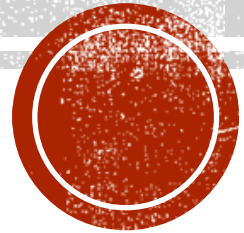
Programming with PHP & MySQL

II Year(CS) - SSCS3A



By

Dr.N.N.Krishna Veni,
Assistant Professor,
Holy Cross Home Science College,
Tuticorin

UNIT -1
INTRODUCTION

- 1.1 Introduction
- 1.2 History
- 1.3 Features
- 1.4 Variables
  - 1.4.1 Storing Values in Variables
  - 1.4.2 Assigning Variable Values
  - 1.4.3 Using Variable Values
  - 1.4.4 Saving Form Input in Variables
  - 1.4.5 Understanding Simple Data Types
  - 1.4.6 Detecting the Data Type of a Variable
  - 1.4.7 A Note on String Values
  - 1.4.8 A Note on Null Values

1.1 INTRODUCTION

- Widely used general purpose scripting language
- Used for the web development and can be embedded into HTML.
- The main goal of the language is to allow web developers to write dynamically generated web pages quickly.
- The only open-source server-side scripting language.
- PHP is available free of charge on the Internet.
- PHP support for the MySQL RDBMS, as well as other commercial database systems.

1.2 HISTORY

- The first version of PHP, PHP/FI, was developed by Rasmus Lerdorf in mid 1995. This version of PHP had support for some basic functions
- PHP/FI 1.0 was followed by PHP/FI 2.0 and in turn quickly updated in 1997 by PHP 3.0
- PHP 3.0 is developed by Andi Gutmans and Zeev Suraski.
  - Support for a wider range of databases, including MySQL and Oracle.
  - It has extensible architecture
  - It was installed on hundreds of thousands of web servers.

- PHP 4.0 was released in 2003, used a new engine to deliver better performance, greater reliability and scalability, support for web servers other than Apache.
- The current version of PHP, PHP 5.0, offers a completely revamped object model that uses object handles for more consistent behavior when passing objects around, as well as abstract classes, destructors, multiple interfaces, and class type hints.

1.3 FEATURES

- **Simplicity:** Easy syntax and clearly written manual, helps beginners find it easy to learn.
- **Portability:** Work on different platforms
- **Speed:** PHP scripts run faster than most other scripting languages.
- **Open Source:** PHP is an open source.
- **Extensible:** Enables developers to easily add support for new technologies
- **XML and Database Support:** PHP supports web application sources

1.4 VARIABLES

1.4.1 Storing Values in Variables

- Variables are the building blocks of any programming language to store both numeric and nonnumeric data.
- The contents of a variable can be altered during program execution, and variables can be compared and manipulated using operators.
- PHP supports a number of different variable types such as Booleans, integers, floating point numbers, strings, arrays, objects, resources, and NULLs.
- Every variable has a name, which is preceded by a dollar ($) symbol, and it must begin with a letter or underscore character, optionally followed by more letters, numbers, and underscores.

Example: $apple, $one_day.

1.4.2 Assigning Variable Values

- To assign a value to a variable, use the assignment operator (=).
- The value being assigned can be value or another variable or an expression.

Example: <?php $age = $dob + 15; ?>

1.4.3 Using Variable Values

To use a variable value in your script, simply call the variable by name, and PHP will substitute its value at run time.

Example:

```
<?php
    $a = "10";
    echo "The value of a is $a";
 ?>
```

1.4.4 Saving Form Input in Variables

Forms have always been one of the quickest and easiest ways to add interactivity to your web site.

Example:

Form Coding:

```
<html>
  <head></head>
    <body>
      <form action="display.php" method="post">
         Enter your message: <input type="text">
         <input type="submit" value="Send">
       </form>
    </body>
</html>
```

Here is what *display.php* looks like:

```
<?php
    // retrieve form data in a variable
    $a = $_POST['msg’];
    // print it
    echo "You said: <i>$a</i>";
?>
```

1.4.5 Understanding Simple Data Types

- PHP supports a wide variety of data types.
- PHP can *automagically* determine variable type by the context in which it is being used.

DATA TYPE	DESCRIPTION	EXAMPLE
Boolean	Boolean variable simply specifies a true or false value.	$a = true;
Integer	An integer is a plain-vanilla number like 75 or 95	$mark = 78;
Floating-point	A floating-point number is typically a fractional number such as 12.5 or 3.149391239129.	$average = 56.89;
String	A string is a sequence of characters. String values may be enclosed in either double quotes ("") or single quotes (').	$name = 'Harry';

1.4.6 Detecting the Data Type of a Variable

- To find out what type a particular variable is, PHP offers the gettype() function, which accepts a variable or value as argument.

Example:

```
<?php
    $a = true;                  // define variables
    $mark = 27;                 // define variables
    echo gettype($a);           // returns "boolean"
    echo gettype($mark);        // returns "integer"
?>
```

- PHP also supports a number of specialized functions to check if a variable or value belongs to a specific type

FUNCTION	WHAT IT DOES
is_bool()	Checks if a variable or value is Boolean
is_string()	Checks if a variable or value is a string
is_numeric()	Checks if a variable or value is a numeric string
is_float()	Checks if a variable or value is a floating point number
is_int()	Checks if a variable or value is an integer
is_null()	Checks if a variable or value is NULL
is_array()	Checks if a variable is an array
is_object()	Checks if a variable is an object

1.4.7 A Note on String Values

- String values enclosed in double quotes are automatically parsed for variable names; if variable names are found, they are automatically replaced with the appropriate variable value.

Example:

```
<?php
    $name = 'James';
    $car = 'BMW';
    $sentence = "$name drives a $car";  // This would contain the string "James drives a BMW"
?>
```

- **Note** that if your string contains quotes, carriage returns, or backslashes, it's necessary to escape these special characters with a backslash.

Example:

```
<?php
    $statement = 'It's hot outside';      // will cause an error due to mismatched quotes
    $statement = 'It\'s hot outside';     // will be fine
?>
```

1.4.8 A Note on NULL Values

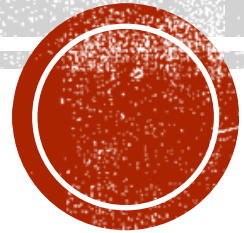
- The NULL data type is “special”: it means that a variable has no value.
- A NULL is typically seen when a variable is initialized but not yet assigned a value, or when a variable has been de-initialized with the unset() function.

Example:

```
<?php
    echo gettype($me);       // returns NULL
    $me = 'David';           // assign a value
    echo gettype($me);       // returns STRING David
?>
```

Programming with PHP & MySQL

II Year(CS) - SSCS3A



By

Dr.N.N.Krishna Veni,
Assistant Professor,
Holy Cross Home Science College,
Tuticorin

UNIT -1
INTRODUCTION

- 1.5 Statements And Comments
- 1.6 Operators
  - 1.6.1 Using Arithmetic Operators
  - 1.6.2 Using String Operators
  - 1.6.3 Using Comparison Operators
  - 1.6.4 The === Operator
  - 1.6.5 Using Logical Operators
  - 1.6.6 Using the Auto-Increment & Auto - Decrement Operators
  - 1.6.7 Understanding Operator Precedence

1.5 STATEMENTS & COMMENTS

- A PHP script consists of one or more *statements*, with each statement ending in a semicolon.

Example:

```
<?php
  // this is a single-line comment
 /* and this is a
    multiline
    comment */
?>
```

1.6 OPERATORS

=	Assignment
+	Addition
-	Subtraction
*	Multiplication
/	Division; returns quotient
%	Division; returns modulus
.	String concatenation
==	Equal to
===	Equal to and of the same type
!==	Not equal to or not of the same type
<>	aka != Not equal to

<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
&&	Logical AND
||	Logical OR
Xor	Logical XOR
!	Logical NOT
++	Addition by 1
--	Subtraction by 1

1.6.1 Using Arithmetic Operators: +, -, *, /, %

```
<?php
        $n1 = 101;
        $n2 = 5;
        $sum = $n1 + $n2;          // add
        $sub = $n1 - $n2;          // subtract
        $mul = $n1 * $n2;          // multiply
        $div = $n1 / $n2;          // divide
        $mod = $n1 % $n2;          // modulus
  ?>
```

To perform an arithmetic operation simultaneously with an assignment, use the two operators together.

```
<?php
    $a = $a + 10;
    $a += 10;
?>
```

1.6.2 Using String Operators: Period(.)

Example:

```
<?php
    $username = 'john';
    $domain = 'gmail.com';
    $email = $username . '@' . $domain;   // combine them using the concatenation operator
?>
```

Output: john@gmail.com

1.6.3 Using Comparison Operators

- To test whether two variables are different, use comparison operators. The result of a comparison test is always a Boolean value (either true or false).

Example:

```
<?php
    $a = 29;
     $b = 40;
     $result = ($a < $b);
     $result = ($a > $b);
     $result = ($a <= $b);
     $result = ($a >= $b);
     $result = ($a == $b);
     $result = ($a != $b);
     $result = ($a <> $b);
?>
```

1.6.4 The === Operator

- The === operator, enables you to test both for equality and type.

Example:

```
<?php
    $s = '14';
    $n = 14;
    $result = ($s == $n);      // returns false since the variables are not of the same type
    $result = ($s === $n);
?>
```

1.6.5 Using Logical Operators: (logical AND, logical OR, logical XOR, and logical NOT)

Example:

```
<?php
    $a = 'joe';
    $b = 'try3';
    $result = (($a == 'joe') && ($b == 'try3'));        //returns true
    $result = (($a == 'joe') || ($b == 'rar'));         //returns true
    $result = !($a == 'jeo');                           //returns false
    $result = (($a == 'joe') xor ($b == 'try3'));       //returns false
?>
```

1.6.6 Using the Auto-Increment and Auto-Decrement Operators

- The *auto-increment* operator is a PHP operator designed to automatically increment the value of the variable it is attached to by 1. It is represented by a double addition symbol (++).

Example:

```
<?php
    $total = 10;
    $total++;        // $total is now 11
?>
```

```
<?php
    $total = 10;
    $total--;        // $total is now 9
?>
```

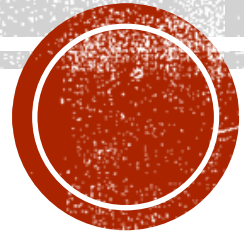
1.6.7 Understanding Operator Precedence

- When it comes to evaluating operators, the language has its own set of rules about which operators have precedence over others. Operators on the same line have the same level of precedence.

'!' '++' '--'
'*' '/' '%'
'+' '–' '.'
'<' '<=' '>' '>='
'==' '!=' '===' '!=='
'&&'
'||'
'?' ':'

Programming with PHP & MySQL

II Year(CS) - SSCS3A



By

Dr.N.N.Krishna Veni,
Assistant Professor,
Holy Cross Home Science College,
Tuticorin

UNIT -1
INTRODUCTION

- 1.7 Conditional Statements
  - 1.7.1 Using the If() Statement
  - 1.7.2 Using the Switch() Statement
  - 1.7.3 The Ternary Operator
  - 1.7.4 Nesting Conditional Statements
  - 1.7.5 Merging Forms and their Result Pages with Conditional Statements

1.7 CONDITIONAL STATEMENTS

- A conditional statement enables you to test whether a specific condition is true or false, and to perform different actions on the basis of the test result. PHP comes with two basic types of conditional statements,

1.7.1 Using the if() Statement

- If the statement evaluates to true, all PHP code within the curly braces is executed; if not, the code within the curly braces is skipped and the lines following the if() construct are executed.

