SALE_DATE DATE)";

The reason there is such a link is because anything you can do in mySQL using a SQL command, you can tell PHP to do for you in code. This represents a valid line of PHP code in which the variable **$sql** is assigned a string value to hold the SQL statement. Of course, there is more that would need to be done beyond this single line of code, but don't worry—we will cover this shortly.



Exercise: Create a Table

Create a table using a SQL statement, then delete the table and create it again using phpmyAdmin. Which is easier?

Working with SQL Statements

INSERT Statements

Now that you have a table created, the next logical step is to put some data into our table. In the world of SQL, this is accomplished with the INSERT command.

The syntax for inserting into a table is as follows:

INSERT INTO {Name of table} (Column Names) VALUES (Column Values);

Click on the SQL tab again, type the following command (if you can), and press enter.

```
INSERT INTO `cars`.`inventory` (`VIN`, `YEAR`, `Make`,
`Model`, `TRIM`, `EXT_COLOR`, `INT_COLOR`,
`ASKING_PRICE`, `SALE_PRICE`, `PURCHASE_PRICE`,
`MILEAGE`, `TRANSMISSION`, `PURCHASE_DATE`,
`SALE_DATE`) VALUES (' 5FNYF4H91CB054036', '2012',
'Honda', 'Pilot', 'Touring', 'White Diamond Pearl',
'Leather', '37807', NULL, '34250', '7076', 'Automatic',
'2012-11-08', NULL);
```

Obviously, writing SQL isn't *conceptually* difficult… but it is tedious and prone to error, especially as the statement gets longer. This statement:

```
INSERT INTO `cars`.`inventory` ( `YEAR`, `Make`,
`Model`, `ASKING_PRICE`) VALUES ( '2012', 'Honda',
'Pilot', '37807');
```

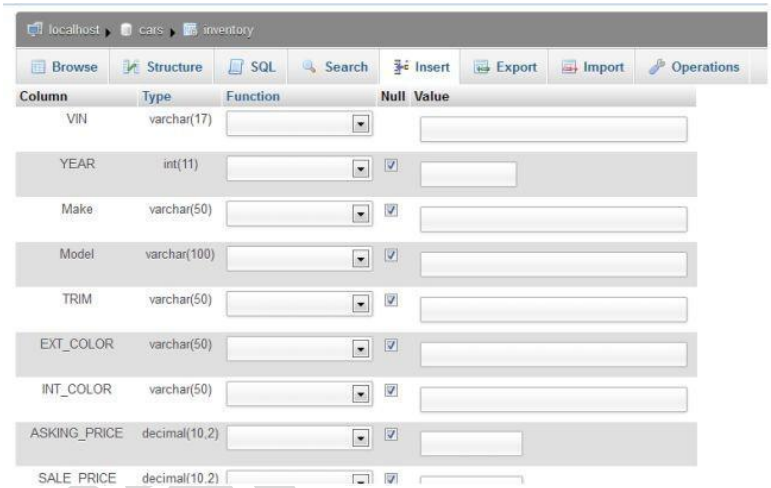
is pretty easy to follow, but this next one is a bit tougher:

```
INSERT INTO `cars`.`inventory` (`VIN`, `YEAR`, `Make`,
`Model`, `TRIM`, `EXT_COLOR`, `INT_COLOR`,
`ASKING_PRICE`, `SALE_PRICE`, `PURCHASE_PRICE`,
`MILEAGE`, `TRANSMISSION`, `PURCHASE_DATE`,
`SALE_DATE`) VALUES (' 5FNYF4H91CB054036', '2012',
'Honda', 'Pilot', 'Touring', 'White Diamond Pearl',
'Leather', '37807', NULL, '34250', '7076', 'Automatic',
'2012-11-08', NULL);
```

The only difference is the number of fields. The syntax is the same, but the challenge becomes making sure that there is a one-to-one relationship for each column name and value, and that they are in the right order—the column names and their respective values, that is.

As you can see, writing an INSERT statement is easy to goof up. We all do it. Luckily, phpMyAdmin makes it easy to generate perfect SQL statements. Simply click on the table, then click the **Insert** button and enter values into the boxes, as shown:

Class: - B.Com, B.Com (Hons), BBA & BAJMC II Year Subject: Web Development Using PHP & MySQL



Once you click the **Go** button, phpMyAdmin will create a SQL statement for you and insert the record, and even offer to convert it into a line of PHP code for you.

Here's a trick used by the professionals: once you have one line of SQL that works, it's pretty easy to copy and paste it and tweak the values for the next car. Go ahead and enter some more values until you get 5 or 6 cars entered into your table. Here's another one:

```
INSERT INTO `cars`.`inventory` (`VIN`, `YEAR`, `Make`,
`Model`, `TRIM`, `EXT_COLOR`, `INT_COLOR`,
`ASKING_PRICE`, `SALE_PRICE`, `PURCHASE_PRICE`,
`MILEAGE`, `TRANSMISSION`, `PURCHASE_DATE`,
`SALE_DATE`) VALUES ('  4T4BF3EK5AR031954',
'2010', 'Toyota', 'Camry', 'LE', ' Magnetic Gray Metallic',
'Tan cloth', '16977', NULL, '14250', ' 32673', '6 Speed
Automatic', '2012-11-08', NULL);
```

Don't worry if you mess up. MySQL will warn you, and prevent you from running incorrect commands. You don't need to enter 10 or 20 cars; the sample code includes a script that does that for you. Just do it enough times that you get it.

SELECT Statements

The syntax of SQL is pretty straight forward, at least syntactically. We have used it thus far to create a database, create a table within that database, and insert data into the table.

Class: - B.Com, B.Com (Hons), BBA & BAJMC II Year Subject: Web Development Using PHP & MySQL

There are just a few basic transactions left for us to master: reading data, updating data, and deleting data. Some people refer to this with the cheery acronym CRUD, for Create, Read, Update, and Delete.

Reading data is accomplished using the SELECT statement. The SELECT statement selects a value or group of values from a table and returns those value(s) to the user. Here's an easy way to remember it: The SELECT statement allows you to be selective. Clever, eh?

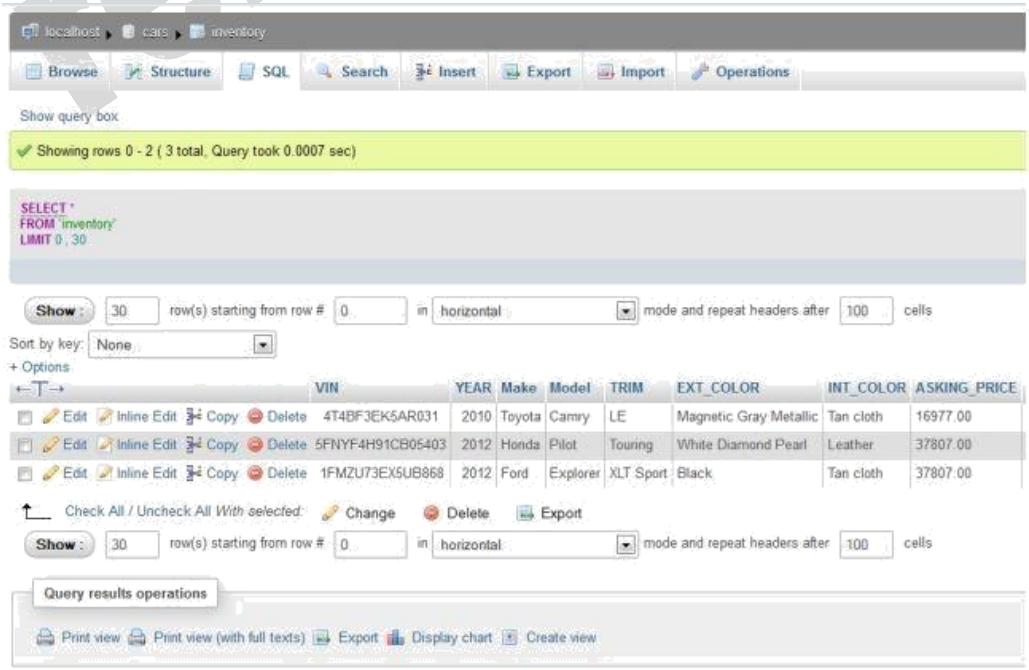
The syntax for selecting data from a table is as follows:

SELECT (Column Names) FROM (Table Name);

Let's start out with a simple SELECT statement. In phpMyAdmin, click on the cars icon on the left side and then click on the SQL tab at the top of the page. Type in the following command and press Go.

SELECT * FROM inventory;

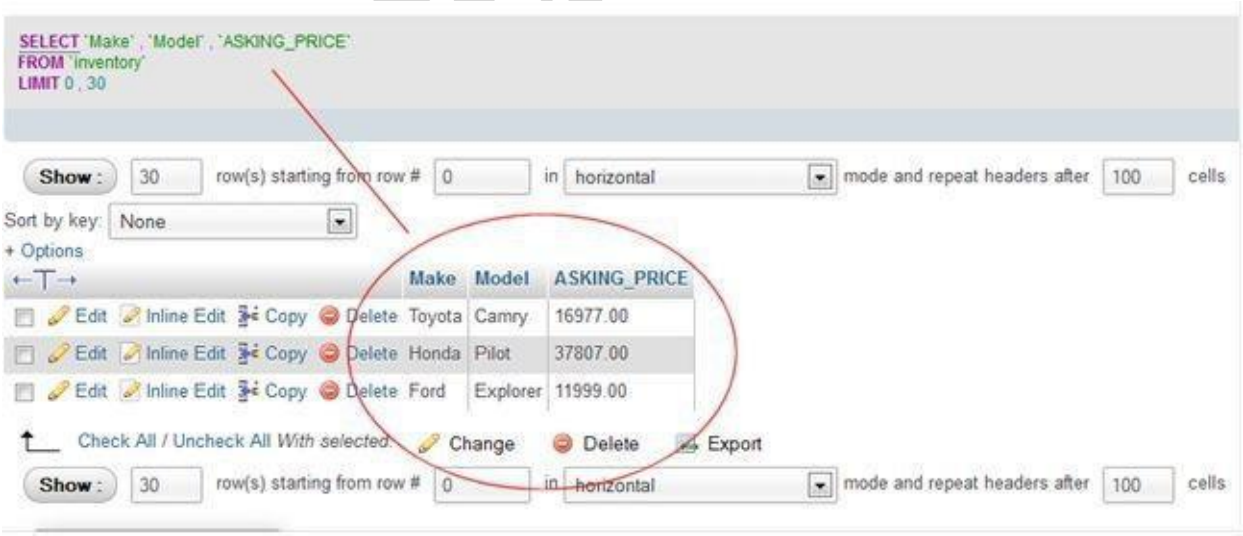
In general, the asterisk character (*) in computer lingo is called a wildcard and basically means "everything", so the result of the command above should return all rows and columns of the inventory table, and look similar to:



If you typed out this statement correctly, you should see the *entire* contents of your table 'inventory'. To select only *certain* columns of a table, type out all of the columns you want to see in that table separated by a comma. Type in the following command and press Go.

```
SELECT `Make`, `Model`, `ASKING_PRICE`FROM `inventory`
```

You should see something like this:



Note that I added the red circle and line to show you where to look. The mySQL database only returned the columns you specified using the SELECT statement.

WHERE Statements

So far, you have learned how to get **all** the rows and columns from a table, and how to get **selected columns** from a table, but what about **selected rows**?

This is where the **WHERE** statement comes into play. The WHERE statement gives a specific set of criteria to the MySQL database so that the results are much more controlled and relevant to what you want. For example, say that you want to select all the Ford Explorers that are in the inventory, or all the Toyotas under $15,000. The WHERE clause makes this possible.

```
SELECT `Model`, `TRIM`, `EXT_COLOR` FROM inventory
WHERE Make = 'Ford'
```

The results should be every automobile made by Ford in the database. If you wanted just Ford Explorers, you would need to have WHERE Make='Ford' AND Model = 'Explorer.