Syntax: if()

```
<?php
   if (conditional test)
   {
     do this;
   }
?>
```

Syntax: if-else()

```
<?php
   if (conditional test)
   {
      do this;
   }
   else
   {
      do this;
   }
?>
```

Syntax: if-elseif-else()

```
<?php
   if (conditional test #1)
   {
      do this;
   }
   elseif (conditional test #2)
   {
     do this;
   }
   …
elseif (conditional test #n)
   {
      do this;
   }
  else
   {
do this;
   }
?>
```

1.7.2 Using the switch() Statement

- A switch() statement evaluates a conditional expression or decision variable; depending on the result of the evaluation, an appropriate case() block is executed. If no matches can be found, a default block is executed instead.

Syntax:

```
<?php
   switch (condition variable)
  {
    case possible result #1:
                              do this;
    case possible result #2:
                               do this;
              ...
    case possible result #n:
                             do this;
    case default;
              do this;
  }
?>
```

1.7.3 The Ternary Operator

- PHP's *ternary operator* is represented by a question mark (?). The ternary operator provides shortcut syntax for creating a single-statement if-else() block.

Example:

```
<?php
    if ($mark >= 35)
   {
       $msg = 'Pass';
    }
    else
   {
        $msg = 'Fail';
    }
?>
```

The above coding can be written as

```
<?php
      $msg = $mark > 10 ? 'Pass' : 'Fail';
?>
```

1.7.4 Nesting Conditional Statements

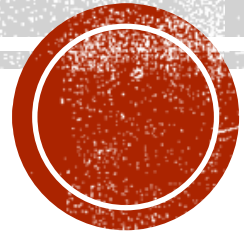
- To handle multiple conditions, you can “nest” conditional statements inside each other.

Example:

```
<?php
    if ($country == 'India')
   {
      if ($state == 'Maharashtra')
                {
                    if ($city == 'Bombay')
          {
              $home = true;
           }
       }
  }
?>
```

Programming with PHP & MySQL

II Year(CS) - SSCS3A



By

Dr.N.N.Krishna Veni,
Assistant Professor,
Holy Cross Home Science College,
Tuticorin

UNIT -1
INTRODUCTION

- 1.8 Loops
  - 1.8.1 Using the While() Loop
  - 1.8.2 Using the Do() Loop
  - 1.8.3 Using the For() Loop
  - 1.8.4 Controlling Loop Iteration with Break and Continue

1.8 LOOPS

- A *loop* is a control structure that enables you to repeat the same set of statements or commands over and over again; the actual number of repetitions may be dependent on a number you specify, or on the fulfillment of a certain condition or set of conditions.

1.8.1 Using the while() Loop

- The first and simplest loop to learn in PHP is the while() loop. With this loop type, so long as the conditional expression specified evaluates to true, the loop will continue to execute. When the condition becomes false, the loop will be broken and the statements following it will be executed.

Syntax:

```
<?php
   while (condition is true)
   {
     do this;
    }
?>
```

1.8.2 Using the do() Loop

- In do-while() loop the statements within the loop are executed first, and the condition is checked after.

Syntax:

```
<?php
   do
     {
         do this;
      } while (condition is true)
?>
```

1.8.3 Using the for() Loop

- The for() loop executes a certain set of statements a fixed number of times,

Syntax:

```
<?php
   for (initialize counter; conditional test; update counter)
  {
    do this;
  }
?>
```

1.8.4 Controlling Loop Iteration with break and continue

Break

The break keyword is used to exit a loop when it encounters an unexpected situation.

Example:

```
<?php
   for ($i=0; $i<=10; $i++)
   {
     if ($i == 8)
       {
          break;
        }
    echo $i;
   }
?>
```

Continue

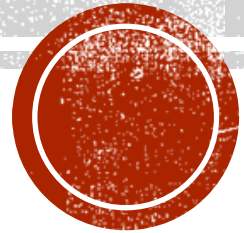
The continue keyword is used to skip a particular iteration of the loop and move to the next iteration immediately.

Example:

```
<?php
   for ($i=0; $i<=10; $i++)
   {
     if ($i == 8)
       {
          echo $i;
        }
     else
       {
          continue;
        }
   }
?>
```

Programming with PHP & MySQL

II Year(CS) - SSCS3A



By

Dr.N.N.Krishna Veni,
Assistant Professor,
Holy Cross Home Science College,
Tuticorin

UNIT -2
ARRAYS

- 2.1 Arrays
  - 2.1.1 Creating An Array
  - 2.1.2 Modifying Array Elements
  - 2.1.3 Processing Arrays with Loops
  - 2.1.4 The Foreach() Loop
  - 2.1.5 Grouping Form Selections with Arrays

2.1 ARRAYS

- An *array* is a complex variable that enables you to store multiple values in a single variable.

Example:

```
<?php
    $fruits = array('apple', 'banana', 'plum', 'grape');          // define an array
?>
```

- The array elements are accessed via an index number, with the first element starting at zero.
- To access the value *grape*, use the notation $fruits[3].

Hash or Associative Array:

- PHP also enables you to replace indices with user-defined "**keys**" to create a slightly different type of array.
- Keys may be made up of any string of characters, including control characters.

Example:

```
<?php
  // define associative array
   $fruits = array('red' => 'apple', 'yellow' => 'banana', 'purple' => 'plum', 'green' => 'grape');
?>
```

- To access the value *grape*, use the notation $fruits['green'].

2.1.1Creating an Array

- To define an array variable, name it using standard PHP variable naming rules and populate it with elements using the array() function.

Example:

```
<?php
    $fruits = array('apple', 'banana', 'plum', 'grape');          // define an array
?>
```

An alternative way to define an array

```
<?php
   $fruits[0] = 'apple';
   $ fruits[1] = 'banana';
   $ fruits[2] = 'plum';
   $ fruits[3] = 'grape';
?>
```

Creating an associative array

```
<?php
   $fruits['red'] = 'apple';
   $fruits['yellow'] = 'banana';
   $fruits['purple'] = 'plum';
   $fruits['green'] = 'grape';
?>
```

2.1.2 Modifying Array Elements

- To **add an element** to an array, assign a value using the next available index number or key:

```
<?php
    $fruits[4] = 'mango';          // add an element to a numeric array
    $fruits[] = 'mango';           // if you don't know the next available index
    $fruits['pink'] = 'peach';     // add an element to an associative array
?>
```

- To **modify an element** of an array, assign a new value to the corresponding scalar variable.

```
<?php
    $fruits[0] = 'blueberry';      // modify an array(apple to blueberry)
?>
```

- To **remove an array** element, use the array_pop() or array_push() function.

2.1.3 Processing Arrays with Loops

- To iteratively process the data in a PHP array, loop over it using any of the loop constructs.

Example:

```
<?php
    $list = array('rose', 'jasmine', 'lily');                    // define array
    for ($x = 0; $x < sizeof($list); $x++)                       // loop over it
    {
        echo "<li>$list[$x]";                                    // print array elements
    }
?>
```

- Here, the for() loop is used to iterate through the array, extract the elements from it using index notation, and display them one after the other in an unordered list.
- Note the sizeof() function is one of the most important and commonly used array functions, and it returns the size of (number of elements within) the array.

2.1.4 The foreach() Loop

- The new loop type introduced in PHP 4.0 for the purpose of iterating over an array is the foreach() loop.
- Unlike a for() loop, a foreach() loop doesn't need a counter or a call to sizeof(); it keeps track of its position in the array automatically.

foreach() loop with array

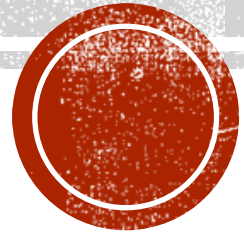
```
<?php
     // define array
    $list = array('rose', 'jasmine', 'lotus');
    foreach ($list as $item)   // loop over it
   {
        echo "<li>$item";
    }
?>
```

foreach() loop with the key-value pairs.

```
<?php
     // define associative array
    $flowers = array ('red'=>'rose', 'white'=>'jasmine', 'pink'=>'lotus');
   // iterate over it
    foreach ($flowers as $key => $value)
    {
            echo "<li>a $key named $value";
    }
?>
```

Programming with PHP & MySQL

II Year(CS) - SSCS3A



By

Dr.N.N.Krishna Veni,
Assistant Professor,
Holy Cross Home Science College,
Tuticorin

UNIT -2
ARRAYS

- 2.2 Functions
  - 2.2.1 Using Array Functions
  - 2.2.2 Creating User-Defined Functions
  - 2.2.3 Defining and Invoking Functions
  - 2.2.4 Using Arguments and Return Values
  - 2.2.5 Using Arrays with Argument Lists and Return Values
  - 2.2.6 Defining Global and Local Variables
  - 2.2.7 Importing Function Definitions

2.2 FUNCTIONS

2.2.1 Using Array Functions : the array_keys() and array_ values() functions

Example:

```
<?php
    $menu = array('breakfast' => 'idly', 'lunch' => 'rice', 'dinner' => 'dosa');
    $result = array_keys($menu);        // returns the array ('breakfast', 'lunch', 'dinner')
    $result = array_values($menu);   //returns the array ('idly', 'rice', 'dosa')
?>
```

- To check if a variable is an array, use the **is_array()** function, as in the following:

```
<?php
      $drinks = array('bovanto', 'pepsi');                    // create array
      echo is_array($drinks);                                 // returns 1 (true)
?>
```

- You can convert array elements into regular PHP variables with the **list()** and **extract()** functions.

Example:

```
<?php
        $fruits = array('apple', 'banana', 'plum', 'grape');        // define an array
         list ($fruits1, $fruits2, $fruits3) = $fruits;                    // extract values into variables
         echo $fruits1;                                                          // returns "apple"
?>
```

- The **extract()** function iterates through a hash, converting the key-value pairs into corresponding variable-value pairs.

Example:

```
<?php
        $fruits = array('red' =>'apple', 'yellow'=>'banana', 'purple'=>'plum', 'green'=>'grape');
        extract ($fruits);                    // extract values into variables
        echo $yellow;                        // returns "banana"
?>
```

- **array_push()** - To add an element to the end of an existing array.
- **array_pop()** – To remove an element from the end.
- **array_shift()** - To pop an element off the top of the array, you can use the array_shift() function.
- **array_unshift()** - To add elements to the beginning of the array.

Example:

```
<?php
    $students = array('Hari', 'Anu', 'Balu');                    // define array
     array_shift($students);              // remove an element from the beginning
     array_pop($students);                // remove an element from the end
     array_push($students, 'John');         // add an element to the end
     array_unshift($students, 'Ronald'); // add an element to the beginning
     print_r($students);                          // array now looks like ('Ronald', 'Anu', 'John')
?>
```

- The **explode()** function splits a string into smaller components on the basis of a user-specified pattern, and then returns these elements as an array.