Of course, if you were looking to buy a car, you would only be interested in those cars that haven't already been sold, so the following query might be better suited:

```
SELECT `Make`, `Mode`, `EXT_COLOR` FROM inventory
WHERE YEAR > = 2010 AND `SALE_DATE` IS Null
```

NULL is a special word meaning that the field does not contain a value, and for some reason you can't say **= NULL**, you have to say **IS NULL**. I'm sure there is a reason for this, but it doesn't really matter. It is what it is.

Comparison Operators

Theze are many different comparison operators in addition to = and IS.

Operation	Symbol
Equal To	=
Not Equal To	!=
Greater Than	>
Less Than	<
Greater Than Or Equal To	>=
Less Than Or Equal To	<=

Remember to surround a string with quotations or parentheses every time you wish to use them in SQL statements. They will not work otherwise. Also, the WHERE command always goes **after** the SELECT statement in MySQL.

To find all of the automobiles with a year that is a 2010 or newer, it is fairly obvious that we need to use the **Greater Than Or Equal To** operator defined above. Type the following command into your compiler and press Go.

```
SELECT `Make`, `Mode`, `EXT_COLOR` FROM inventory
WHERE YEAR > = 2010 AND `SALE_DATE` IS Null
```

ORDER BY

The ORDER BY statement is probably one of the easiest and handiest commands in SQL. You can attach it at the end of any SELECT statement to put the results in the order of the column that you specify.

```
SELECT * FROM inventory ORDER BY YEAR DESC
```

The above statement should display the automobiles in order of the column 'Year' with the newest cars at the top. This is because the modifier DESC, or descending, is placed at the end of the command.

```
SELECT * FROM inventory WHERE YEAR > = 2006
ORDER BY YEAR ASC
```

The above statement should display the automobiles in order of the column 'Year' with the oldest cars at the top. This is because the modifier ASC, or ascending, is placed at the end of the command.

The ORDER BY modifier can also be used with a WHERE statement such as:

```
SELECT * FROM inventory WHERE YEAR > = 2006
ORDER BY YEAR ASC
```

Just remember that the **WHERE** command always goes **before** the **ORDER BY** command. If you mix them up, you will get an error.

To limit how many results you receive in an ORDER BY statement, use the limit clause after you write 'asc' or 'desc', such as

SELECT * FROM inventory ORDER BY YEAR DESC limit 10;

The number after limit determines how many results are returned.

UPDATE Statements

To update existing records in a database, you use the UPDATE statement. This would be useful, for example, when a car in the inventory goes on sale with a lower asking price.

The syntax for an update statement is

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

To change the asking price for a car in our database, you can use a statement such as:

```
UPDATE `cars`.`inventory` SET `ASKING_PRICE` =
'36999' WHERE `inventory`.`VIN` =
'5FNYF4H91CB054036';
```

DELETE Statements

To delete records from a database you use the **DELETE** statement, specifying the table name and a WHERE clause that specifies which records to delete.

```
DELETE FROM table_name

WHERE some_column=some_value
```

For example, to delete the Caravan cars from the inventory you could use a command similar to

```
DELETE from inventory WHERE `Model` = 'Caravan'
```

If you wanted to delete everything from a database table, you could skip the WHERE clause and use our friend the wildcard with a statement like

Delete * from inventory

Class: - B.Com, B.Com (Hons), BBA & BAJMC II Year Subject: Web Development Using PHP & MySQL

Unit -5

Using mySQL and PHP Together

In the previous chapter, we learned all the basics of using a database, in our case mySQL. All the SQL statements that we learned so far would likely work with other database systems, in this chapter, we're going to use PHP and mySQL together. This is where it really starts to get good.

Code Listing: createdb.php

```
1. <?php
2. /**
3. * Joy of PHP sample code
4. * Demonstrates how to create a database, create a table, and insert records.
5. */
6.
7. $mysqli = new mysqli('localhost', 'root', 'mypassword' );
8.
9. if (!$mysqli) {
10. die('Could not connect: ' . mysqli_error($mysqli));
11. }
12. echo 'Connected successfully to mySQL. <BR>';
13.
14.
15. /* Create table doesn't return a resultset */
16. if ($mysqli->query("CREATE DATABASE Cars") === TRUE) {
```

```
17. echo “<p>Database Cars created</P>”;

18. }

19. else

20. {

21. echo “Error creating Cars database: ” . mysqli_error($mysqli).”<br>”;

22. }

23. //select a database to work with

24. $mysqli->select_db(“Cars”);

25. Echo (“Selected the Cars database”);

26.

27. $query = ” CREATE TABLE INVENTORY

28. ( VIN varchar(17) PRIMARY KEY, YEAR INT, Make varchar(50), Model varchar(100),

29. TRIM varchar(50), EXT_COLOR varchar (50), INT_COLOR varchar (50), ASKING_PRICE DECIMAL (10,2),

30. SALE_PRICE DECIMAL (10,2), PURCHASE_PRICE DECIMAL (10,2), MILEAGE
int, TRANSMISSION varchar (50), PURCHASE_DATE DATE, SALE_DATE DATE)”;

31. //echo “<p>**********</p>”;

32. //echo $query ;

33. //echo “<p>**********</p>”;

34. if ($mysqli->query($query) === TRUE)

35. {

36. echo “Database table ‘INVENTORY’ created</P>”;

37. }

38. else

39. {

40. echo “<p>Error: </p>” . mysql_error();
```

Class: - B.Com, B.Com (Hons), BBA & BAJMC II Year Subject: Web Development Using PHP & MySQL

41. }

42. // Dates are stored in 'YYYY-MM-DD' format

43. $query = "INSERT INTO `cars`.`inventory`

44. (`VIN`, `YEAR`, `Make`, `Model`, `TRIM`, `EXT_COLOR`, `INT_COLOR`, `ASKING_PRICE`, `SALE_PRICE`, `PURCHASE_PRICE`, `MILEAGE`, `TRANSMISSION`, `PURCHASE_DATE`, `SALE_DATE`)

45. VALUES

46. ('5FNYF4H91CB054036', '2012', 'Honda', 'Pilot', 'Touring', 'White Diamond Pearl', 'Leather', '37807', NULL, '34250', '7076', 'Automatic', '2012-11-08', NULL);";

47.

48.

49. if ($mysqli->query($query) === TRUE) {

50. echo " Honda Pilot inserted into inventory table. ";

51. }

52. else

53. {

54. echo " Error inserting Honda Pilot: " . mysqli_error($mysqli);

55. echo " ******** ";**

56. echo $query ;

57. echo " ******** ";**

58. }

59.

60. // Insert a Dodge Durango

61.

62. $query = "INSERT INTO `cars`.`inventory` (`VIN`, `YEAR`, `Make`, `Model`, `TRIM`, `EXT_COLOR`, `INT_COLOR`, `ASKING_PRICE`, `SALE_PRICE`, `PURCHASE_PRICE`, `MILEAGE`, `TRANSMISSION`, `PURCHASE_DATE`, `SALE_DATE`)

```
63. VALUES

64. ('LAKSDFJ234LASKRF2', '2009', 'Dodge', 'Durango', 'SLT', 'Silver', 'Black', '2700', NULL, '2000',
'144000', '4WD Automatic', '2012-12-05', NULL);";

65.


66. If ($mysqli->query($query) === TRUE) {

67. echo "<p>Dodge Durango inserted into inventory table.</p>";

68. }

69. else

70. {

71. echo "<p>Error Inserting Dodge: </p>" . mysqli_error($mysqli);

72. echo "<p>**********</p>";

73. echo $query ;

74. echo "<p>**********</p>";

75. }


76. $mysqli->close();


77. ?>
```


Code Explained: createdb.php

Next I'll walk you through the code, line by line. Please take the time to follow along with me, as this is the **only way** to really get it. Yes, every line *does* matter.

```
1. <?php
```

line 1 is the start tag for PHP, and it tells the PHP interpreter that what follows is code, not HTML.

2. /**

3. * Joy of PHP sample code

4. * Demonstrates how to create a database, create a table, and insert records.

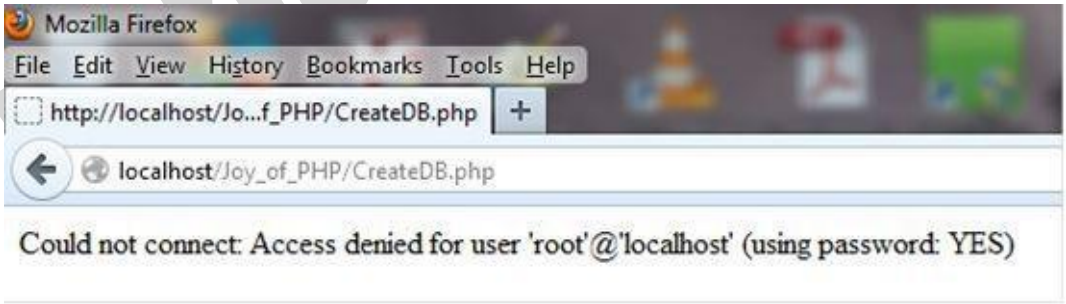
5. */

6.

lines 2 - 5 are comments. Comments are good, so put lots of comments in your code.

7. $mysqli = new mysqli('localhost', 'root', 'mypassword');

line 7 creates a variable called $con (for connection) and sets it equal to a built-in function for connecting to mySQL. You need to supply the hostname, username, and password for your mySQL server. If you do not have the correct username and password, you will see this:



9. if (!$mysqli) {

line 9 is the start of an **if** statement, saying basically "if you are **not** connected". The exclamation point is the not operator. The point of this line is to test to see if line 7 succeeded.

10. die('Could not connect: ' . mysqli_error($mysqli));

line 10 is what to do if the connection failed. '**die**' is a command that stops further code execution and prints out the text that follows. If I had been the one who invented PHP, I might have named that command 'stop' rather than 'die', but it does make the point.

```
11. }
12. echo 'Connected successfully to mySQL. <BR>';
```

line 12 prints out "Connected successfully to mySQL". This is the first line you see in the browser.

```
15. /* Create table doesn't return a resultset */
16. if ($mysqli->query("CREATE DATABASE Cars") === TRUE) {
17. echo "<p>Database Cars created</P>";
18. }
19. else
20. {
21. echo "Error creating Cars database: ". mysqli_error($mysqli)."<br>";
22. }
```

Line 15 is a comment that explains the function of the next line.

Line 17 prints to the browser if the SQL statement in line 15 ran without error.