Example:

```
<?php
    $string = 'English Latin Greek Spanish';          // define string
// split on whitespace $languages now contains ('English', 'Latin', 'Greek', 'Spanish')
    $languages = explode(' ', $string);
 ?>
```

- The **implode()** function creates a single string from all the elements of an array, joining them together with a userdefined separator. Revising the previous example, you have the following:

```
<?php
    $string = 'English Latin Greek Spanish';          // define string
    $languages = explode(' ', $string);               // split on whitespace
    $newString = implode(" and ", $languages); // returns "English and Latin and Greek and Spanish"
?>
```

2.2.2 Creating User-Defined Functions

- A *function* is simply a set of program statements that perform a specific task, and that can be called, or executed, from anywhere in your program.
- **Functions are a Good Thing for three important reasons:**
  - User-defined functions enable developers to extract commonly used pieces of code into separate packages, thereby **reducing unnecessary code repetition and redundancies**.
  - Because functions are defined once (but used many times), **they are easy to maintain.**
  - Because functions force developers to think in abstract terms, they encourage **better software design and help in creating extensible applications.**

2.2.3 Defining and Invoking Functions

- To understand how custom functions work, examine the following script:

```
<?php
    function display()                  // define a function
    {
        echo 'WELCOME';
    }
    display();                          // invoke a function
?>
```

2.2.4 Using Arguments and Return Values

- *What is Arguments, argument list, return value?*

Example:

```
<?php
    function add($m)                          // define a function with a single-argument list
    {
        echo "Value of m = "$m + 10. ";       // answer will be 60
    }
    add(50);                                  // invoke a function pass it a single argument
?>
```

Example:

```
<?php
    function add($a, $b)                              // define a function
    {
      $sum = $a + $b;
     return $sum;
    }
    echo 'The sum of a & b is ' add(10, 50);        // invoke a function
?>
```

2.2.5 Using Arrays with Arguments and Return Values

Example:

```
<?php
    function addDomain($u, $d)                    // define a function with a single-argument list
    {
        $result = array();                        // create empty result array
        foreach ($u as $element)
        {
            $result[] = $element . '@' . $d;
        }
        return $result;                                    // return result array
    }
    $users = array('john', 'jim', 'harry');                // define variables
    $newUsers = addDomain($users, 'gmail.com');
?>
```

2.2.6 Defining Global and Local Variables

```
<?php
    $x = 65;
    $y = 125;                          // define two variables
    function addx()                    // write a function that alters the global $x variable
   {
        global $x;
        $x = $x + 100;
   }

    function addy()                    // write a function that alters a local variable
  {
      $y = 2000;
  }
  echo "Initial value of x: $x";                  // returns 65
  addx();
  echo "Value of x after addx(): $x";             // returns 165
  echo "Initial value of y: $y";                  // returns 125
  addy();
  echo "Value of y after addy(): $y";             // returns 125
?>
```

2.2.7 Importing Function Definitions

- *the include()* and *require()* functions..

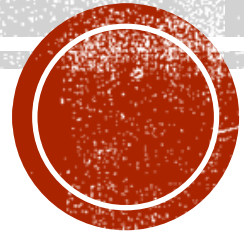
Example:

```
<?php
  include("/path/to/user/defined/functions.php");      // import file
// invoke functions here
?>
```

- The include() function generates a warning if the file cannot be found, although script processing continues.
- However, the require() function forces a file to be included in the script and generates a fatal error that stops script processing if the file cannot be found.

Programming with PHP & MySQL

II Year(CS) - SSCS3A



By

Dr.N.N.Krishna Veni,
Assistant Professor,
Holy Cross Home Science College,
Tuticorin

UNIT -2
ARRAYS

- 2.3 Using Files
  - 2.3.1 Reading and Writing Files
  - 2.3.1.1 Reading Data From a File
  - 2.3.1.2 Writing Data to a File
  - 2.3.2 Testing File Attributes
  - 2.3.3 Obtaining Directory Listings

2.3 USING FILES

2.3.1 Reading and Writing Files

2.3.1.1 Reading Data from a File

- To begin with, let's consider the process of opening a file and reading its contents.

Example:

```
<?php
    $file = '/home/web/projects.txt';                          // set file to read
    $fh = fopen($file, 'r') or die('Could not open file!');    // open file
    $data = fread($fh, filesize($file)) or die('Could not read file!');   //read file contents
    fclose($fh);                                               // close file
        echo $data;                                            // print file contents
?>
```

- The **three basic steps** to reading data from a file:
  - **Step 1:** Open the file(**fopen()**) and assign it to a file handle(**fh**). fopen() function, which accepts two arguments: the name and path to the file, and **mode** in which the file is to be opened ('r' for read).
  - **Step 2:** Interact with the file via its handle and extract its contents into a PHP variable. **fread**()
  - **Step 3:** Close the file. fclose().

Example:

```
<?php
    $file = '/home/web/projects.txt';                    // set file to read
    $data = file($file) or die('Could not read file!');  // read file into array
    foreach ($data as $line)                             // loop through array and print each line
    {
        echo $line;
    }
?>
```

- Another way to do this is with the file_get_contents() function, which reads the entire file into a string.

Example:

```
<?php
    $file = '/home/web/projects.txt';                    // set file to read
    $data = file_get_contents($file) or die('Could not read file!');  // read file into string
    echo $data;                                          // print contents
?>
```

2.3.1.2 Writing Data to a File

- The steps used for writing data to a file are same as reading a file but there are two differences:
  - You must fopen() the file in write mode ('w' for write).
  - Instead of using the fread() function to read from the file handle, use the fwrite() function to write to it.

Example:

```
<?php
    $file = '/tmp/dummy.txt';                                   // set file to write
    $fh = fopen($file, 'w') or die('Could not open file!');     // open file
    fwrite($fh, 'Hello, file!') or die('Could not write to file');   // write to file
    fclose($fh);                                                // close file
?>
```

- An alternative here is the file_put_contents() function, which takes a string and writes it to a file in a single line of code.

Example:

```
<?php
    $file = '/tmp/dump.txt';                                    // set file to write
    file_put_contents($file, 'Hello, file!') or die('Could not write to file’);   //write to file
?>
```

2.3.2 Testing File Attributes

FUNCTION	WHAT IT DOES
file_exists()	Returns a Boolean indicating whether the file exists
is_dir()	Returns a Boolean indicating whether the specified path is a directory
is_file()	Returns a Boolean indicating whether the specified file is a regular file
is_link()	Returns a Boolean indicating whether the specified file is a symbolic link
is_executable()	Returns a Boolean indicating whether the specified file is executable
is_readable()	Returns a Boolean indicating whether the specified file is readable
is_writable()	Returns a Boolean indicating whether the specified file is writable
filesize()	Gets file size, in bytes
filemtime()	Gets last modification time of fil
fileatime()	Gets last access time of file
fileowner()	Gets file owner
filegroup()	Gets file group
fileperms()	Gets file permissions
filetype()	Gets file type

2.3.3 Obtaining Directory Listings

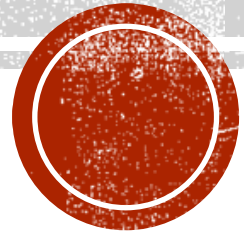
Example (which lists all the files in the directory /bin):

```
<?php
    $count = 0;                 // initialize counter
    $dir = "/bin";              // set directory name
    if (is_dir($dir))
   {
      if ($dh = opendir($dir)) // open directory and parse file list
     {
       while (($filename = readdir($dh)) !== false) // iterate over file list & print filenames
       {
           if (($filename != ".") && ($filename != ".."))
           {
             $count++;
             echo $dir . "/" . $filename . "\n";
           }
        }
      closedir($dh);            // close directory
     }
   }
echo "-- $count FILES FOUND --";
?>
```

- Here, the opendir() function first retrieves a handle to the named directory; this handle serves as the primary point of contact for all subsequent operations.
- The readdir() function then uses the file handle to read the contents of the directory, and return a list of file names one after another. Once the complete contents of the directory have been retrieved, readdir() returns a false value.
- The closedir() function is used to destroy the directory handle. Notice the manner in which entries for the current (.) and parent directory (..) are excluded from the list—with an if() conditional statement.

Programming with PHP & MySQL

II Year(CS) - SSCS3A



By

Dr.N.N.Krishna Veni,
Assistant Professor,
Holy Cross Home Science College,
Tuticorin

UNIT -2
ARRAYS

2.4 Managing Sessions and Using Session Variables

- 2.4.1 Creating a Session and Registering Session Variables
- 2.4.2 Destroying a Session

2.5 Cookies

- 2.5.1 Storing Data in Cookies
- 2.5.2 Setting Cookies
- 2.5.3 Retrieving Cookie Data
- 2.5.4 Deleting Cookies

2.6 Executing External Programs

2.4 MANAGING SESSIONS AND USING SESSION VARIABLES

- HTTP is the protocol on which the Web runs, is a "**stateless**" protocol.
- Consider, for example, the common **shopping cart** used in web storefronts.
- Consequently, what is required is a method that makes it possible to "**maintain state**," something that allows client connections to be tracked and connection-specific data to be maintained.
- A common solution to the problem is to use ***sessions*** to store information about each client and track its activities.
- This session data is preserved for the duration of the visit, and is usually destroyed on its conclusion.
- Client transactions are identified through unique numbers; these identifiers are used to re-create each client's prior session environment whenever required.
- The ***session identifier*** may be stored on the client in a **cookie** or it may be passed from page to page in the URL.

2.4.1 Creating a Session and Registering Session Variables

- The **session_start()** function is used to create a client session and generate a session ID.
- Any number of ***session variables*** can be registered.
- In a PHP script, session variables may be **registered as key-value** pairs in the special $_SESSION associative array.

Example:

```
<?php
    session_start();                                  // first page create a session
    $_SESSION['username'] = 'college';                // register some session variables
    $_SESSION['role'] = 'admin';
?>
```

- On subsequent pages, calls to the session_start() function re-create the prior session environment by restoring the values of the $_SESSION associative array.

Example:

```
<?php
    session_start();              // second page re-create the previous session
    echo $_SESSION['username']; // print the value of the session variable & returns
?>
```

2.4.2 Destroying a Session

To destroy an extant session use the **session_destroy()** function to erase session data.

Example:

```
<?php
    session_start();                    // re-create session
    $_SESSION = array();                // reset session array
    session_destroy();                          // destroy session
?>
```

2.5 COOKIES

2.5.1 Storing Data in Cookies

- Cookies allow web sites to store client-specific information in a file on the client system, and retrieve this information on demand.

Ground rules of cookies:

1. they can only be read by the site that created them.
2. single domain can set only 20 cookies, and each cookie is limited to a maximum size of 4KB.
3. A cookie usually possesses five types of attributes.
4. Of all the five attributes, only the first is not optional.

ATTRIBUTE	WHAT IT DOES
Name	Sets the name and value of the cookie
Expires	Sets the date and time at which the cookie expires
Path	Sets the top-level directory on the domain from which cookie data can be accessed
Domain	Sets the domain for which the cookie is valid
Secure	Sets a Boolean flag indicating that the cookie should be transmitted only over a secure HTTP connection

2.5.2 Setting Cookies

- The setcookie() function, which accepts **six arguments**: the cookie name, its value, its expiry date, its path and domain, and a Boolean flag indicating its security status.
- Only the **first argument is required**, others are optional.
- The setcookie() function **returns true** if successful. By checking for this, you can verify if the cookie was sent to the browser or not.

Example:

```
<?php
    // set a cookie called 'username' with value 'admin' & expiring after 1 day
    $ret = setcookie('username', 'admin', mktime()+86400, '/');
    if (!$ret)                                  // check if cookie was set & display error if not
    {
        echo "Unable to set cookie";
    }
?>
```

2.5.3 Retrieving Cookie Data

Aavailable in the $_COOKIE associative array, and its value may be **accessed using standard array notation**.

Example:

```
<?php
    if ($_COOKIE['username'])          // if cookie present, use it else display generic message
    {
        echo "Welcome back, " . $_COOKIE['username'];
    }
    else
    {
        echo "Is this your first time here? Take our guided tour!";
    }
?>
```

2.5.4 Deleting Cookies

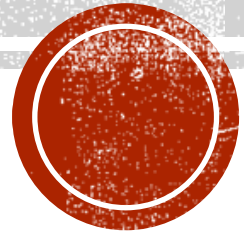
To delete a cookie, simply use setcookie() with its name to set the cookie's expiry date to a value in the past.

Example:

```
<?php
        setcookie('username', "NULL, mktime()-10000, '/');
 ?>
```

Programming with PHP & MySQL

II Year(CS) - SSCS3A



By

Dr.N.N.Krishna Veni,
Assistant Professor,
Holy Cross Home Science College,
Tuticorin

UNIT -3
FILE HANDLING

3.1 Opening Files Using Fopen

3.2 Looping Over a File’s Content with Feof

3.3 Reading Text From a File Using Fgets

3.4 Closing a File

3.1 OPENING FILES USING FOPEN

To start working with a file in PHP, you must first open that file,

Syntax:

$filehandle = fopen (filename, mode[, use_include_path [, zcontext]])

Here,

- Filename is the name of the file to open
- Mode indicates how you want to open the file(Table 3.1)
- use_include_path may be set to 1 or TRUE to specify that you want to search for the file in the PHP include path
- zcontext holds an optional file context(contexts modify or enhance the behavior of the data streams from and to files).

MODES	OPERATIONS
'r'	Open for reading only.
'r+'	Open for reading and writing.
'w'	Open for writing only and truncate the file to zero length. If the file does not exist, attempt to create it.
'w+'	Open for reading and writing and truncate the file to zero length. If the file does not exist, attempt to create it
'a'	Open for appending only. If the file does not exist, attempt to create it.
'a+'	Open for reading and writing, starting at the end of the file. If the file does not exist, attempt to create it.
'x'	Create an open for writing only. If the file already exits, the fopen call will fail by returning FALSE.
'x+'	Create an open for reading and writing. If the file already exits, the fopen call will fail by returning FALSE.

Example to open a file for reading:

$handle = fopen (“ /home /file.txt”, “r”);

Example to open a file for writing:

$handle = fopen (“ /home /file.txt”, “w”);

Example to open a file for binary writing:

$handle = fopen (“ /home /file.txt”, “wb”);

Example to open a file for reading:

$handle = fopen (“ /home /file.txt”, “r”);

Example to open URLs on the internet:

$handle = fopen (“ http://www.php.net”, “r”);

Example to escape any backslashes

$handle = fopen (“c: \\data\\file.txt”, “r”);

Example to open a file from different website:

$handle = fopen (“http : //www.superduperbigco.co/file.txt”, “r”);

Example to open files using the FTP protocols:

$handle = fopen (“ftp : //user:booking.com / file.txt”, “w”);

Example to check whether the file is open or not:

Say that you have a file, file.txt, with below content

```
Here
Is
Your
Data.
```

If the open operation fails, fopen returns FALSE. You can check if the file was opened as shown below:

```
<?php
    $handle=fopen (“file.txt”, “r”);
    If ($handle)
    {
        Echo “File Opened OK”;
    }
?>
```

3.2 LOOPING OVER A FILE'S CONTENT WITH FEOF

If there are multiple lines in a file and you want to read those lines, one line at a time, loop over all the lines in the file using a while loop and using feof function.

Example:

```
<?php
        $handle = fopen ("file.txt", "r");
        While (!feof($handle))
        {
            ….
            ….
            ….
        }
?>
```

3.3 READING TEXT FROM A FILE USING FGETS:

You can use fgets function to get a string from a file.

Syntax:

fgets (handle [, length])

This function returns a string of up to length -1 bytes read from the file corresponding to the file handle. If no length is specified, the length defaults to 1024bytes.

Example:

Here's how you might read a line of text from the file,file.txt and display the text:

```
<?php
    $handle=fopen("file.txt", "r");
    While( !feof($handle))
    {
        $text=fgets($handle);          //Read the file
        echo $text, "<br>";            //Print the file
    }
?>
```

3.4 CLOSING A FILE

Closing the file frees up the resources connected with that file, and avoids conflicts later in your code in case you recycle file handle variables.

Syntax:

fclose($file handle);

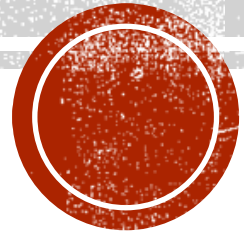
This function returns TRUE if the file was closed successfully, and FALSE otherwise.

Example:

```
<?php
    $handle=fopen("file.txt", "r");
    While( !feof($handle))
    {
        $text=fgets($handle);
        echo $text, "<BR>";
     }
    fclose($handle);                    //Closing the file
?>
```

Programming with PHP & MySQL

II Year(CS) - SSCS3A



By

Dr.N.N.Krishna Veni,
Assistant Professor,
Holy Cross Home Science College,
Tuticorin

UNIT -3
FILE HANDLING

3.5 Reading From a File Character with Fgetc

3.6 Reading a Whole File at Once with File_get_contents

3.7 Reading a File Into an array with File

3.5 READING FROM A FILE CHARACTER WITH FGETC

You can read individual characters from a text file using the fgetc function:

Syntax:

fgetc ($ filehandle);

Example:

```
<?php
   $handle = fopen("file.txt", "r");      //Open the file
   $char = fgetc ($handle)                // Read an individual character from file.txt
?>
```

To loop over all the characters in the file, you can use a while loop statement which ends when fgetc returns FALSE, means there are no more characters to read:

Example:

```
<?php
    $handle = fopen("file.txt", "r");                    //Open the file
    While ($char = fgetc($handle))                       // Loop over the characters
    {
          Echo "$char";                                        //Print the characters
     }
?>
```

The result of the example is : Here is your data

But the actual content of the file,file.txt, is:

```
Here
Is
Your
Data.
```

To display the output as it is in file we have to use the newline characters.

Example:

```
<?php
    $handle = fopen ("file.txt", "r");          //Open a file
    While ($char = fgetc($handle))              // Loop over the characters
    {
        if ($char == "\n")                      //Check for the new line
        {
            $char = "<br>";
        }
      Echo "$char";                            //Print the character
    }
Fclose($handle);                                //Close the file
?>
```

3.6 READING A WHOLE FILE AT ONCE WITH FILE_GET_CONTENTS

You can read the entire contents of a file with the file_get_contents function:

Syntax:

File_get_contents (filename [,use_include_path [, context [, offset [, maxlen]]]])

Here,

- filename is the name of the file.
- use_include_path is set to TRUE if you want to search PHP's include path.
- context is a context for the operation.
- offset is the offset into the file at which to start reading.
- maxlen is the maximum length of data to read.

Example:

```
<?php
        //Read the entire contents of the file file.txt into the variable $text
        $text=file_get_contents(file.txt");

        // Converts all newlines into <br> elements using the PHP function str_replace
        $fixed_text=str_replace("\n", "<br>", $text);

        // The converted text is echoed to the browser
        Echo $fixed_text;
?>
```

3.7 READING A FILE INTO AN ARRAY WITH FILE

Use the file function to read a file into an array all at once; each line becomes an element in the array.

Syntax:

file (filename [, use_include_path_path [, context]])

Here,

- Filename is the name of the file you want to read,
- use_include_path dhould be set to TRUE if you want to search the PHP include path for the file,
- Context for the operation.

This function returns an array or FALSE if the operation failed. This function is useful if you want to write your own database files.

Example:

Here's an example, which reads file.txt into an array, $data.

```
<?php
    $data=file('file.txt');              // load the contents of the file into the array
    foreach  ($data as $line)            //read each line from $data
    {
        echo  $line  ,  "<br>";
    }
?>
```

You can also display line numbers. Here's how that working :

```
<?php
    $data=file('file.txt');
        foreach  ($data as $number =>  $line)
        {
                echo   "Line   $number:  "  , $line ,   "<br>";
        }
?>
```

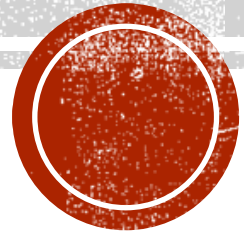
You can also open Web pages and read them into arrays using the file function.

```
<?php
    $data=file('http://www.php.net');          //reading web page into $data
    foreach  ($data as $number =>  $line)
    {
            echo   "Line   $number:  "  , $line ,   "<br>";
    }
 ?>
```

Each element in the array still has a new line character at the end of it. If you want to get rid of that new line character, use can use the rtrim PHP function.

Programming with PHP & MySQL

II Year(CS) - SSCS3A



By

Dr.N.N.Krishna Veni,
Assistant Professor,
Holy Cross Home Science College,
Tuticorin

UNIT -3
FILE HANDLING

- 3.8 PARSING FILES WITH fscanf
- 3.9 PARSING INI FILES WITH parse_ini_file
- 3.11 GETTING FILE INFO WITH stat
- 3.12 SETTING THE FILE POINTER'S LOCATION WITH fseek
- 3.13 COPYING FILE WITH copy
- 3.14 DELETING FILES WITH unlink

3.8 PARSING FILES WITH FSCANF

You can also parse files with the fscanf function:

Syntax:

fscanf (handle, format)

Example:

File Name: students.txt

File Content: Arun Kumar
Hari Bala
Jeya Ganesh
Siva Kumar

```
<?php
    $handle= fopen("students.txt","r");                    // Open the file
   while ($name=fscanf($handle, "%s\t%s\n"))               // Read and  parse a line
   {
       list ($firstname, $lastname)=$name;                 // Assign the values
       echo $firstname, " ", $lastname,"<br>";             //Display the names
    }
   fclose($handle);                                                // Close the file
?>
```

3.9 PARSING INI FILES WITH PARSE_INI_FILE

Much like fscanf, parse_ini_files lets you parse files.

Syntax:

parse_ini_file (filename [, process_sections])

If process_Section parameter to TRUE , you get a multidimensional array, The default is FALSE.