Line 21 prints error information to the browser if the SQL statement in line 15 did not run successfully.

```
23. //select a database to work with
```

line 23 is a comment. Comments are good.

```
24. $mysqli->select_db("Cars");
```

line 24 creates a variable called $selected which uses a built-in function for selecting a mySQL database, using the connection created in line 7.

```
25. Echo ("Selected the Cars database");
```

line 25 prints "Selected the Cars database" to the browser.

```
27. $query = " CREATE TABLE INVENTORY

28. ( VIN varchar(17) PRIMARY KEY, YEAR INT, Make varchar(50), Model varchar(100),

29. TRIM varchar(50), EXT_COLOR varchar (50), INT_COLOR varchar (50), ASKING_PRICE
DECIMAL (10,2),

30. SALE_PRICE DECIMAL (10,2), PURCHASE_PRICE DECIMAL (10,2), MILEAGE
int, TRANSMISSION varchar (50), PURCHASE_DATE DATE, SALE_DATE DATE)";
```

lines 27 - 30 creates a variable called $query which holds an SQL statement. Recall that phpMyAdmin created this line of code for us. Good thing too, as it is an easy one to goof up.

```
31. //echo "<p>**********</p>";

32. //echo $query ;

33. //echo "<p>**********</p>";
```

lines 31 - 33 are comments *now*, but previously they were part of the script that printed out the value of the variable **$query**. I had this in there to help me figure out why it didn't work at first, and I leave it in there as an example of what to do when as script doesn't do quite what you thought it would. I then copied the output of line 32 to the clipboard and pasted it into phpMyAdmin for syntax advice.

```
34. if ($mysqli->query($query) === TRUE)
```

line 34 executes a SQL statement "query($query)" then tests for the result of the SQL statement held in the variable $mysqli.

```
35. {
```

Class: - B.Com, B.Com (Hons), BBA & BAJMC II Year Subject: Web Development Using PHP & MySQL

```
36. echo "Database table 'INVENTORY' created</P>";

37. }
```

line 36 prints the message "Database table 'INVENTORY' created" if line 34 is a success.

```
38. else
39. {
40. echo "<p>Error: </p>". mysqli_error($mysqli));
41. }
```

line 40 prints the message "Error:" and the mySQL error if line 34 fails. Hopefully the value returned by **mysql_error()** will tell you something helpful about why it failed. Sometimes it actually does.

```
42. // Dates are stored in MySQL as 'YYYY-MM-DD' format
```

line 42 is a comment to remind me (and you) to format dates the way mySQL expects them

```
43. $query = "INSERT INTO `cars`.`inventory`
44. (`VIN`, `YEAR`, `Make`, `Model`, `TRIM`, `EXT_COLOR`, `INT_COLOR`,
`ASKING_PRICE`, `SALE_PRICE`, `PURCHASE_PRICE`, `MILEAGE`, `TRANSMISSION`,
`PURCHASE_DATE`, `SALE_DATE`)
45. VALUES
46. ('5FNYF4H91CB054036', '2012', 'Honda', 'Pilot', 'Touring', 'White Diamond Pearl', 'Leather',
'37807', NULL, '34250', '7076', 'Automatic', '2012-11-08', NULL);";
```

lines 43 - 46 changes the value of $query to a new SQL statement, this time an INSERT.

```
49. if ($mysqli->query($query) === TRUE) {
```

line 49 tests for the execution of the SQL statement held in the variable $query

```
50. echo "<p>Honda Pilot inserted into inventory table. </p>";
```

line 50 prints the message " Honda Pilot inserted into inventory table " if line 49 is a success. The tags put the message on its own line.

51. }

52. else

53. {

54. echo " Error inserting Honda Pilot: " . mysql_error();

55. echo " ******** ";**

56. echo $query ;

57. echo " ******** ";**

58. }

lines 54 - 57 print a message if line 49 fails.

```
60.  // Insert a Dodge Durango
61.
62.  $query = "INSERT INTO `cars`.`inventory` (`VIN`,
`YEAR`, `Make`, `Model`, `TRIM`, `EXT_COLOR`,
`INT_COLOR`, `ASKING_PRICE`, `SALE_PRICE`,
`PURCHASE_PRICE`, `MILEAGE`, `TRANSMISSION`,
`PURCHASE_DATE`, `SALE_DATE`)
63.VALUES
64.('LAKSDFJ234LASKRF2', '2009', 'Dodge', 'Durango',
'SLT', 'Silver', 'Black', '2700', NULL, '2000', '144000',
'4WD Automatic', '2012-12-05', NULL);";
65.
```

Class: - B.Com, B.Com (Hons), BBA & BAJMC II Year Subject: Web Development Using PHP & MySQL

```
60. // Insert a Dodge Durango

61.

62. $query = "INSERT INTO `cars`.`inventory` (`VIN`, `YEAR`, `Make`, `Model`, `TRIM`, `EXT_COLOR`, `INT_COLOR`, `ASKING_PRICE`, `SALE_PRICE`, `PURCHASE_PRICE`, `MILEAGE`, `TRANSMISSION`, `PURCHASE_DATE`, `SALE_DATE`)

63. VALUES

64. ('LAKSDFJ234LASKRF2', '2009', 'Dodge', 'Durango', 'SLT', 'Silver', 'Black', '2700', NULL, '2000', '144000', '4WD Automatic', '2012-12-05', NULL);";

65.


66. If ($mysqli->query($query) === TRUE) {

67. echo "<p>Dodge Durango inserted into inventory table.</p>";

68. }

69. else

70. {

71. echo "<p>Error Inserting Dodge: </p>" . mysql_error();

72. echo "<p>**********</p>";

73. echo $query ;
```

```
74. echo "<p>**********</p>";

75. }

76.
```

lines 60 -76 does the same thing as 43 - 58, except for a different car.

```
78. $mysqli->close();

79. ?>
```

line 78 closes the connection to mySQL.

line 79 is the end tag for PHP, and any text that followed would be treated as HTML, rather than code.

Hey, where's the HTML?

The astute reader might have noticed that this script didn't appear inside the usual pattern of <HTML><Body> <html code here> <php code here> </Body></HTML>.

Yet it worked. How come? I discovered this quite by accident, actually. It's not a function of PHP but apparently some browsers will fill in the HTML framework for you if you "forget" to do so, which I did one time. Try it yourself. It works. Is this a best practice? No, I can't imagine that it is. But while you are learning it *does* let you focus on the PHP code.

Creating forms to Display, Add, Edit, and Delete data

Introduction

So far we've learned how to use SQL to create databases, add records, edit records, delete records, and select records. Then we learned how to use PHP to perform those same operations.

Next we'll get even more awesome. We'll learn how to use HTML forms along with PHP to create the SQL statements that perform the operation.

Forms that Add Data to a Database

A Basic Form

Let's start with a simple example that is easy to follow. Here's a simple, four-field form:

Obviously, it doesn't have all the attributes of a car that we have previously identified, and it's not very pretty to look at, but it *is* simple, and it will illustrate the point without any extra junk to get in the way of your understanding of the concept.

HTML Code

The code to produce such a form follows

<HTML>

Class: - B.Com, B.Com (Hons), BBA & BAJMC II Year Subject: Web Development Using PHP & MySQL

```
<head>

<title>Joy of PHP</title>

</head>
<body>

<h1>Sam's Used Cars

</h1>

<form>

VIN: <input name="VIN" type="text" /><br />

<br />

Make: <input name="Make" type="text" /><br />

<br />

Model: <input name="Model" type="text" /><br />

<br />

Price: <input name="Asking_Price" type="text"
/><br /> <br />

<input name="Submit1" type="submit" value="submit"
/><br />  </form>

</body>
</html>
```

So far what we have is just HTML, and in fact the form won't actually **do anything** if you press the submit button…yet.

Form Action

To make the form actually **do something**, we need to modify the <form> tag. Change the line of code above so that instead of saying **<form>** it says **<form action="SubmitCar.php" method="post">**

This tells the browser that when the form is submitted by pressing the submit button, it should pass this form to the PHP script entitled 'SubmitCar.php' and use the 'Post' method to do so.

Forms can be submitted either using **method='post'** or **method='get'**. There's really no good reason to use 'get' when submitting a form so to keep things simple, we'll just use 'post' whenever we submit a form.

We'll use get later in the book for a different purpose, though.

PHP Code

Here's what we are going to accomplish. We want the script referenced by the form to get the values from the form, produce a SQL **INSERT** statement using those values, write the SQL statement to the browser so we can see it, execute the SQL statement that we just created, and finally, let us know if it worked.

If all goes well, the script should output something similar to this:

```
INSERT INTO Inventory (VIN, Make, Model, ASKING_PRICE)

VALUES ('9T4BF3EKXBR153775', 'Ford', 'Fiesta', 800)
```

Connected successfully to mySQL

Selected the Cars database.

You have successfully entered Ford Fiesta into the database.

Here's the code for the SubmitCar.php file, which is also available in the sample code. Again, you don't have to study it here because I will walk you through it next. For now, just give it a quick look over.

```
1. <html>
2. <head>
3.     <title>Car Saved</title>
4. </head>
5. <body bgcolor="#FFFFFF" text="#000000" >
6.
7. <?php
8. // Capture the values posted to this php program from the text
fields in the form
9.
10.$VIN =   $_POST['VIN'] ;
11.$Make =  $_POST['Make'] ;
12.$Model =  $_POST['Model'] ;
13.$Price =  $_POST['Asking_Price'] ;
14.
15.//Build a SQL Query using the values from above
16.
17.$query = "INSERT INTO Inventory
18.  (VIN, Make, Model, ASKING_PRICE)
19.   VALUES (
20.   '$VIN',
21.   '$Make',
22.   '$Model',
23.    $Price
24.    )";
25.
26.// Print the query to the browser so you can see it
27.echo ($query. "<br>");
28.
```

```
29.$mysqli = new mysqli('localhost', 'root', 'password', 'cars' );
30./* check connection */
31.if (mysqli_connect_errno()) {
32.    printf("Connect failed: %s\n", mysqli_connect_error());
33.    exit();
34.}
35.
36.  echo 'Connected successfully to mySQL. <BR>';
37.
38.//select a database to work with
39.$mysqli->select_db("Cars");
40.   Echo ("Selected the Cars database. <br>");
41.
```

```
42./* Try to insert the new car into the database */
43.if ($result = $mysqli->query($query)) {
44.    echo "<p>You have successfully entered $Make $Model into
the database.</P>";
45.}
46.else
47.{
48.    echo "Error entering $VIN into database: " .
mysql_error()."<br>";
49.}
50.$mysqli->close();
51.?>
52.</body>
53.</html>
```

```
1. <html>
```

Line 1 is the opening <html> (which is closed on line 53).

```
2. <head>
3.    <title>Car Saved</title>
4. </head>
```

Lines 2 – 4 constitute the Head tag, while line 3 sets the page title.

```
5.<body bgcolor="#FFFFFF" text="#000000" >
```

Line 5 opens the body tag (which is closed on line 52). Note that we used the optional parameter to set the background and text colors.

Line 7 is the opening <php> tag, to signify that the text that follows is code rather than HTML.

```
8.  // Capture the values posted to this php program from
the text fields in the form
```

Line 8 is a comment. You can never have too many comments in your code. Get in the habit early of over-commenting your code. I have never heard anyone complain that the code they were trying to figure out had "too many" comments distracting from the elegance of the logic.

```
10.  $VIN =   $_POST['VIN'] ;
11.  $Make =  $_POST['Make'] ;
12.  $Model =  $_POST['Model'] ;
13.  $Price =  $_POST['Asking_Price'] ;
```

Lines 10 – 13 get the values that were on the form and assign them to variables in PHP. $_REQUEST is a special variable that is used to collect data after submitting HTML forms. You follow it with the name of the field on the HTML form that you want to retrieve.

A number of readers of the first edition of this book have commented that you should **never** trust the information that users give you, even in a corporate application like this one,where the users are generally trusted. So a safer way to acheive what we did above would be to use the PHP function called **mysql_real_escape_string** to strip out anything dangerous that users might try to enter. For instance, $Make = mysql_real_escape_string($_POST[‘Make’]);

The mysql_real_escape_string() function escapes special characters in a string for use in an SQL statement

```
15.  //Build a SQL Query using the values from above
```

Line 15 is a comment. Comments are good.

```
17.$query = "INSERT INTO Inventory
18.  (VIN, Make, Model, ASKING_PRICE)
19.   VALUES (
20.   '$VIN',
21.   '$Make',
22.   '$Model',
23.    $Price
24.    )";
```

Lines 17 – 24 build a SQL INSERT command. It could have been all on one line, but it is easier to read this way. Notice that the variables $VIN, $Make, $Model, and $Price are put into the formula as they are. Later, when the code is actually executed, PHP will substitute the variable names with their actual values.

```
// Print the query to the browser so you can see it
```

Line 26 is a comment. Comments are good.

```
27.echo ($query. "<br>");
```

Line 27 writes the SQL statement out to the browser, on its own line. The .”
” after the $query adds a
 to the end of the line. That’s what puts it on its own line. Line 27 was not required for the function to work. It is there so you can see how PHP translated the variables into their values when producing the SQL statement, which in turn is stored in the variable $query.

To add two stings together in PHP use the . character. In line 27 we have added two strings together-- $query and “
”, then used the echo command to write it to the browser.