Example:

```
//This is a sample .ini files
[first_section]
    First_color = red
    Second _color = white
    Third_color = blue
[second_section]
    ----------
    ----------
```

PHP Coding:

```
<? Php
        // read the contents of sample.ini into an array , $array:
        $array = parse_ini_file ("sample.ini ");
        .......
        .......
    ?>
```

3.11 GETTING FILE INFO WITH STAT

Syntax:

stat(filename)

NUMERICAL INDEX	TEXT KEYS	DESCRIPTIONS
0	dev	Device number
1	ino	Inode number
2	mode	Inode number protection mode
3	nlink	Number of links
4	uid	User id of owner
5	gid	Group id of owner
6	rdev	Device type, if inode device
7	size	Size in bytes
8	atime	Time of last access (Unix timestamp)
9	mtime	Time of last modification (Unix Timestamp)
10	ctime	Time of last inode change (Unix timestamp)
11	blsize	Block size of filesystem I/O
12	blocks	Number of blocks allocated

3.12 SETTING THE FILE POINTER'S LOCATION WITH FSEEK

PHP uses file pointers to keep track of where it is in a file, and where the next read or write operation occurs from.

Syntax

fseek (handle, offset, [start_point])

Here,

- handle is the handle of the file to set the file pointer in.
- offset is the number of bytes you want to set the pointer to.
- start_point indicates a starting point for the pointer, Which is one of these constant:

- **SEEK_SET** The beginning of the file
- **SEEK_CUR** The current pointer location
- **SEEK_END** The end of the file

You can set the offset to negative values.

3.13 COPYING FILE WITH COPY

Syntax:

copy (source, destination**)**

Here,

- source is the name of the source file,
- destination is the name of the copy (including pathnames, if applicable).
- This function returns TRUE if it was successful, FALSE otherwise.

Example:

```
<?php
    $file = ‘file.txt’;
   $copy = ‘copy.txt’;
  if (copy($file, $copy))
  {
    echo “Copied $file.”;
   }
  else
  {
    echo “Could not copy $file.”;
  }
?>
```

3.14 DELETING FILES WITH UNLINK

Syntax:

Unlink (filename [, context])

Here,

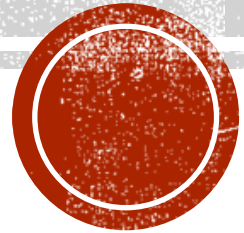
- filename is the name of the file, and
- context is an optional context.
- This function returns TRUE if the file was deleted, FALSE otherwise.

Example:

```
<?php
        if(unlink("copy.txt"))
        {
            echo "Deleted the file.";
        }
        else
        {
            echo "could not delete the file.";
        }
?>
```

Programming with PHP & MySQL

II Year(CS) - SSCS3A



By

Dr.N.N.Krishna Veni,
Assistant Professor,
Holy Cross Home Science College,
Tuticorin

UNIT -3
FILE HANDLING

- 3.14 WRITING TO A FILE WITH fwrite
- 3.15 READING AND WRITING BINARY FILES

3.14 WRITING TO A FILE WITH FWRITE

Syntax:

fwrite (handle, string [, length]) **// returns the number of bytes written, or FALSE if there was an error.**

Example:

```
<?php
    $handle = fopen("data.txt", "w");
    $text = "Here \n is\n the\n text.";
   fwrite($handle, $text);
?>
```

- To check whether the write operation failed:

```
<?php
    $handle = fopen ("data.txt", "w");
    $text = "Here \n is \n the \n text.";
     if (fwrite($handle, $text) == FALSE)
    {
        echo " Cannot write data.txt.";
     }
?>
```

- When the operation was successful, you can indicate that to the user like this:

```
<?php
$handle = fopen("data.txt", "w");
$text = "Here \n is \n the \n text.";
if (fwrite($handle, $text) == FALSE)
{
  echo "Cannot write data.txt.";
}
 else
 {
    echo "Created data.txt.";
 }
fclose($handle);
?>
```

3.15 READING AND WRITING BINARY FILES

You can pack binary data into strings using the pack function, and unpack data using the unpack function.

Example:

```
<?php
   $number  =  512;
   $handle  =  fopen  (“data.dat”,  “wb”);    //Writes the number 512 to a file in binary format
   pack  (“L”,   $number);                    //Packs the data into long integer format
 ?>
```

Here are the formats for the pack function:

- **a** - NUL-padded string
- **A** - SPACE-padded string
- **h** - Hex string, low nibble first
- **H** - Hex string, high nibble first
- **c** - Signed char
- **C** - Unsigned char

3.15 READING AND WRITING BINARY FILES

- **s** - Signed short (always 16 bit, machine byte order)
- **S** - Unsigned short (always 16 bit, machine byte order)
- **n** - Unsigned short (always 16 bit, big endian byte order)
- **v** - Unsigned short(always 16 bit, little endian byte order)
- **i** - Signed integer(machine –dependent size and byte order)
- **I** - Unsigned integer(machine –dependent size and byte order)
- **l** - Signed long(always 32 bit, machine byte order)
- **L** - Unsigned long(always 32 bit, machine byte order)
- **N** - Unsigned long(always 32 bit,big endian byte order)
- **V** - Unsigned long(always 32 bit, little endian byte order)
- **f** - Float(machine –dependent size and representation)
- **d** - Double(machine –dependent size and representation)
- **x** - NUL byte
- **X** - Back up one byte
- **@** - NUL-fill to absolute position

3.15 READING AND WRITING BINARY FILES

Example:

```
<?php
        $number = 512;
        $handle = fopen ("data.dat", "wb");
 if (fwrite ($handle, pack ("L", $number)) == FALSE)    //Writes data to a file
        {
                echo "cannot write data.dat.";
        }
        else
        {
                echo "created data.dat. and stored $number.";
        }
       fclose($handle);                                                    //Close the file
 ?>
```

3.15 READING AND WRITING BINARY FILES

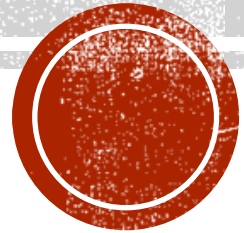
To read and display the binary data from the files.

- **Step1:** You open the file for binary reading.
- **Step2**: Use fread to read the binary data, indicating that you want four bytes(the length of a long integer
- **Step3:** Use unpack function to unpack the data into an array with an element under the index “data” containing a long value.
- **Step4:** Recover the binary data from the array using the key “data”.
- **Step5:** Display that data.

```
<?php
    $handle = fopen(“data.dat”,”rb”);          //Open the file
    $data = fread($handle,4);                  //Read the binary data
    $array = unpack(“Ldata”,$data);            //Unpack the data
    $data = $array[“data”];                    //Recover & store the binary data from array.
   echo ”Read this value from data.dat:”,$data;        // Display that data
?>
```

Programming with PHP & MySQL

II Year(CS) - SSCS3A



By

Dr.N.N.Krishna Veni,
Assistant Professor,
Holy Cross Home Science College,
Tuticorin

UNIT -3
FILE HANDLING

- 3.16 LOCKING FILES

3.16 LOCKING FILES

Syntax:

Flock (handle, operation [, &wouldblock])

Here,

- handle is the handle of the file you want to lock,
- operation is one of these:
  - LOCK_SH - To acquire a shared lock(reader).
  - LOCK_EX - To acquire an exclusive lock(writer).
  - LOCK_UN - To release a lock(shared or exclusive).
- optional third argument is set to TRUE if the lock would block.

This function return TRUE if it got a lock, FLASE otherwise.

3.16 LOCKING FILES

Example:

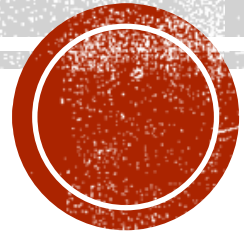
```
<?php
        $handle =fopen("data.txt","w");                    //Open the file
        $text = "Here\nis\nthe\ntext.";
 If (flock($handle, LOCK_EX | LOCK_NB))     //Lock the file
{
          echo "Locked the file.<br>";
        if (fwrite ($handle, $text) == FALSE)
        {
              echo "Cannot write data.txt. <br>";
        }
    else
    {
          echo"Created data.txt. <br>";
     }
     flock ($handle , LOCK_UN);                    //Unlock the file
   echo "Unlocked the files.  <br>";
  }
 ?>
```

If you couldn’t get the lock, other code is using the file, and you should let the user know as in below example

```
<?php
        $handle=fopen(“data.txt”,”w”);
        $text=”Here\nis\nthe\ntext.”;
        If  (flock ($handle, LOCK_EX | LOCK_NB) )
        {
            echo “Locked the file.<br>”;
                if (fwrite ($handle, $text) == FALSE)
            {
                echo “Cannot write data.txt. <br>”;
            }
                else
                {
                echo “Created data.txt. <br>”;
                }
            flock ($handle , LOCK_UN);
                echo ”Unlocked the files.  <br>”;
    }
    else
    {
                echo”could not lock the file.<br>”;
    }
fclose($handle);
?>
```

Programming with PHP & MySQL

II Year(CS) - SSCS3A



By

Dr.N.N.Krishna Veni,
Assistant Professor,
Holy Cross Home Science College,
Tuticorin

UNIT - 4
MYSQL

4.1 EFFECTIVENESS OF MYSQL

4.2 MYSQL TOOLS

4.3 PREREQUISITES FOR MYSQL CONNECTION

4.4 DATABASES AND TABLES

4.4.1 MySQL data types

4.1 EFFECTIVENESS OF MYSQL

MySQL has been around for a long time, and is now installed and in use at millions of installations worldwide. Why do so many organizations and developers use MySQL? Here are some of the reasons:

- **Cost**: MySQL is open-source, and is usually free to use (and even modify) the software without paying for it.
- **Performance**: MySQL is fast (make that very fast).
- **Trusted**: MySQL is used by some of the most important and prestigious organizations and sites, all of whom entrust it with their critical data.
- **Simplicity**: MySQL is easy to install and get up and running.

4.2 MYSQL TOOLS

Mysql: Command-Line Utility : Every MySQL installation comes with a simple command-line utility called mysql.

MySQL Administrator:

- **Server Information** displays status and version information about the connected server and client.
- **Service Control** allows you to stop and start MySQL as well as specify server features.
- **User Administration** is used to define MySQL users, logins, and privileges.
- **Catalogs lists** available databases and allows for the creation of databases and tables.

MySQL Query Browser: MySQL Query Browser is a graphical interactive client used to write and execute MySQL commands.

4.3 PREREQUISITES FOR MYSQL CONNECTION

- MySQL, like all client-server DBMSs, requires that you log in to the DBMS before being able to issue commands. Login names might not be the same as your network login name; MySQL maintains its own list of users internally, and associates rights with each.

- When you first installed MySQL, you were probably prompted for an administrative login (often named root) and a password. If you are using your own local server and are simply experimenting with MySQL, using this login is fine.

- In the real world, however, the administrative login is closely protected (as access to it grants full rights to create tables, drop entire databases, change logins and passwords, and more).