Authors Note: If *I* was designing PHP, I would use the & character to join two strings. That would make it consistent with other languages, but it is what it is.

```
29.  $mysqli = new mysqli('localhost', 'root', 'password',
'cars' );
```

Line 29 makes a connection to the mySQL database by passing the name of the server ('localhost'), username ('root'), password ('password'), and initial database ('cars'). Note your password will likely be different.

```
30. /* check connection */
```

Line 30 is a comment, using the alternate syntax for denoting a comment.

```
31.  if (mysqli_connect_errno()) {
32.        printf("Connect failed: %s\n",
mysqli_connect_error());
33.     exit();
34.    }
```

Lines 31 – 34 test to see if the connection made with line 29 worked or not. If not, it prints an error message then stops further code execution (line 33 – exit). exit() is an alternative to the command **die**.

```
36.  echo 'Connected successfully to mySQL. <BR>';
```

Line 36 prints to the browser the message 'Connected successfully to mySQL'. This line would not execute if line 33 was called. Since we made it this far, we can conclude that we did in fact connect.

```
38.  //select a database to work with
```

Line 38 is a comment that explains the purpose of the next line.

```
39.  $mysqli->select_db("Cars");
40.   Echo ("Selected the Cars database. <br>");
```

Line 39 selects the 'cars' database, and **line 40** prints this fact.

```
42./* Try to insert the new car into the database */
```

Line 42 is a comment. You see a theme here, right? The more comments you add, the easier it will be to figure out your code when you come back later to look at it.

```
42. /* Try to insert the new car into the database */
43. if ($result = $mysqli->query($query)) {
44.     echo "<p>You have successfully entered $Make
$Model into the database.</P>";
```

Line 43 is the grand finale. Here we actually **execute the SQL statement** against the cars database. **Line 43** is the start of an if statement and **line 44** prints a success message while **line 48** prints a failure message.

Note: Line 48 really should read use 'mysqli_error($mysql)' not mysql_error(). This is corrected in the sample code. As an astute reader of the first edition pointed out, you can't mix **mysql** and **myslqi** in the same script– they are not the same. In any case, the mysql extension has been deprecated in favor of the mysqli extension.

```
50. $mysqli->close();
```

Line 50 closes the connection to the mySQL database. This is not strictly required, as the page will still work if you don't do it, but apparently it's a good idea because if you don't do it, eventually the server will develop problems and ultimately require a reboot.

```
51. ?>
```

Line 51 closes the PHP tag that was opened on **line 7**, signaling that the lines that follow are html not PHP code.

```
52.</body>
53.</html>
```

Lines 52 and 53 close the body tag and the HTML tags, respectively.

Wow, we made it through the whole script. If you are still with me, you have a good future in PHP development! Stay with it!

A Brief Time Out…include files and SQL Injection

Include Files

You may recall from the earlier section on *Includes* the notion of reusing code by including the contents of one file in another. This is a good time to revisit this important topic.

So far we've made two different PHP files—the **first** one to create a database and table, and the **second** one in the section above to insert data into the database using a web form. As you can guess from the section headings coming up later in this chapter, we're about to make **even more** scripts that will allow us to edit and delete data as well.

Each of these scripts will have a something in common—code that connects to the mySQL database, and in each case that code will be exactly the same. So far, we've been developing on our own computer, so the host name has been 'LocalHost'. Imagine yourself, sometime in the near future, having written a dozen or more scripts into the future, and suddenly you decide to move your application to another computer—one accessible from the Internet. The host name will not be the same. Nor, most likely, will the username and password be the same. What if your password got out and you needed to change it?

Without my helpful intervention **right here**, you would be facing the prospect of changing dozens of .php files—searching for the line that reads something like…

```
$mysqli = new mysqli('localhost', 'root', 'password', 'cars' );
```

…and changing it to reflect the new host name, username, or password. Uck— there would be *no joy* in that task at all. From now on, we're going move the part of the code that connects to the database to a separate file, and all our new scripts from this point forward will simply refer to that code using an **include** statement. If any of the values change, we will only have to change it in one place… the file that all the others point to.

Just imagine the joy of changing one line of code and seeing that change propagate across dozens of pages. That's what I'm talking about. The include feature is one of the most helpful and important features of PHP, in my humble opinion.

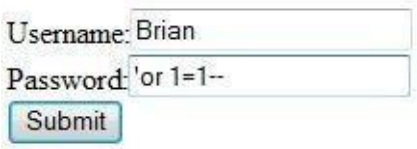
We'll use the line …

```
include 'db.php';
```

… to tell PHP to insert the contents of the db.php file into the current script. Be sure to use include files whenever you can, as the extra few minutes it takes to move some code out to a separate file is more than paid back when that code has to change.

SQL Injection

In general, it is **not** a good idea to take *whatever* the user enters into a form and pass that directly to a SQL script as we did in the above example. If the user were malicious (and skilled) they might enter SQL code into one of your forms and this could have a big impact on what the script actually does. For example, imagine a basic username/password form and the user entering 'or 1=1– into the Password field, as shown:



Now the statement that is executed in the database is the following:

SELECT * FROM Users WHERE Username= 'Brian' and Password= "or 1 = 1—'

Because 1=1 is **always** true, this query will return all users. (Note that the last quotation is commented out.) So, in the script above, sqlsrv_has_rows is true, and all the username password rows will be returned.

SQL injection is possible here because user input is concatenated with the executed SQL code. One way to prevent against this is to strip out any slashes or quote marks from the the user' input. The following code snippet demostrates this:

```
// To protect against SQL injection
$make = stripslashes($myusername);
$model = stripslashes($mypassword);
$make = mysql_real_escape_string($myusername);
$model = mysql_real_escape_string($mypassword);
```

At the risk of stating the obvious, the **stripslashes** command removes any slashes the users and **mysql_real_escape** command removes the quote characters.

An even better way to reduce the chance for SQL injection is to use prepared statements, but this is a topic that is beyond the scope of this beginner's book. If you want to learn more, here's a good place to start —> http://www.dreamincode.net/forums/topic/54239-introduction-to-mysqli-and-prepared-statements/

Forms that Display Summary Data

One of the first things we'll want to do for Sam's Used Cars is to display a list of all the cars that meet the selected criteria. At first, our criteria will be to select **all** the cars, but later on **you** can modify the query to return only certain cars simply by modifying the SELECT statement in the code.

The way this will work is that we will execute a SQL Select statement to retrieve the cars that match the criteria, then loop through all the rows. We'll put each row of data into a nicely formatted table.

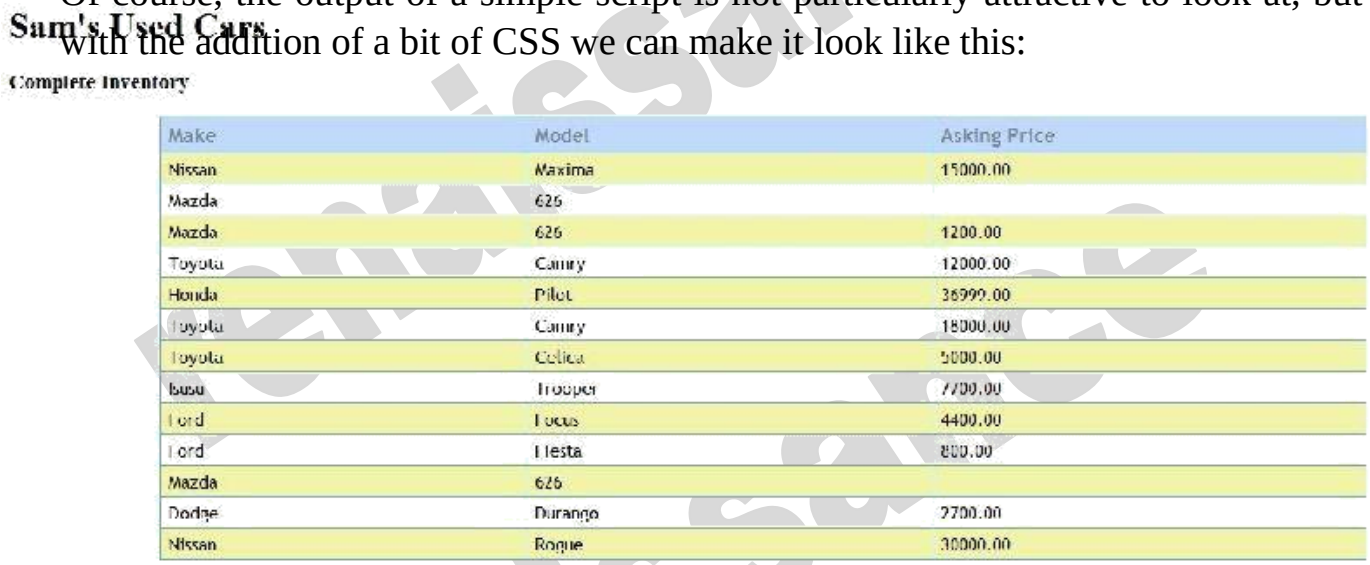
Sam's Used Cars

Complete Inventory

Make	Model	Asking Price
Nissan	Maxima	15000.00
Mazda	626	
Mazda	626	1200.00
Toyota	Camry	12000.00
Honda	Pilot	36999.00
Toyota	Camry	18000.00
Toyota	Celica	5000.00
Isusu	Trooper	7700.00

The source code can be found as viewcars.php. If all goes well the page should look like this:

Of course, the output of a simple script is not particularly attractive to look at, but with the addition of a bit of CSS we can make it look like this:



Sam's Used Cars

Complete Inventory

Make	Model	Asking Price
Nissan	Maxima	15000.00
Mazda	626	
Mazda	626	1200.00
Toyota	Camry	12000.00
Honda	Pilot	36999.00
Toyota	Camry	18000.00
Toyota	Celica	5000.00
Isuzu	Trooper	7700.00
Ford	Focus	4400.00
Ford	Fiesta	800.00
Mazda	626	
Dodge	Durango	2700.00
Nissan	Rogue	30000.00

But let's not get too far ahead of ourselves. First, here is the code that produces the basic version of the table. The output of this script is more interesting if you have a lot of cars in your database, so if you haven't done so already, use the script "**createdb.php**" included with the sample code to populate your inventory table with a lot of cars.

Class: - B.Com, B.Com (Hons), BBA & BAJMC II Year Subject: Web Development Using PHP & MySQL

Code

```
<html>
<head>
    <meta charset="utf-8">
    <title>Sam's Used Cars</title>
   </head>

<body>
<h1>Sam's Used Cars</h1>
<h3>Complete Inventory</h3>
<?php
/* Try to query the database */
if ($result = $mysqli->query($query)) {
   // Don't do anything if successful.
}
else
{
    echo "Error getting cars from the database: " .
mysql_error()."<br>";
}

// Create the table headers
echo "<table id='Grid' style='width: 80%'><tr>";
echo "<th style='width: 50px'>Make</th>";
echo "<th style='width: 50px'>Model</th>";
echo "<th style='width: 50px'>Asking Price</th>";
echo "</tr>\n";

$class ="odd";  // Keep track of whether a row was even
or odd, so we can style it later

// Loop through all the rows returned by the query,
creating a table row for each
while ($result_ar = mysqli_fetch_assoc($result)) {
    echo "<tr class=\"$class\">";
    echo "<td>" . $result_ar['Make'] . "</td>";
    echo "<td>" . $result_ar['Model'] . "</td>";
      echo "<td>" . $result_ar['ASKING_PRICE'] . "</td>";
   echo "</td></tr>\n";

   // If the last row was even, make the next one odd and
vice-versa
    if ($class=="odd"){
        $class="even";
    }
    else
    {
        $class="odd";
```

```
46.    }
47.}
48.echo "</table>";
49.$mysqli->close();
50.?>
51. </body>
52.
53.</html>
```

Code Explained

I won't walk you through **every** line anymore, as I no longer think you need it. From now on, I'll just explain the important ones.

Line 11 is our first use of the **include** option which refers to an external file named db.php which will be included in this script *just as if it were part of the same file*. I highlighted line 11 above in blue and the code below in blue, in hopes that you would better understand how it works. The content of the blue box below is substituted into the code for the blue line (11) above, so that both files are combined into a single script.