To connect to MySQL you need the following pieces of information:

- The hostname (the name of the computer)this is localhost if connecting to a local MySQL server
- The port (if a port other than the default 3306 is used)
- A valid user name
- The user password (if required)

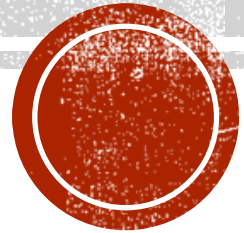
4.4 DATABASES AND TABLES

4.4.1 MySQL data types

DATA TYPES	USED FOR
TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT	Integer values
FLOAT	Single-precision floating-point values
DOUBLE	Double-precision floating-point values
DECIMAL	Decimal values
CHAR	Fixed-length strings up to 255 characters
VARCHAR	Variable-length strings up to 255 characters
TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB	Large blocks of binary data
TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT	Longer blocks of text data
DATE	Date values
TIME	Time values or durations
YEAR	Year values
DATETIME	Combined date and time values
TIMESTAMP	Timestamps
ENUM	Fields that must contain one of a set of predefined mutually exclusive values
SET	Fields that can contain zero, one, or more of a set predefined values9-

Programming with PHP & MySQL

II Year(CS) - SSCS3A



By

Dr.N.N.Krishna Veni,
Assistant Professor,
Holy Cross Home Science College,
Tuticorin

UNIT - 4
MYSQL

4.4.2 CREATING AND MANIPULATING TABLES

4.4.2.1 CREATING DATABASE

4.4.2.2 CREATING TABLES

4.4.2.3 ALTERING TABLES

4.4.2.4 DROPPING DATABASES AND TABLES

4.4.2 CREATING AND MANIPULATING TABLES

4.4.2.1 CREATING DATABASE

- To create a database use the CREATE DATABASE command, which creates an empty database.
- Database names cannot exceed 64 characters and names that contain special characters or consist entirely of digits or reserved words must be quoted with the backtick (`) operator.

➢ **Example:** mysql> CREATE DATABASE db;

- To select a particular database we use the USE command. Once you select a database with the USE command, it becomes the default database for all operations.

➢ **Example: mysql> USE db;**

4.4.2 CREATING AND MANIPULATING TABLES

4.4.2.2 CREATING TABLES

Example: mysql> CREATE TABLE stud
(-> regno int(10) UNSIGNED NOT NULL,
-> name varchar(255) NOT NULL default '',
-> year year(4) NOT NULL default '0000',
-> PRIMARY KEY (regno)
->) TYPE=MyISAM;

Adding field modifiers and keys:

- NULL and NOTNULL – To Specify whether the field is allowed to be empty
- DEFAULT - To specify a default value for a field.
- AUTO_INCREMENT – To have MySQL automatically generate a number for a field.
- CHARACTER SET - Can be used for fields that accept string values.
- INDEX – Used to index a field.
- UNIQUE – To specify that values entered into a field must be either unique or NULL.
- PRIMARY KEY – To specify a primary key for the table.
- FOREIGN KEY - To specify a foreign key for a table.

4.4.2 CREATING AND MANIPULATING TABLES

Selecting a table type: A number of table types are available, each with different advantages. Here is a list:

- **MyISAM -** The MyISAM format is optimized for **speed and reliability**, it supports tables in excess of 4GB in size, and it can be compressed to save space. It is the default table type

- **InnoDB -** It is the most **sophisticated table type** available in MySQL. It supports transactions and foreign keys, and allows multiple simultaneous users to execute SELECT statements; this improves performance and query response times. InnoDB tables are fully portable between different operating systems, and include crash recovery features to avoid data corruption or loss.

- **HEAP -** A HEAP table is stored in memory, making it **extremely fast**. This format is optimized for temporary tables and it is rarely used for other purposes. This is because the data in a HEAP table is available only while the server is running, and is automatically erased when the server shuts down and the memory is flushed.

4.4.2 CREATING AND MANIPULATING TABLES

- **BerkeleyDB -** The BerkeleyDB format is one of the more advanced table formats supported by MySQL. It supports transactions, checkpoints, crash recovery, and page-level locking. However, it also has certain disadvantages: BerkeleyDB tables are not easily portable between different operating systems and less memory efficient.

- **MERGE -** The MERGE table format makes it possible for a collection of MyISAM tables to be treated as one, by combining them into a single “virtual” table. This table format makes improving performance or increasing query efficiency possible in certain situations; however, it can only be used for tables that are completely identical in their internal structure.

- **ISAM -** The ISAM format is primarily offered for compatibility with older MySQL tables. It lacks many of the features of the MyISAM format, cannot handle large tables, and is more prone to fragmentation.

4.4.2 CREATING AND MANIPULATING TABLES

- **4.4.2.3 ALTERING TABLE**

To alter the tables the ALTER TABLE command is used. It allows us to add or delete fields, alter field types; add, remove, or modify keys; alter the table type; and change the table name. This includes the following

- Altering Table and Field Names
- Altering Field Properties
- Adding and Removing Fields and Keys
- Altering Table Types

Table Name : stud

ID	NAME	M1	M2
111	AAA	7	4
222	BBB	5	6
333	CCC	2	9

Altering table and field names: To alter a table name, use an ALTER TABLE command with a supplementary RENAME clause.

Example: mysql> ALTER TABLE stud RENAME TO student;

(OR)

mysql> RENAME TABLE stud TO student;

4.4.2 CREATING AND MANIPULATING TABLES

We can uses the ALTER TABLE command with a CHANGE clause to modify the name of field from address to address1.

Example: mysql> ALTER TABLE stud CHANGE m1 mark1 INT(5);

Output:

ID	NAME	Mark1	M2
111	AAA	7	4
222	BBB	5	6
333	CCC	2	9

Altering field properties: You can use the CHANGE clause to alter a field's type and properties as well. When you CHANGE a field from one type to another, MySQL will automatically attempt to convert the data in that field to the new type.

Example: mysql> ALTER TABLE stud CHANGE name age TINYINT(2);

Here in the above example we changed the field "name" defined as VARCHAR(30) to a field "age" with definition TINYINT(2).

4.4.2 CREATING AND MANIPULATING TABLES

Adding and removing fields and keys: We can add a new field to a table by including an ADD clause in your ALTER TABLE command.

Example: mysql> ALTER TABLE stud ADD m3 INT(5) NOT NULL.

Output:

ID	NAME	M1	M2	M3
111	AAA	7	4	0
222	BBB	5	6	0
333	CCC	2	9	0

Deleting: To delete an existing field from a table use a DROP clause instead of an ADD clause.

Example: mysql> ALTER TABLE stud DROP m3;

Output:

ID	NAME	M1	M2
111	AAA	7	4
222	BBB	5	6
333	CCC	2	9

- Delete a table's primary key with the DROP PRIMARY KEY clause.
  Example: mysql> ALTER TABLE stud DROP PRIMARY KEY;

- Add a new primary key with the ADD PRIMARY KEY clause.
  Example: mysql> ALTER TABLE stud ADD PRIMARY KEY (id);

4.4.2 CREATING AND MANIPULATING TABLES

Altering table types: You can alter the table type by adding a TYPE clause to the ALTER TABLE command.

Example: mysql> ALTER TABLE stud TYPE = INNODB;

4.4.2.4 DROPPING DATABASES AND TABLE

- To delete a database, use the DROP DATABASE command, which deletes the named database and all its tables permanently.
- Similarly, you can delete a table with the DROP TABLE command. Try this out by creating and dropping a table.

Example: mysql> DROP DATABASE db;

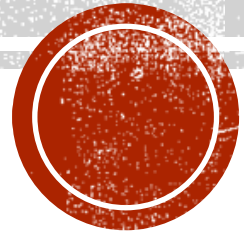
mysql> DROP TABLE stud;

- If what you want to empty the table of all records, use the TRUNCATE TABLE command instead, which internally DROP-s the table, and then re-creates it.

Example: mysql> TRUNCATE TABLE stud;

Programming with PHP & MySQL

II Year(CS) - SSCS3A



By

Dr.N.N.Krishna Veni,
Assistant Professor,
Holy Cross Home Science College,
Tuticorin

UNIT - 4
MYSQL

4.4.2 Creating And Manipulating Tables

4.4.2.5 Viewing Database, Table, And Field Information

4.4.3 Insertion, Updation And Deletion Of Rows In Tables

4.4.3.1 Inserting Records

4.4.3.2 Deleting Records

4.4.3.3 Updating Records

4.4.4 Retrieving Data

4.4.2 CREATING AND MANIPULATING TABLES

4.4.2.5 VIEWING DATABASE, TABLE, AND FIELD INFORMATION

You can view all available databases with the SHOW DATABASES command.

Example: mysql> SHOW DATABASES;

You can view available tables in a database with the SHOW TABLES command.

Example: mysql> SHOW TABLES FROM db;

To see the structure of a particular table, use the DESCRIBE command.

Example: mysql> DESCRIBE stud;

4.4.3 INSERTION, UPDATION AND DELETION OF ROWS IN TABLES

INSERTION, UPDATION AND DELETION OF ROWS IN TABLES

Table Name: movies

ID	TITLE	YEAR
1	AAA	1954
2	BBB	1959
3	CCC	1970
4	DDD	1975

4.4.3.1 INSERTING RECORDS

//Inserting single value

Example: mysql> INSERT INTO movies (title, year) VALUES ('AAA', 1954);

//Inserting single value but should be in order

Example: mysql> INSERT INTO movies VALUES (NULL, 'AAA', 1954);

//Inserting multiple values

Example: mysql> INSERT INTO movies (title, year) VALUES ('AAA', 1954),
('BBB', 1955), ('CCC', 1941);

4.4.3 INSERTION, UPDATION AND DELETION OF ROWS IN TABLES

4.4.3.2 DELETING RECORDS

Example: mysql> DELETE FROM movies;

Output:

ID	TITLE	YEAR

Example: mysql> DELETE FROM movies WHERE year > 1960;

Output:

ID	TITLE	YEAR
1	AAA	1954
2	BBB	1959

4.4.3 INSERTION, UPDATION AND DELETION OF ROWS IN TABLES

4.4.3.3 UPDATING RECORDS

Example: mysql> UPDATE movies SET title = 'AAA' WHERE title = 'BBB';

Output:

ID	TITLE	YEAR
1	AAA	1954
2	AAA	1959
3	CCC	1970
4	DDD	1975

Example: mysql> UPDATE movies SET mtitle = 'AAA', year = 1958 WHERE id = 4;

Output:

ID	TITLE	YEAR
1	AAA	1954
2	BBB	1959
3	CCC	1970
4	AAA	1958

4.4.4 RETRIEVING DATA

RETRIEVING DATA

Example: mysql> SELECT * FROM movies;

Output:

ID	TITLE	YEAR
1	AAA	1954
2	BBB	1959
3	CCC	1970
4	DDD	1975

Retrieving specific columns:

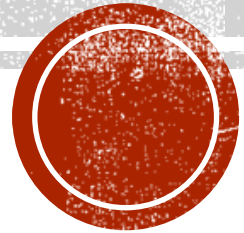
Example: mysql> SELECT title FROM movies;

Output:

TITLE
AAA
BBB
CCC
DDD

Programming with PHP & MySQL

II Year(CS) - SSCS3A



By

Dr.N.N.Krishna Veni,
Assistant Professor,
Holy Cross Home Science College,
Tuticorin

UNIT - 4
MYSQL

4.4.5 Sorting And Filtering Retrieved Data

- 4.4.5.1 Filtering Records
- 4.4.5.2 Sorting Records And Eliminating Duplicates
- 4.4.5.3 Limiting Results

UNIT - 4
MYSQL

Table Name: movies

ID	TITLE	YEAR
1	AAA	1954
2	BBB	1959
3	CCC	1970
4	DDD	1975

4.4.5 SORTING AND FILTERING RETRIEVED DATA

4.4.5.1 FILTERING RECORDS

Example: mysql> SELECT year FROM movies WHERE title = 'CCC';

Output:

YEAR
1970

Filtering Using Operators:

The = symbol previously used is an equality operator, used to test whether the left side of the expression is equal to the right side. MySQL comes with numerous such operators that can be used in the WHERE clause for comparisons and calculations.

4.4.5 SORTING AND FILTERING RETRIEVED DATA

OPERATOR	WHAT IT DOES
Arithmetic operators	
+	Addition
-	Subtraction
*	Multiplication
/	Division; returns quotient
%	Division; returns modulus
Logical operators:	
NOT aka !	Logical NOT
AND aka &&	Logical AND
OR aka ||	Logical OR
XOR	Exclusive OR

Comparison operators	
=	Equal to
<> aka !=	Not equal to
<=>	NULL-safe equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
BETWEEN	Exists in specified range
IN	Exists in specified set
IS NULL	Is a NULL value
IS NOT NULL	Is not a NULL value
LIKE	Wildcard match
REGEXP aka RLIKE	Regular expression match

4.4.5 SORTING AND FILTERING RETRIEVED DATA

Example: mysql> SELECT year, title FROM movies WHERE year > 1950;

Output:

YEAR	TITLE
1954	AAA
1959	BBB
1970	CCC
1975	DDD

You can combine multiple conditions by using the AND or OR logical operators.

Example: mysql> SELECT title FROM movies WHERE year >= 1955 AND year <= 1965;

(OR)

mysql> SELECT title FROM movies WHERE year BETWEEN 1955 AND 1965;

Output:

TITLE
BBB

4.4.5 SORTING AND FILTERING RETRIEVED DATA

The LIKE operator can be used to perform queries using wildcards, and comes in handy when you're not sure what you're looking for. Two types of wildcards are allowed when using the LIKE operator: the % wildcard, which is used to signify zero or more occurrences of a character, and the wildcard, which is used to signify exactly one occurrence of a character.

Example: mysql> SELECT title FROM movies WHERE title LIKE '%B%' OR title LIKE '%n%';

Output:

TITLE
BBB

4.4.5 SORTING AND FILTERING RETRIEVED DATA

4.4.5.2 Sorting records and eliminating duplicates

Table Name: persons

ID	NAME	GENDER	DOB
1	FFF	F	12.03.2000
2	AAA	M	26.09.2002
3	KKK	F	18.02.1999
4	CCC	M	30.05.2004
5	TTT	F	05.01.2006

4.4.5 SORTING AND FILTERING RETRIEVED DATA

Example: mysql> SELECT * FROM persons ORDER BY name ASC;

Output:

ID	NAME	GENDER	DOB
2	AAA	M	26.09.2002
4	CCC	M	30.05.2004
1	FFF	F	12.03.2000
3	KKK	F	18.02.1999
5	TTT	F	05.01.2006

Here is the same table sorted by date of birth, in descending order:

Example: mysql> SELECT * FROM persons ORDER BY dob DESC;

Output:

ID	NAME	GENDER	DOB
5	TTT	F	05.01.2006
4	CCC	M	30.05.2004
2	AAA	M	26.09.2002
1	FFF	F	12.03.2000
3	KKK	F	18.02.1999

4.4.5 SORTING AND FILTERING RETRIEVED DATA

To eliminate duplicate records in a table, add the DISTINCT keyword.

Example: mysql> SELECT DISTINCT gender FROM persons;

Output:

GENDER
F
M

4.4.5.3 Limiting Results

You can limit the number of records returned by MySQL with the LIMIT clause,

Example: mysql> SELECT name FROM persons LIMIT 0,4;

Output:

NAME
AAA
CCC
FFF
KKK
TTT

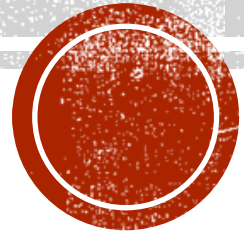
Example: mysql> SELECT name FROM persons ORDER BY dob LIMIT 0,2;

Output:

NAME
AAA
FFF
KKK

Programming with PHP & MySQL

II Year(CS) - SSCS3A



By

Dr.N.N.Krishna Veni,
Assistant Professor,
Holy Cross Home Science College,
Tuticorin

UNIT - 4
MYSQL

4.5 Advanced Filtering

4.5.1 Data Manipulation Functions

4.5.2 Aggregate Functions

4.5.2.1 SQL Aggregate Functions

4.5.2.2 Aggregates On Distinct Values

4.5.2.3 Combining Aggregate Functions

4.5 ADVANCED FILTERING

Table Name: persons

ID	NAME	GENDER	DOB
1	FFF	F	12.03.2000
2	AAA	M	26.09.2002
3	KKK	F	18.02.1999
4	CCC	M	30.05.2004
5	TTT	F	05.01.2006

4.5 ADVANCED FILTERING

DATA MANIPULATION FUNCTIONS

MySQL comes with over 100 built-in functions to help you perform calculations and process the records in a result set. These functions can be used in a SELECT statement, either to manipulate field values or in the WHERE clause.

You can calculate the number of record using COUNT() function, as in the following:

Example: mysql> SELECT COUNT(*) FROM persons;

Output:

COUNT(*)
5

Example: mysql> SELECT name, LENGTH(name) FROM persons;

Output:

NAME	LENGTH(NAME)
TTT	3
CCC	3
AAAA	4
FFF	3
K	2

4.5 ADVANCED FILTERING

You can use the DATE() function to format date and time values into a human-readable form.

Example: mysql> SELECT name, DATE_FORMAT(dob, '%W %d %M %Y') FROM persons;

Output:

NAME	DOB
FFF	Wednesday12 March 2000
AAA	Tuesday 2 September 2002
CCC	Monday 30 May 2004
KKK	Sunday 18 February 1999
TTT	Friday 05 January 2006

You can even use functions in the WHERE clause of a SELECT statement.

Example: mysql> SELECT name FROM persons WHERE YEAR(NOW()) - YEAR(dob) >100

AGGREGATE FUNCTIONS

It is often necessary to summarize data without actually retrieving it all, and MySQL provides special functions for this purpose. Using these functions, MySQL queries are often used to retrieve data for analysis and reporting purpose. Examples of this type of retrieval are

- Determining the number of rows in a table (or the number of rows that meet some condition or contain a specific value)
- Obtaining the sum of a group of rows in a table
- Finding the highest, lowset, and average values in a table column (either for all rows or for specific rows)

FUNCTION	DESCRIPTION
AVG()	Returns a column's average value
COUNT()	Returns the number of rows in a column
MAX()	Returns a column's highest value
MIN()	Return a column's lowest value
SUM()	Return the sum of a column's value

4.5.2 AGGREGATE FUNCTIONS

4.5.2.1 SQL AGGREGATE FUNCTIONS

Table Name: stud

Reg.no	Name	M1	M2	M3
1	AAA	67	92	77
2	BBB	89	57	98
3	CCC	39	38	45
4	DDD	54	57	67

The AVG ( ) Function: AVG () can be used to returns the average value of all columns or of specific columns or rows.

Example: mysql> **SELECT AVG** (m1) **AS** mark1 **FROM** stud;

Output:

Mark1
62.25

4.5.2 AGGREGATE FUNCTIONS

The COUNT () function

- Use count (*) to count the number of rows in a table, whether columns contain values or NULL values.
- Use count(column0to count the number of rows that have values in a specific column, ignoring NULL values.

Example: mysql>**SELECT COUNT** (*) **AS** No_Of_Student **FROM** stud;

Output:

No_Of_Stud
4

The MAX () function

MAX () returns the highest value in a specified colum.MAX() requires that the column name be specified, as seen here:

Example: mysql> **SELECT MAX** (m2) **AS** maximum_mark **FROM** stud;

Output:

Maximum_mark
92

4.5.2 AGGREGATE FUNCTIONS

The MIN() Function

Example: mysql> **SELECT MAX** (m2) **AS** minimum_mark **FROM** stud;

Output:

Minimum_mark
38

The SUM() Function

Example: mysql> **SELECT SUM**(m3) As total_mark **FROM** stud;

Output:

Total_Mark
287

Example: mysql> **SELECT SUM**(m3) As total_mark **FROM** stud where id>=3

Output:

Total_Mark
112

4.5.2 AGGREGATE FUNCTIONS

4.5.2.2 AGGREGATES ON DISTINCT VALUES

The five aggregate functions can all be used in two ways:

- To perform calculations on all rows, specify no arguments, or specify no arguments at all (because All is the default behavior).
- To only include unique values, specify the DISTINCT arguments.

Example: mysql> **SELECT SUM**(DISTINCT m2) As total_mark **FROM** stud;

Output:

Total_Mark
187

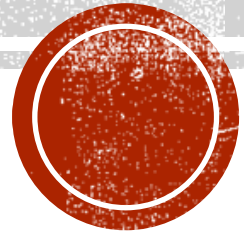
4.5.2.3 COMBINING AGGREGATE FUNCTIONS

Example: mysql>SELECT COUNT() AS no_of_students,
MIN(m1) AS minimum_mark,
MAX (m2) AS maximum-mark,
AVG(m3) AS average_mark
FROM stud;

No_of_Students	Minimum Mark	Maximum Mark	Average Mark
4	39	92	71.75

Programming with PHP & MySQL

II Year(CS) - SSCS3A



By

Dr.N.N.Krishna Veni,
Assistant Professor,
Holy Cross Home Science College,
Tuticorin

UNIT - 4
MYSQL

4.6 SET OPERATORS

UNION

UNION ALL

INTERSECT

MINUS

4.6 SET OPERATORS

MySQL supports few Set operations which can be performed on the table data. These are used to get meaningful results from data stored in the table, under different special conditions.

The different types of SET operations are:

- UNION
- UNION ALL
- INTERSECT
- MINUS

Consider the tables for the examples :

Table Name: First

ID	Name
1	Abhi
2	Adam

Table Name: Second

ID	Name
2	Adam
3	Chitra

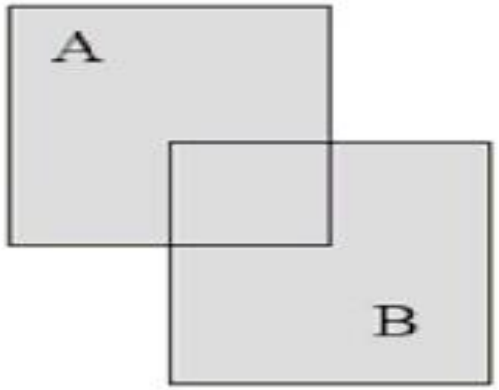
4.6 SET OPERATORS

UNION Operation

UNION is used to combine the results of two or more SELECT statements. However it will eliminate duplicate rows from its result set. In case of union, number of columns and datatype must be same in both the tables, on which UNION operation is being applied.