The contents of the 'db.php' file are shown below:

```
<?php
$mysqli = new mysqli('localhost', 'root', 'mypassword',
'cars' );
/* check connection */
if (mysqli_connect_errno()) {
   printf("Connect failed: %s\n", mysqli_connect_error());
   exit();
}
//select a database to work with
$mysqli->select_db("Cars");

?>
```

The code in the db.php file is identical to the code explained as line 29 in the previous section, so I won't explain it again here. That's another key benefit of include files. Once the code inside it works, you don't really have to think about it much anymore.

```
12.$query = "SELECT * FROM INVENTORY";
```

Line 12 is the query that produces the list of cars to be displayed. In this simple case, we are selecting all the cars.

```
14.  if ($result = $mysqli->query($query)) {
15.   // Don't do anything if successful.
16.  }
17.  else
18.  {
19.    echo "Error getting cars from the database: " .
mysql_error()."<br>";
20.   }
```

Lines 14 – 20 runs the query and displays an error message if the query fails.

```
23. echo "<table id='Grid' style='width: 80%'><tr>";
```

Line 23 is an opening tag to create a table with the ID of 'grid'. The ID is optional but makes it easy to apply styles to the table later. style=width:80% prevents the column from extending to fill the entire screen; instead it takes 80% of the width. <tr> starts the Table Row with the opening <tr> tag.

```
24. echo "<th style='width: 50px'>Make</th>";
25. echo "<th style='width: 50px'>Model</th>";
26. echo "<th style='width: 50px'>Asking Price</th>";
27. echo "</tr>\n";
```

Lines 24– 27 create the first row of the table, the row that contains the column titles of make, model, and price. Line 27 is a closing Table Row tag, followed by a new line.

```
29.$class ="odd";  // Keep track of whether a row was
even or odd, so we can style it later
```

Line 29 sets the value of a variable called $class to 'odd' because the first data row in our table will be odd. As we loop through each row of data, we'll alternately set the $class to the value of either 'odd' or 'even'. We do this so we can style the table later to have alternate rows show different coloring to make it easier on the eyes.

```
31.  // Loop through all the rows returned by the query,
creating a table row for each
32.  while ($result_ar = mysqli_fetch_assoc($result)) {
33.    echo "<tr class=\"$class\">";
34.    echo "<td>" . $result_ar['Make'] . "</td>";
35.    echo "<td>" . $result_ar['Model'] . "</td>";
36.    echo "<td>" . $result_ar['ASKING_PRICE'] . "
</td>";
37.    echo "</td></tr>\n";
```

Lines 31 to 37 create a row in the HTML table to correspond with each row in the

database table that we extracted using the query. Each table cell contains data from the mySQL table. For instance, line 34 (echo "<td>" . $result_ar['Make'] . "</td>";) should produce something like

<td>Ford</td>

because **$result_ar['Make']** says get the value of Make (one of the columns in the table, and in this case 'Ford') and put it here between the <td> tags. Take the time to really understand what that line is doing, because if you can understand this, you can do virtually anything! Remember the . character means join these two strings.

```
39.  // If the last row was even, make the next one odd
and vice-versa
40.   if ($class=="odd"){
41.      $class="even";
42.   }
43.   else
44.   {
45.      $class="odd";
46.   }
```

Lines 39 – 46 just alternate the value of $class from even to odd.

```
47. }
```

Line 47 closes the While loop.

```
48. echo "</table>";
```

Line 48 closes the table with the </table> tag.

```
49.  $mysqli->close();
```

Line 49 closes the mySQL database.

```
50. ?>
```

Line 50 indicates the end of the PHP code.

```
51. </body>
```

Line 51 is the end of the body in the HTML page.

```
53. </html>
```

Finally, **line 53** indicates the end of the HTML.

Exercise: Tweaking the SELECT

Go back and modify this code so that it doesn't select **all** the cars but rather a subset that pleases you.

Improving the look of the table with CSS

Here's the CSS that improves the look of the form. This style information is added to the <head> section of the page, but often people put styles into a separate style sheet too. See the file **viewcarswithstyle.php** to see the form in action.

Explaining how CSS works is beyond the scope of this book, and a topic in itself. But the important thing is to see how easily we were able to change the look of the HTML table using a little style information. Take a look at the complete style sheet here, and I'll explain it next.

Class: - B.Com, B.Com (Hons), BBA & BAJMC II Year Subject: Web Development Using PHP & MySQL

```
<style>
    /* The grid is used to format a table */
#Grid
{
font-family:"Trebuchet MS", Arial, Helvetica, sans-serif;
width:80%;
border-collapse:collapse;
margin-left: auto;
margin-right: auto;
}
#Grid td, #Grid th
{
font-size:1em;
border:1px solid #61ADD7;
padding:3px 7px 2px 7px;
}
#Grid th
{
font-size:1.1em;
text-align:left;
padding-top:5px;
padding-bottom:4px;
background-color:#C2D9FE;
color: lightslategray;
```

Class: - B.Com, B.Com (Hons), BBA & BAJMC II Year Subject: Web Development Using PHP & MySQL

```
26.}
27.#Grid tr.odd td
28.{
29.color:#000000;
30.background-color: #F2F5A9;
31.}
32.
33.#Grid tr.even
34.{
35.color:#000000;
36.background-color: white;
37.}
38.#Grid head
39.{
40.color:#000000;
41.background-color:teal;
42.}
43.
44. </style>
```

CSS Explained

```
1. <style>
2.    /* The grid is used to format a table */
```

Line 1 opens the <style> tag, telling the browser that what follows is a style sheet. Line 2 is a comment.

```
3. #Grid
4. {
5. font-family:"Trebuchet MS", Arial, Helvetica, sans-serif;
6. width:80%;
7. border-collapse:collapse;
8. margin-left: auto;
9. margin-right: auto;
10. }
```

Line 3 says to select an item on the page with the id of Grid. The # symbol is the selector to select something defined using an id, and what follows is the name of the specific thing you want to select. See line 23 of the previous PHP script, which set the id of our table to 'grid' with the line **echo "<table id='Grid' style='width: 80%'>;** Since we have a table with an id='Grid', this style will apply.

Everything that follows between the { and the } symbols define the style for that item.

We pick font, border, margin, etc.

```
11. #Grid td, #Grid th
12.  {
13.  font-size:1em;
14.  border:1px solid #61ADD7;
15.  padding:3px 7px 2px 7px;
16. }
```

Line 11 specifies that the following lines only apply to <td> and <th> tags, if they appear within an item with an ID of 'Grid'.

Each line that follows gets more specific about how an item should be formatted. A specific selector overwrites a general one. So we started off specifying default formatting for Grid, but later we modified specific elements of the grid item. The next bit is how we color alternate rows differently:

```
27.#Grid tr.odd td
28.{
29.color:#000000;
30.background-color: #F2F5A9;
31.}
```

Line 27 says to selects a <td> tag, within a table row <tr> if it is a member of the class odd. Look at the HTML that is output by the script. You'll see a table row for the table defined like this: <tr class='odd'> or <tr class='even'>.

There is another selector for the table headers. It does make sense if you look at it long enough. The # symbol in CSS is a selector. OK, that's it for now. Maybe someday I'll do a "Joy of CSS" book. Let me know…

Modifying the form to link to the detail page

The last thing this form needs is way to link to a specific car. When the site visitor clicks on a specific car in a row, it should take them to more detail about that specific car. In other words, it should take them to the 'car detail' page. We're going to have to make that page, of course.

The <a> tag defines a hyperlink, which is used to link from one page to another.

Amazingly, 'A' actually stands for Anchor. I don't associate anchors with jumping around, but I guess someone did, and the tag somehow stuck.

The most important attribute of the <a> element is the href attribute, which indicates the link's destination.

Note that for this to work we will need to **build** the detail page because otherwise the link will naturally fail. Nothing happens automatically. Assuming that the detail page exists, we can modify the code on line 34 that reads as:

```
echo "<td>" . $result_ar['Make'] . "</td>";
```

to instead read as:

```
 echo "<td><a href='viewcar.php?
VIN=".$result_ar['VIN']."'>" . $result_ar['Make'] . "</a>
</td>";
```

What this does is create an 'anchor' or a link which makes the first column of each row a clickable link. It should output HTML similar to:

```
<td><a href='viewcar.php?
VIN=123234FE221'>Nissan</a></td>
```

You can see that the URL created will be similar to **/viewcar.php?VIN=123234FE221** as shown above. This tells the browser to open the **viewcar.php** file and pass it the query string of VIN= followed by a VIN. It is called a **query string** because the primary purpose of passing data to a form this way is so it can use the data in a SQL query—and that's exactly what we are going to do.

Remember back when I said to use 'Post' rather then 'Get' when submitting a form? If you *had* used get, clicking the submit button would send to the browser a really long URL with all the field names and values appended to the end of it as a query string in a format similar to *?Make=Ford&Model=Explorer, etc.* We are going to take advantage of that technique to create our own query string and pass it to a script.

For now, clicking on the link will only trigger an error, because the **viewcar.php** file does not yet exist, but that's what we're going to build next.

Forms that Display Detail Data

Class: - B.Com, B.Com (Hons), BBA & BAJMC II Year Subject: Web Development Using PHP & MySQL

Once a site visitor has identified a car that they want more information about, the car shopper will want to click on a particular car to learn more about it. So we'll make a PHP page to handle this. We'll call this the Car Detail page, and its file name will be viewcar.php.

Again, we'll keep the example relatively simple for the purpose of following the logic. If all goes well, clicking on a car from the previous screen will bring up a form similar to:

Code

```
1. <html >
2. <head>
3. <title>Sam's Used Cars</title>
4. </head>
5.
6. <body>
7.
8. <h1>Sam's Used Cars</h1>
9. <?php include 'db.php';
10.$vin = $_GET['VIN'];
11.$query = "SELECT * FROM INVENTORY WHERE VIN='$vin'";
12./* Try to query the database */
13.if ($result = $mysqli->query($query)) {
14.   // Don't do anything if successful.
15.}
16.else
17.{
18.    echo "Sorry, a vehicle with VIN of $vin cannot be found " .
mysql_error()."<br>";
19.}
```

```
20.// Loop through all the rows returned by the query, creating a
table row for each
21.while ($result_ar = mysqli_fetch_assoc($result)) {
22.    $year = $result_ar['YEAR'];
23.     $make = $result_ar['Make'];
24.    $model = $result_ar['Model'];
25.    $trim = $result_ar['TRIM'];
26.    $color = $result_ar['EXT_COLOR'];
27.    $interior = $result_ar['INT_COLOR'];
28.    $mileage = $result_ar['MILEAGE'];
29.    $transmission = $result_ar['TRANSMISSION'];
30.    $price = $result_ar['ASKING_PRICE'];
31.}
```

```
32.echo "$year $make $model </p>";
33.echo "<p>Asking Price: $price </p>";
34.echo "<p>Exterior Color: $color </p>";
35.echo "<p>Interior Color: $interior </p>";
36.
37.$mysqli->close();
38.?>
39.
40.</body>
41.</html>
```

Code Explained

```
1. <html >
```

Line 1 opens the HTML tag and starts the document.

```
2.<head>
3.<title>Sam's Used Cars</title>
4.</head>
```

Lines 2 – 4 are the head tags, and in between specifies the document title, 'Sam's Used Cars'.

```
8.<h1>Sam's Used Cars</h1>
```

Line 8 is ordinary HTML; it prints Sam's Used Cars at the top of the page in a headline style type.

```
9.<?php include 'db.php';
```

Line 9 specifies that the current script *include* the db.php file, which logs into the mySQL database.

```
10.  $vin = $_GET['VIN'];
```

Line 10 creates a variable called $vin and assigns it the value that follows VIN= in the URL string. Remember, for this form to work, you have to pass it the VIN like this: **/viewcar.php?VIN=123234FE221.** We use the command $_GET because when you submit a form using get the values are appended to the URL in a similar fashion.

```
11.  $query = "SELECT * FROM INVENTORY WHERE
VIN='$vin'";
```

Line 11 builds a query using the value passed to the form in the Query String, and assigns it to the cleverly named variable $query. See why we call it a 'query string'?

```
12.  /* Try to query the database */
13.  if ($result = $mysqli->query($query)) {
14.   // Don't do anything if successful.
15.  }
16.  else
17.  {
18.    echo "Sorry, a vehicle with VIN of $vin cannot be
found " . mysql_error()."<br>";
```

Lines 12 – 19 run the query against the mySQL database and create something called a 'result set'. A result set is the set of data that results from the running of a query. This result set is assigned to the variable $result.

```
20.  // Loop through all the rows returned by the query,
creating a table row for each
21.  while ($result_ar = mysqli_fetch_assoc($result)) {
22.    $year = $result_ar['YEAR'];
23.     $make = $result_ar['Make'];
24.    $model = $result_ar['Model'];
25.    $trim = $result_ar['TRIM'];
26.    $color = $result_ar['EXT_COLOR'];
27.    $interior = $result_ar['INT_COLOR'];
28.    $mileage = $result_ar['MILEAGE'];
29.    $transmission = $result_ar['TRANSMISSION'];
30.    $price = $result_ar['ASKING_PRICE'];
31.  }
```

Lines 20 – 31 loop through 'all' the rows returned as a result of the query. In our case, since VINs are unique we would only expect to get one row of data back, but we are using basically the same technique we learned in the prior section – Forms that Display Summary Data.

```
22.    $year = $result_ar['YEAR'];
23.     $make = $result_ar['Make'];
...
```

Lines 22 to 30 assign a series of variables with the values of the specified data columns, which match the names of the columns in the database table 'inventory'.

```
31.  }
```

Line 31 closes the while loop.

Forms that Edit Data

If you understand how to make *Forms that Add Data to a Database*, and you understand *Forms that Display Detail Data*, it isn't much of a stretch (conceptually anyway) to make a form that Edits data. Simply create a form just like the one you made to add data, but before displaying it retrieve data from the database and pre-populate it with values.

Instead of executing a SQL Insert command when the user clicks submit, instead execute an Update.

Forms that Delete Data

To delete a specific record from a database, you need a way for the user to select the data they want to delete. You already learned how to do this in the section *Forms that Display Summary Data.* In the section *Modifying the form to link to the detail page* we created an <HREF> link that takes the user to a detail page, and you can use that same technique to take them to a delete page, such as the one shown below:

Code to delete data

```
1. <html>
2. <head>
3. <title>Sam's Used Cars</title>
4. </head>
5. <body>
6. <h1>Sam's Used Cars</h1>
7. <?php
8. include 'db.php';
9. $vin = $_GET['VIN'];
10.$query = "DELETE FROM INVENTORY WHERE VIN='$vin'";
11.echo "$query <BR>";
12./* Try to execute the DELETE against the database
```

```
22.$mysqli->close();
23.
24.?>
25.
26.</body>
27.</html>
```

Code Explained

```
1.<html>
2.<head>
3.<title>Sam's Used Cars</title>
4.</head>
5.<body>
6.<h1>Sam's Used Cars</h1>
```

Lines 1 – 6 set up the basics of the page. We open an <html>, open and close the <head> tags, and start the body with a headline proclaiming "Sam's Used Cars".

```
7.  <?php
8.  include 'db.php';
```

Lines 6 – 7 are also familiar to us by now. We open the php tag and add the insert line to connect us to our mySQL database.

```
9.   $vin = $_GET['VIN'];
10. $query = "DELETE FROM INVENTORY WHERE
VIN='$vin'";
11. echo "$query <BR>";
```

Line 9 gets the VIN from the query string. Remember, this page will be called with ? VIN='23ABC..' appended to the end. **Line 10** builds a SQL delete statement using the VIN, so we know which vehicle to delete. **Line 11** simply writes the query to screen so we can see the query we built. It is not strictly required for the function to work.

```
14.  if ($result = $mysqli->query($query)) {
15.   Echo "The vehicle with VIN $vin has been deleted.";
16.  }
17.  else
18.  {
19.    echo "Sorry, a vehicle with VIN of $vin cannot be
found " . mysql_error()."<br>";
20.   }
```

Lines 14 through 20 do the actual work. Line 14 performs the query, and returns True if the query succeeds. If so, line 15 prints a success message to the screen, and if not, line 19 prints a failure message to the screen.

```
22. $mysqli->close();
23.
24. ?>
25.
26. </body>
27. </html>
```

The rest of the page close the database connection, closes the php tag, closes the body tag, and finally closes the html tag.

Exercise

To add edit and delete functionality, simply add two new columns to the table with the links for edit and delete, and call the appropriate php page. **deletecar.php** has been provided, while **editcar.php** you will have to make yourself. If you *absolutely can't* get **editcar.php** to work, I did include it in the sample code. Just do yourself a favor and TRY to make it.

For the answer to this challenge, look at the sample page **viewcarswithstyle2.php**, which is included in the sample code.

13

Session Variables

Introduction

Variables in PHP typically have a specific and limited scope—generally, a variable is only available on the page on which it was declared. The prime exception to this rule is when you declare a variable inside a function, it only works in that function.

But what if you want access to the **same** variable across multiple pages in your application? For instance, I'm a regular shopper on Amazon.com. If you are too, you may have noticed that once you're logged in, **every page** has your name on the top of it.

Presumably, there is a variable in a script somewhere called something like $FirstName containing the value 'Alan'. By now, you could probably easily write such a script. Here's a hint:

```
echo "$FirstName's Amazon.com";
```

But how does that value $FirstName pass from page to page as I wander about the site? And how does the site keep track of hundreds of unique $FirstName variables for all the

unique customers who happen to be on the site at the same time? The answer is session variables.

Sessions

A session variable is a special kind of variable that, once set, is available to all the pages in an application for as long as the user has their browser open, or until the session is explicitly terminated by the developer (you).

The great thing about session variables is that PHP will magically keep track of which particular session variable goes with each particular user. So while my Amazon.com experience will always say *"Alan's Amazon"*, yours will say something different (unless your name also happens to be Alan, of course.) Sessions work by creating a unique id (UID) for each visitor and storing variables based on this UID. The UID is typically stored in a cookie.

A cookie, also known as an HTTP cookie, web cookie, or browser cookie, is a small piece of data sent from a website and stored in a user's web browser while a user is browsing a website.

It doesn't really matter *how* they work, the important thing is that they *do* work. And, they are very cool. They open up a whole realm of possibilities for customizing your web application for a specific customer. For example, in the case of **Sam's Used Cars**, you could ask a customer their preferred car color, make/model, features, etc. From that point on, you can customize the pages to reflect the customers' preferences. For example, *Hey look, this car has a sunroof!* (And it's red too!) It's just a sample app, so it's OK to code annoying features to learn something valuable.

Once a user closes their browser, the cookie will be erased and the session will end. So sessions are not a good place to store data you intend to keep for long. The right place to store long-term data is in a database. Of course, sessions and databases *can* work together. For instance, you can store a user's preferences in a database, and retrieve them from the database when the user "logs in" or types in their email address or does whatever it is that you coded for them to identify themselves. Once the data is retrieved, assign the preferences to the session variables and they will be available from then on.

Starting a PHP Session

Before you can store user information in your PHP session, you must first start up the session using the **session_start()** function. The session_start() function **must** appear BEFORE the <html> tag, or it won't work.

```
<?php session_start(); ?>

<html>
    <body>
        <p>Hello world</p>
    </body>
</html>
```

The code above will start the user's session with the server and allow you to start saving user information into session variables.

Using Session Variables

The correct way to store and retrieve session variables is to use the PHP **$_SESSION** variable:

Store a variable

```
<?php
session_start();
// store session data
$_SESSION['FirstName']='Alan';
?>

<html>
<body>
```

Retrieve a variable

```
<?php
//retrieve session data
 echo  $_SESSION['FirstName']."'s Amazon";
?>
```

Output: **Alan's Amazon**

Checking for a variable

You can check to see if a session variable is available or not by using the **isset()** function.

```
bool isset ( mixed $var [, mixed $... ] )

Determines if a variable is set and is not
NULL.  You can unset a variable with unset().
```

Here's an example:

```
<?php
session_start();
if(isset($_SESSION['FirstName']))
echo  $_SESSION['FirstName']."'s Amazon";
else
$_SESSION['views']=1;
echo "Welcome to Amazon";
?>
```

Destroying a Session

Class: - B.Com, B.Com (Hons), BBA & BAJMC II Year Subject: Web Development Using PHP & MySQL

If you wish to delete some session data, you can use the **unset()** function. If you want to delete it all, use the **session_destroy()** function. The **unset()** function is used to delete a specific session variable:

```
<?php
session_start();
if(isset($_SESSION['FirstName']))
 unset($_SESSION['FirstName']);
?>
```

You can also completely destroy all the session by calling the **session_destroy()**

function:

```
<?php
  session_destroy();
?>
```

Note: session_destroy() will reset your session and you will lose all your stored session data. This is an easy way to implement a logout function.

If you would like to learn more about Session Variables, I have a whole book on this topic titled “The Joy of PHP: Deep Dive into Sessions”.

Working with Images

Introduction

A used car web site would not be of much use to the typical car shopper without providing images of the cars, so in this chapter we will cover how to add images to our site. It would be rather simple if each car had a single image associated with it—in that case, we could simply add an additional column to our inventory table called ‘image’ (or something equally descriptive, such as ‘primary_image’) which would store the file name of the image associated with the particular car.

Then we would build PHP to retrieve the image name and insert it into an HTML image tag on the car details page.

To display an image in your HTML page, use the syntax **<img src="filename">** in the simplest form or add optional parameters to specify height and width, and text to display if the image isn't available (or for screen readers) such as **<img src="smiley.gif" alt="Smiley face" height="42" width="42">**

```
1. <html >
2. <head>
3. <title>Sam's Used Cars</title>
4. </head>
5.
6. <body>
7.
8. <h1>Sam's Used Cars</h1>
9. <?php include 'db.php';
10.$vin = $_GET['VIN'];
11.$query = "SELECT * FROM INVENTORY WHERE VIN='$vin'";
12./* Try to query the database */
13.if ($result = $mysqli->query($query)) {
14.   // Don't do anything if successful.
15.}
16.else
17.{
18.    echo "Sorry, a vehicle with VIN of $vin cannot be found " .
mysql_error()."<br>";
19.}
20.// Loop through all the rows returned by the query, creating a
table row for each
21.while ($result_ar = mysqli_fetch_assoc($result)) {
22.   $year = $result_ar['YEAR'];
23.    $make = $result_ar['Make'];
24.   $model = $result_ar['Model'];
25.   $trim = $result_ar['TRIM'];
26.   $color = $result_ar['EXT_COLOR'];
27.   $interior = $result_ar['INT_COLOR'];
28.   $mileage = $result_ar['MILEAGE'];
29.   $transmission = $result_ar['TRANSMISSION'];
```

Of course, PHP would be well suited for this. We would read the file name from the database and use PHP to create the image tag dynamically.

For instance, we *could* modify our earlier example, which shows the detail for a specific car by adding the lines highlighted in red as follows:

```
32.}
33.echo "$year $make $model </p>";
34.echo "<p>Asking Price: $price </p>";
35.echo "<p>Exterior Color: $color </p>";
36.echo "<p>Interior Color: $interior </p>";
37.echo "<IMG src='$image'>";
38.
39.$mysqli->close();
40.?>
41.
42.</body>
43.</html>
```

This example assumes that we have a column in our database called **Primary_Image,** which contains the file name of an image file that is stored on our server. The sample files home page contains a script that makes this modification, if you are so inclined.

If the images were in a folder called 'images', the line would read:

```
echo "<IMG src='images\$image'>";
```

Exercise: Viewing Images

Get the above example to work. Create an images folder underneath the folder that is running the car lot application and put some images into it. Modify your inventory table to add a **Primary_Image** field and enter some values in that field to associate specific cars with specific images.

Make a copy of the viewcar.php script (call it viewcar-backup.php in case you need it later), then modify the viewcar.php as shown in red above so that it reads the image location out of the database and inserts the image into the page using the <img> tag.

Pulling an unknown number of images from a database

Assuming you got the above exercise to work, you must admit that it is pretty slick. Congratulations, you are officially awesome. But, we can do **much more**. Just having *one* image of a car doesn't really reflect the reality of a visitor's expectation of a car site. More likely a visitor to Sam's Used Cars web site would want to see *many* images of a car he or she is interested in, and our site will have to accommodate this. Some cars might have only one image, but some might have 10 or more. It will be different for each car. So how would we accomplish this? Having a single column called **Primary_Image** is obviously not the permanent solution. As soon as you show it to Sam, he'll surely say 'But what if I have two pictures of the car to show?' That's the nature of web development sometimes. One good idea sparks another. Don't get frustrated when this happens, but rather think to yourself, 'Wow, I inspired an even better idea!'

The easiest way to handle a variable number of images would be to create a database table to store them in.

Let's add a table called 'images' to our cars database. It should have the columns ID, VIN, and ImageFile.

Exercise: Create a Database Table to store images

Use phpMyAdmin to create this table, like so.

Now you need to populate the table with some sample data. Here's what I did. Go to http://www.cars.com and search for some cars. Copy the VIN to the clipboard, and save some the pictures of the car to your hard drive. Enter a row in the images table for each of the images you save, and enter the VIN of the car for each one. There should also be a corresponding entry in the inventory table for that car, with the exact same VIN. It's easy to do in phpMyAdmin. Don't worry about trying to automate that part of it yet.

Exercise: Modify the viewcar.php page to show multiple images

Once you have some sample data that matches up specific VINs with specific images, it's actually pretty easy to display those images on the page along with the description of the car. Here's a code snippet you can append to the **viewcars.php** form to extract the names of the images for the selected car.

The assumption of this script is you have the VIN of the car in the variable $vin, and that you have included 'db.php' to establish the database connection.

```
1. <?php
2. $query = "SELECT * FROM images WHERE VIN='$vin'";
3. /* Try to query the database */
4. if ($result = $mysqli->query($query)) {
5.     // Got some results
6.     // Loop through all the rows returned by the query, creating a
table row for each
7. while ($result_ar = mysqli_fetch_assoc($result)) {
8.       $image = $result_ar['ImageFile'];
9.       echo "<img src='uploads/$image' width= '250'>  " ;
10.}
11.}
12.$mysqli->close();
```

Code explained

```
2. $query = "SELECT * FROM images WHERE
VIN='$vin'";
```

Line 2 sets up the query whereby we select all the fields in the images table for the specific car (WHERE VIN=).

```
4. if ($result = $mysqli->query($query)) {
```

Line 4 runs the query and checks to see if any results were returned from the database.

```
7.  while ($result_ar = mysqli_fetch_assoc($result)) {
8.      $image = $result_ar['ImageFile'];
9.      echo "<img src='uploads/$image' width= '250'> " ;
10.  }
```

Lines 7 – 10 loops through the result set as many times as there are rows. In other words, if there were five images for a specific car, there would be five rows of data returned and the while loop would go around five times.

```
11. }
12. $mysqli->close();
```

Line 11 closes the if statement and the line 12 closes the connection to the mySQL database.

PHP File Uploads

Introduction

In the previous section, we captured images for our cars and then saved them manually onto the hard drive. That's cool, but tedious. What would be *really* cool would be to simply select a car in our inventory and click a button called "Add Image", and let the script handle the rest

– putting the file in the right place and creating the correct row in the images table using the VIN of the selected vehicle.

That's what we'll do next.

Create an Upload File form

In its most basic incarnation, here is an HTML form you can use to upload a file.

```
<html>
<body>

<form action="upload_file.php" method="post"
enctype="multipart/form-data">
        <label for="file">Filename:</label>
        <input type="file" name="file" id="file"><br>
        <input type="submit" name="submit"
value="Submit">
</form>

</body>
</html>
```

There are a couple of things worth pointing out.

First, notice the form attributes: action='upload_file.php' means that when you click the submit button, the result of the form post will be passed to the **upload_file.php** script for further processing. Next, the enctype="multipart/form-data" is a new one for us. Here we are specifying the encoding type to be used by the form. You have to specify that it is **multipart/form-data** if you are including a file upload control on a form, so the browser knows to pass the file as a file, and not as just another big block of text.

We also have a new type of input box. In the past, we've been using the input boxes mostly to allow users to type in text. When you specify that an input type="file", the browser handles it differently. It will put a browse button next to the input field,allowing the user to select a file from his or her computer.

Create a Script to Process the Uploaded File

The form above specified that the post be processed by 'upload_file.php'. This script is used to do something with the file once it's been uploaded. The script that follows simply echoes back to the browser *some* of the attributes of the file that has just been uploaded. There are, of course, other file attributes that we won't cover, because you probably won't ever need to use them.

```
<?php
if ($_FILES["file"]["error"] > 0)
  {
  echo "Error: " . $_FILES["file"]["error"] . "<br>";
  }
else
  {
  echo "Upload: " . $_FILES["file"]["name"] . "<br>";
  echo "Type: " . $_FILES["file"]["type"] . "<br>";
  echo "Size: " . ($_FILES["file"]["size"] / 1024) . "
kB<br>";
  echo "Stored in: " . $_FILES["file"]["tmp_name"];
  }
?>
```

It doesn't really matter *how* they work, the important thing is that they *do* work. And, they are very cool. They open up a whole realm of possibilities for customizing your web application for a specific customer. For example, in the case of **Sam's Used Cars**, you could ask a customer their preferred car color, make/model, features, etc. From that point on, you can customize the pages to reflect the customers' preferences. For example, *Hey look, this car has a sunroof!* (And it's red too!) It's just a sample app, so it's OK to code annoying features to learn something valuable.

I highlighted in yellow the parts that need to match. In other words, if the name of the input control on the upload form refers to the file as 'foo', like <input type="file" name="foo"> you would **also** have refer to it as foo on the script that follows, such as $_FILES["foo"]["name"]. The actual name doesn't matter, but what does matter is consistency.

When you upload a file using PHP, the file is stored in a temporary folder. Unless you specifically do *something* with the file, it will soon disappear.

For Sam's Used Cars, the ideal thing to do would be to upload the file, copy the file into a specific folder, and then create a record in the images table that inserts the proper vehicle VIN and the file name of the image we just uploaded. In the sample data, see the script ViewCarsAddImage.php to see this exact concept in action.

Code: ViewCarsAddImage.php

```
1. <?php
2.  include 'db.php';
3.  $vin = trim($_POST['VIN']);
4. if ($_FILES["file"]["error"] > 0)
5.  {
6.  echo "Error: " . $_FILES["file"]["error"] . "<br>";
7.  }
8. else
9.  {
10. echo "Upload: " . $_FILES["file"]["name"] . "<br>". "\n";
11. echo "Type: " . $_FILES["file"]["type"] . "<br>". "\n";
12. echo "Size: " . ($_FILES["file"]["size"] / 1024) . " kB<br>". "\n";
13. echo "VIN: ".$vin."<br>";
14. echo "Stored temporarily as: " .
$_FILES["file"]["tmp_name"]."<br><BR>". "\n";
15. $currentfolder =  getcwd();
16. echo "This script is running in: " .$currentfolder."<br>". "\n";
17. $target_path = getcwd() ."/uploads/";
18. echo "The uploaded file will be stored in the folder:
".$target_path."<br>". "\n";
19.
20. $target_path = $target_path . basename( $_FILES['file']['name']);
21. $imagename = "uploads/". basename( $_FILES['file']['name']);
22. echo "The full file name of the uploaded file is '".
$target_path."'<br>". "\n";
23.
```

```
24.  echo "The relative name of the file for use in the IMG tag is " .
$imagename ."<br><br>". "\n";;
25.
26.if(move_uploaded_file($_FILES['file']['tmp_name'], $target_path)) {
27.    echo "The file ".  basename( $_FILES['file']['name']). " has been
uploaded<br>". "\n";
28.
29.    // Create a database entry for this image
30.    if (mysqli_connect_errno()) {
31.        printf("Connect failed: %s\n", mysqli_connect_error());
32.        exit();
33.    }
34.
35.  echo 'Connected successfully to mySQL. <BR>';
36.  $file_name =  $_FILES["file"]["name"];
37.  $query = "INSERT INTO images (VIN, ImageFile) VALUES ('$vin',
'$file_name')";
38.  echo $query."<br>\n";
39.   echo  "<a href='AddImage.php?VIN=";
40.   echo $vin;
41.   echo "'>Add another image for this car </a></p>\n";
42./* Try to insert the new car into the database */
43.if ($result = $mysqli->query($query)) {
44.        echo "<p>You have successfully entered $target_path into
the database.</P>\n";
45.
46.    }
47.    else
48.    {
49.        echo "Error entering $VIN into database: " .
mysql_error()."<br>";
50.    }
```

```
51.    $mysqli->close();
52.    echo "<img src='$imagename' width='150'><br>";
53.
54.} else{
55.    echo "There was an error uploading the file, please try again!";
56.}
57.  }
58.
59.  include 'footer.php'
60.?>
```

Code Explained

```
1.<?php
2. include 'db.php';
```

Line 1 opens the php tag, and **line 2** adds the necessary include file to connect to our database.

```
3. $vin = trim($_POST['VIN']);
```

Line 3 creates a variable called $vin and assigns it the value that was passed to it using when a form was posted. Again, see this in action with the sample scripts included with this book. This is not the only way we could have done this. We could also have passed the VIN in a query string, the technique we used in **viewcar.php**

```
4.  if ($_FILES["file"]["error"] > 0)
5.  {
6.    echo "Error: " . $_FILES["file"]["error"] . "<br>";
7.  }
```

Lines 4 – 7 test to see if a file was, in fact, uploaded. If not, an error is printed using line 6.

```
8. else
9.  {
```

Beginning with **Line 9**, the script begins to process the uploaded file.

```
10.  echo "Upload: " . $_FILES["file"]["name"] . "<br>".
"\n";
11.  echo "Type: " . $_FILES["file"]["type"] . "<br>".
"\n";
12.  echo "Size: " . ($_FILES["file"]["size"] / 1024) . "
kB<br>". "\n";
13.  echo "VIN: ".$vin."<br>";
```

Lines 10 – 12 print information about the file, and **line 13** prints the VIN, just to make sure we got it without any problems.

```
14. echo "Stored temporarily as: " . $_FILES["file"]
["tmp_name"]."<br><BR>". "\n";
```

Line 14 tells us the name that PHP used to temporarily store the uploaded file.

TIP: Notice that on the end of the line I also have it write "\n", which means to add a new line at the end of this. This doesn't affect the script at all, but it does put a new line on the HTML that is created by the script. Putting \n at the end of the line on scripts makes the HTML code easier to read when you look at a page and select View Source— something that every PHP developer has to do from time to time.

```
15. $currentfolder = getcwd();
16. echo "This script is running in: " .$currentfolder."
<br>". "\n";
```

Line 14 tells us the name that PHP used to temporarily store the uploaded file.

TIP: Notice that on the end of the line I also have it write "\n", which means to add a new line at the end of this. This doesn't affect the script at all, but it does put a new line on the HTML that is created by the script. Putting \n at the end of the line on scripts makes the HTML code easier to read when you look at a page and select View Source— something that every PHP developer has to do from time to time.

Line 15 uses the command **getcwd()** to figure out the name of the folder in which the current script is running. Why did I want that? Because I want to put the uploaded file into a folder that is under the current folder, and to do *that* you need to know where you are. **Line 16** outputs what it just learned.

getcwd() The getcwd() function returns the current directory. This function returns the current directory on success and FALSE on failure.

```
17.  $target_path = getcwd() ."/uploads/";
```

In **line 17**, we create a variable called $target_path and assign it a value by adding two strings together using the . character. The two strings we added are the current directory and /uploads/. We are creating the target path to specify where we want the uploaded file to be put— in the uploads folder.

```
18.  echo "The uploaded file will be stored in the folder:
".$target_path."<br>". "\n";
```

Line 18 outputs the result of the calculation to set the target path.

```
20.  $target_path = $target_path . basename(
$_FILES['file']['name']);
```

In **line 20** we tweak the target path yet again, this time appending the original file name of the uploaded file to it.

```
21.  $imagename = "uploads/". basename(
$_FILES['file']['name']);
```

Line 21 calculates the name of just the image file without the entire file path. This is because when you are working with HTML <img> tags, you don't have to specify the entire path of the image; you only need to specify where it is *relative to where you are*.

```
22.  echo "The full file name of the uploaded file is '".
$target_path."'<br>". "\n";
23.
24.  echo "The relative name of the file for use in the
IMG tag is " . $imagename ."<br><br>". "\n";
```

Lines 22 and 24 output the values of these calculations so you can see what was the result. Of course, if this was a "real" web site for a used car lot, you wouldn't want all this extra information going to the browser.

> The reason for all these echo statements is so you can follow along with the code as it executes.

```
26.  if(move_uploaded_file($_FILES['file']['tmp_name'],
$target_path)) {
27.    echo "The file ".  basename( $_FILES['file']
['name']). " has been uploaded<br>". "\n";
```

Lines 26 moves the uploaded file from the temporary location assigned by PHP into the target path that you calculated in line 20. **Line 27** informs you of this fact.

```
29.   // Create a database entry for this image
30.    if (mysqli_connect_errno()) {
31.        printf("Connect failed: %s\n",
mysqli_connect_error());
32.        exit();
33.    }
34.
35.  echo 'Connected successfully to mySQL. <BR>';
```

Next, we want to create a record in the images table that points to this new image file.

Lines 29 to 35 set the stage for this to happen.

```
36.  $file_name =  $_FILES["file"]["name"];
37.  $query = "INSERT INTO images (VIN, ImageFile)
VALUES ('$vin', '$file_name')";
38.  echo $query."<br>\n";
```

In **line 36** we get just the name of the uploaded file, without any path information at all. This is because we just want to insert the name of the file into the database. When referring to the file later with an <IMG> tag, we can always specify a path if needed.

Line 37 builds the query to insert the record into the database, and line 38 writes out what the query is. Line 38 was very helpful while I was originally writing this script, because of course it didn't work the first time I tried it. Seeing the actual query is the first step to figuring out why a particular query did not work.

```
39.  echo  "<a href='AddImage.php?VIN=";
40.  echo $vin;
41.  echo "'>Add another image for this car </a>
</p>\n";
```

Lines 39 to 41 create a link that allows us to easily add another image for this car if we have one.

```
42.  /* Try to insert the new car into the database */
43.  if ($result = $mysqli->query($query)) {
44.       echo "<p>You have successfully entered
$target_path into the database.</P>\n";
45.
46.   }
47.   else
48.   {
49.       echo "Error entering $VIN into database: " .
mysql_error()."<br>";
50.   }
51.   $mysqli->close();
```

Lines 42 to 51 execute the query and prints out either a success or failure message. Line 52 closes the connection to mySQL.

```
52.   echo "<img src='$imagename' width='150'>
<br>";
```

Line 52 creates an image tag for the file we just uploaded so you can see what it looks like. When I first created this the images were so big they took over the whole screen, so I added the attribute width='150' to keep the images to a reasonable size. This tells the browser to resize the image.

PHP Quirks and Tips

Introduction

Every language has its quirks. As I encounter those aspects of PHP that are not immediately intuitive, or if I find a great tip that could make your life easier, it will go into this section.

Single Quotes vs Double Quotes

When working with strings, it is important to understand the difference in how PHP treats single quotes (echo 'Hello $name';) as compared with double quotes (echo "Hello $name";)

Single quoted strings will display things exactly "as is." Variables will not be substituted for their values. The first example above (echo 'Hello $name';) will print out Hello $name.

Double quote strings will display a host of escaped characters and variables in the strings will be substituted for their values. The second example above (echo "Hello $name") will print out Hello Alan if the $name variable contains 'Alan'.

This is an easy thing to mix up, so read it again. :)

The Equal Sign

The equal sign can often be a source of confusion. A single equal sign is used to **assign** a value to a variable, for instance $FirstName = 'Alan';

The equal sign can also be used to **compare** to values, if you put two of them together and include it in an **if** statement. For instance, $FirstName == 'Alan' will return true for me, as the following code demonstrates

```
<?php
$FirstName = 'Alan';
if ($FirstName == "Alan")
{
echo "Hello Mr. Awesome";
}
else
{
echo "Hello $FirstName";
}
?>
```

See the sample code comparisons.php

The quirky thing about the double equal test is that PHP will attempt to convert the two variables being compared into different types to see if it gets a match. For instance, if $a

= 1 and $b = "1" you might think that they are not equal because they are different types. (One is a *number* and the other is a *string*.)

However, comparing $a and $b using the == comparison will return **true**, because if you convert $b from the type string to the type number the two variables are equal.

When you convert a variable from one type to another type, that's called 'Casting'. I could also have written that PHP will cast the string into a number, then find the two variables equal, but you have to admit that does sound awful geeky. But I mention it because you may well come across the term.

If you want to test if two values are the same value **and** the same type, you compare them using **three** equal signs. This way, $a === $b would return false.

```
<html>
  <body>
     <h1>Comparison Operators</h1>
     <?php
     $FirstName = 'Alan';
     if ($FirstName == "Alan")
     {
     echo "Hello Master";
     }
     else
     {
     echo "Hello $FirstName";
     }
     echo "<br>";
     $a = 1;
     $b = "1";
     if ($a == $b)
     {
     echo '$a is equal to $b';
     }
     else
     {
     echo '$a is not equal to $b';
     }

     echo "<br><BR>";
     if ($a === $b)
     {
     echo '$a is equal to $b';
     }
     else
     {
     echo '$a is not equal to $b';
     }
     ?>
</body>
</html>
```

17

Comparison Operators

Example	Name	Result
$a == $b	Equal	TRUE if *$a* is equal to *$b* after type juggling.
$a === $b	Identical	TRUE if *$a* is equal to *$b*, and they are of the same type.
$a != $b	Not equal	TRUE if *$a* is not equal to *$b* after type juggling.
$a <> $b	Not equal	TRUE if *$a* is not equal to *$b* after type juggling.
$a !== $b	Not identical	TRUE if *$a* is not equal to *$b*, or they are not of the same type.
$a < $b	Less than	TRUE if *$a* is strictly less than *$b*.
$a > $b	Greater than	TRUE if *$a* is strictly greater than *$b*.
$a <= $b	Less than or equal to	TRUE if *$a* is less than or equal to *$b*.
$a >= $b	Greater than or equal to	TRUE if *$a* is greater than or equal to *$b*.

If you compare a number with a string or if the comparison involves numerical strings, then each string is converted to a number and the comparison performed numerically. These rules also apply to the switch statement. The type conversion does not take place when the comparison is === or !== as this involves comparing the type as well as the value.

Security Considerations

Introduction

As we have seen, PHP is a very easy language to learn, and many people without any sort of formal background in programming will learn it as a way to add inter-activity to their web sites.

Unfortunately, that often means PHP programmers, especially those newer to web development, are unaware of the potential security risks their web applications can contain.

Security is something that is often overlooked when designing a web project, because there isn't really any "joy" in thinking about someone hacking into your shiny new application.

Security is a difficult thing to measure, and it is impossible to say whether an application is truly secure or not—there are only degrees of security. Naturally, the more effort you put into making an application secure, the more secure it will be. The trick, of course, is finding the right balance in time and effort—and expense.

It is fairly easy and relatively inexpensive to provide a sufficient level of security for most applications. However, if your security needs are very demanding—because the information stored in your application is very valuable (or very sensitive, like nuclear launch codes)—then you must ensure a higher level of security despite the increased costs that will be associated with it. Remember, a security breach can also be very expensive.

Balancing Security and Usability

Sadly, many of the steps taken to increase the security of a web application also decrease its usability. Passwords, session time-outs, and access control levels and roles all create

obstacles for legitimate users. While these steps will increase the security of the application, you can't have it so secure that nobody can use it.

I did a year-plus contract as a developer at an unnamed government agency that claimed to be very security conscious. They required a thorough background check prior to employment, and everyone had to wear high-tech badges to move about the building. We even had guards at the entrance to the building. It was "so secure" that we had to change our passwords every 30 days to a password we hadn't used before, and that password had to be at least 10 characters long and contain numbers, letters, mixed case, and punctuation marks—and it couldn't be found in the dictionary.

In short, they required passwords that no human could actually remember, and the system was not very usable. If your computer was idle for 15 minutes or more, you'd be prompted to type in the password in again. **Everyone** I worked with on that project had their password *written down on a piece of paper **right next to** their computer*. Clearly the "powers that be" in the security department had picked security over usability to such an extreme that the very security they were seeking was utterly compromised.

SQL Injection

One of PHP's greatest strengths is the ease with which it can communicate with databases, such as MySQL. The **Sam's Used Car Lot** example from this book and thousands of other high profile web sites, such as http://Facebook.com, rely on databases to function.

With that strength also comes risks. The most common security hazard faced when interacting with a database is something called SQL Injection - when a user deliberately uses part of your application to run unauthorized and unintended SQL queries on your database.

Let's use a common example. Although we didn't cover it in this book, many systems that ask a user to login feature a line of PHP code that looks a lot like this one:

```
$authorized = mysql_query("SELECT Username,
Password, UserLevel FROM Users WHERE Username =
'".$_POST['username']."' and Password =
'".$_POST['password']."'");
```

The script takes the username and password that was entered on the form and builds a query using the text entered by the user.

Does it look familiar? You'll see many variations of this as your journey into the Joy of PHP continues. So what's the problem? It does not look like such code could do much damage. But let's say for a moment that I enter the following into the "username" input box in the form and submit it:

'OR1=1#

The hash symbol (#) tells MySQL that everything following it is a comment and to ignore it.
The query that is going to be executed by mySQL will now look like this:

```
SELECT Username, Password FROM Users
     WHERE Username = '' OR 1=1 #' and
Password = ''
```

The # symbol tells mySQL to ignore any text that follows, leaving a WHERE statement of 'WHERE Username = '' OR 1=1'. Since 1 **always** equals 1, the WHERE clause of the SQL will match for **every** row—and here's the bad part. The query will return **all** of the usernames and passwords from the database. What may happen next is that if the first username and password combination is the admin user, then the person who simply entered a few symbols into a username box is now logged in as your website administrator, *as if* they actually knew the admin's username and password, which they

probably don't, and shouldn't know.

With a little creativity which is beyond the scope of this book, SQL Injection can be used to accomplish some nasty tricks you probably never thought of when designing your application.

Fortunately, it is pretty easy to put up roadblocks that help prevent this type of vulnerability. By checking for apostrophes in the items we enter into the database, and removing or substituting them, we can prevent anyone from running their own SQL code on our database.

The function below would do the trick:

```
function make_safe($variable) {
   $variable =
mysql_real_escape_string(trim($variable));
   return $variable;
}
```

Next we would need to modify our query. Instead of directly using the **_POST** variables, we would pass all user-provided data through the make_safe function, such as:

```
$username = make_safe($_POST['username']);
$password = make_safe($_POST['password']);
$authorized = mysql_query("SELECT Username,
Password, UserLevel FROM Users WHERE Username =
```

Now, if a user entered the malicious data above, the query will look like the following, which is perfectly harmless. The following query will select from a database where the username is equal to "' OR 1=1 #".

```
SELECT Username, Password, UserLevel
FROM Users
     WHERE Username = '\' OR 1=1 #' and
Password = ''
```

Now, unless you happen to have a user with a very unusual username and a blank password, your attacker will not be able to do any damage.

It is important to check **all** the data passed to your database like this, however secure you may think it is.

****