Union SQL query :

```
SELECT * FROM First
UNION
SELECT * FROM Second;
```



The result set table:

ID	Name
1	Abhi
2	Adam
3	Chitra

4.6 SET OPERATORS

4.6.2 UNION ALL

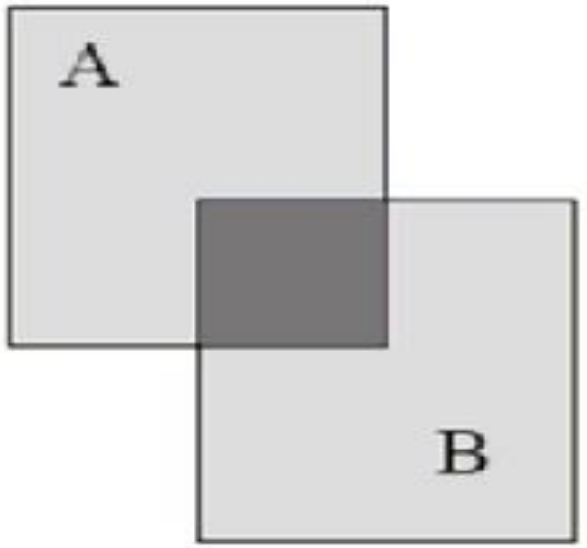
This operation is similar to Union. But it also shows the duplicate rows.

Union All query:

```
SELECT * FROM First
UNION ALL
SELECT * FROM Second;
```

The result set table:

ID	Name
1	Abhi
2	Adam
2	Adam
3	Chitra



4.6 SET OPERATORS

4.6.3 INTERSECT

Intersect operation is used to combine two SELECT statements, but it only returns the records which are common from both SELECT statements. In case of **Intersect** the number of columns and datatype must be same.

NOTE: MySQL does not support INTERSECT operator.

ID	Name
2	Adam

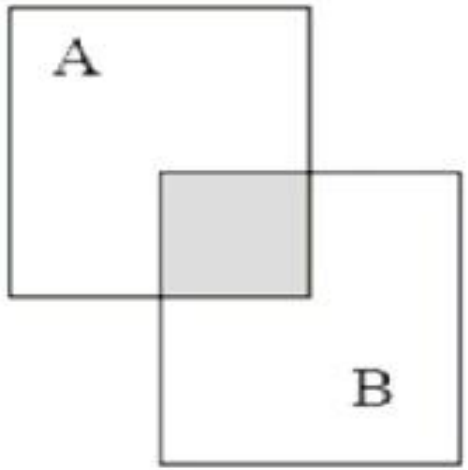
4.6 SET OPERATORS

MINUS

The Minus operation combines results of two SELECT statements and return only those in the final result, which belongs to the first set of the result.

Intersect query:

```
SELECT * FROM First
MINUS
SELECT * FROM Second;
```

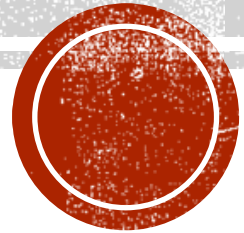


The result set table:

ID	Name
1	Abhi

Programming with PHP & MySQL

II Year(CS) - SSCS3A



By

Dr.N.N.Krishna Veni,
Assistant Professor,
Holy Cross Home Science College,
Tuticorin

UNIT - 4
MYSQL

4.7FULL TEXT SEARCHING

4.7.1 Using Full-Text Searching

4.7.2 Enabling Full-Text Searching Support

4.7 FULL TEXT SEARCHING

FULL TEXT SEARCHING

4.7.1 Using Full-Text Searching

In order to perform full-text searches, the columns to be searched must be indexed and constantly re-indexed as data changes. MySQL handles all indexing and reindexing automatically after table columns have been appropriately designated. After indexing, SELECT can be used with Match() and Against() to actually perform the searches.

Table Name:stud

Reg.No	NAME	Year	Remark
111	AAA	2000	Good Nature. Use to be brave in tough situation.
222	BBB	1994	Brave boy. Helping mind.
333	CCC	1999	Want to improve in studies. Good in sports.

4.7 FULL TEXT SEARCHING

4.7.2 Enabling Full-Text Searching Support

Generally, full-text searching is enabled when a table is created. The CREATE TABLE statement accepts a FULLTEXT clause, which is a comma-delimited list of the columns to be indexed. The following CREATE statement demonstrates the use of the FULLTEXT clause

Example: mysql> CREATE TABLE stud

```
(-> regno int(10) UNSIGNED NOT NULL,
 -> name varchar(255) NOT NULL default '',
 -> year year(4) NOT NULL default '0000',
 ->remark text NULL
 -> PRIMARY KEY (regno)
 ->) TYPE=MyISAM;
```

4.7 FULL TEXT SEARCHING

4.7.3 Performing Full-Text Searches

After indexing, full-text searches are performed using **two functions**:

- Match() to specify the columns to be searched
- Against() to specify the search expression to be used.

Example: SELECT remark FROM stud WHERE Match(remark) Against('brave');

Output:

Remark
Brave boy.
Use to be brave in tough situation.

- **Use Full Match() Specification**: The value passed to Match() must be the same as the one used in the FULLTEXT() definition. If multiple columns are specified, all of them must be listed (and in the correct order).
- **Searches Are Not Case Sensitive**: Full-text searches are not case sensitive, unless BINARY mode is used.

4.7 FULL TEXT SEARCHING

The truth is that the search just performed could just as easily have used a LIKE clause, as seen here:

Example: SELECT remark FROM stud WHERE remark LIKE '%brave%';

Output:

Remark
Use to be brave in tough situation.
Brave boy.

This SELECT retrieves the same two rows, but the order is different. Neither of the two SELECT statements contained an ORDER BY clause.

The LIKE statement returns data in no particularly useful order. But the full-text searching returns data ordered by how well the text matched. Both rows contained the word brave, but the row that contained the **word brave as the first word ranked higher than the row that contained it as the fourth word.**

This is important. **An important part of full-text searching is the ranking of results.** Rows with a higher rank are returned first.

4.7 FULL TEXT SEARCHING

Example: SELECT remark Match(remark) Against('brave') AS rank FROM stud;

Output:

Remark	Rank
Good Nature.	0
Use to be brave in tough situation.	1.5
Brave boy.	1.6
Helping mind.	0
Want to improve in studies.	0
Good in sports.	0

.

4.7 FULL TEXT SEARCHING

4.7.4 Ranking Multiple Search Term

If multiple search terms are specified, **those that contain the most matching words will be ranked higher than those with less** (or just a single match).

4.7.5 Using Query Expansion

When query expansion is used, MySQL makes two passes through the data and indexes to perform your search:

- First, a basic full-text search is performed to **find all rows that match the search criteria**.
- Next, MySQL examines **those matched rows and selects all useful words**.

4.7 FULL TEXT SEARCHING

Example: A simple full-text search, without query expansion:

SELECT remark FROM stud WHERE Match(remark) Against('nature');

Output:

Remark
Good Nature.

Example: A simple full-text search, with query expansion:

SELECT remark FROM stud WHERE Match(remark) against('nature' WITH QUERY EXPANSION);

Remark
Good Nature.
Good in sports.

This time two rows were returned. The first contains the word nature and is thus ranked highest. The second row has nothing to do with nature, but as it contains one word that is also in the first row (good) it was retrieved, too. As you can see, query expansion greatly increases the number of rows returned, but in doing so also increases the number of returns that you might not actually want.

4.7 FULL TEXT SEARCHING

4.7.6 Boolean Text Searche

MySQL supports an additional form of full-text searching called boolean mode. In Boolean mode you may provide specifics as to

- Words to be matched
- Words to be excluded (if a row contained this word it would not be returned, even though other specified words were matched)
- Ranking hints (specifying which words are more important than others so they can be ranked higher)
- Expression grouping
- And more

Example:

Input SELECT remark FROM stud WHERE Match(remark) Against('good' IN BOOLEAN MODE);

Output:

Remark
Good Nature.
Good in sports.

.

4.7 FULL TEXT SEARCHING

To match the rows that contain good but not any word with nature, the following can be used:

Example:

Input SELECT remark FROM stud WHERE Match(remark) Against('good nature*' IN BOOLEAN MODE);

Output:

Remark
Good in sports.

This time only one row is returned. Again, the word good is matched, but this time nature* instructs MySQL to explicitly exclude any row that contains nature* (any word beginning with nature, including nature, which is why one of the rows was excluded).

.

4.7 FULL TEXT SEARCHING

Table 4.2. Full-Text Boolean Operators

Privilege	Description
+	Include, word must be present.
-	Exclude, word must not be present.
>	Include, and increase ranking value.
<	Include, and decrease ranking value.
( )	Group words into subexpressions (allowing them to be included, excluded, ranked, and so forth as a group).
~	Negate a word's ranking value.
*	Wildcard at end of word.
" "	Defines a phrase (as opposed to a list of individual words, the entire phrase is matched for inclusion or exclusion).

4.7 FULL TEXT SEARCHING

Example:

- SELECT remark FROM stud WHERE Match(remark) Against('+good +nature"' IN BOOLEAN MODE);

  This search matches rows that contain both the words good and nature.

- SELECT remark FROM stud WHERE Match(remark) Against('good nature' IN BOOLEAN MODE);

  Without operators specified, this search matches rows that contain at least one of rabbit or bait.

.

4.7 FULL TEXT SEARCHING

Example:

- SELECT remark FROM stud WHERE Match(remark) Against('>good <sports' IN BOOLEAN MODE);
  - **Analysis Match both good and sports, increasing the rank of the former and decreasing the rank of the latter.**

- SELECT remark FROM stud WHERE Match(remark) Against('+good +(<sports)' IN BOOLEAN MODE);
  - **This search matches the words safe and combination, lowering the ranking of the latter.**

.

Programming with PHP & MySQL
II Year(CS) - SSCS3A
UNIT – 5

By
Dr.N.N.Krishna Veni,
Assistant Professor,
Holy Cross Home Science College,
Tuticorin

PHP with MySQL

5.1 Working MySQL with PHP

PHP has included support for MySQL since version 3.*x* The MySQL API built into PHP is designed to accomplish four primary goals:

- Manage database connections
- Execute queries
- Process query results
- Provide debugging and diagnostic information

To illustrate these functions, let's create a simple MySQL database table, and then use PHP to connect to the server, retrieve a set of results, and format them for display on a web page.

Example:

```
CREATE TABLE items (itemID int(11) NOT NULL, itemName varchar(255) NOT NULL", itemPrice
                    float NOT NULL , PRIMARY KEY (itemID) ) TYPE=MyISAM;
                    INSERT INTO items VALUES (1, 'P', '3');
                    INSERT INTO items VALUES (2, 'K', '2');
                    INSERT INTO items VALUES (3, 'C', '14');
                    INSERT INTO items VALUES (4, 'P1', '1');
                    INSERT INTO items VALUES (5, 'C1', '4');
```

Item id	Item Name	Item price
1	P	3
2	K	2
3	C	14
4	P1	1
5	C1	4

Now, to do the same thing using PHP, create the following PHP script:

```
<html> <head></head>
<body>
<?php
        $connection = mysql_connect('localhost', 'guest', 'pass') or die ('Unable to connect!');
        mysql_select_db('db2') or die ('Unable to select database!');
        $query = 'SELECT * FROM items';
        $result = mysql_query($query) or die ('Error in query: $query. ' . mysql_error());

        if (mysql_num_rows($result) > 0)
        {
                echo '<table width=100% cellpadding=10 cellspacing=0 border=1>';
                echo'<tr><td><b>ID</b></td><td><b>Name</b></td><td><b>Price</b></td></tr>';

        while($row = mysql_fetch_row($result))
        {
                echo '<tr>';
                echo '<td>' . $row[0] . '</td>';
                echo '<td>' . $row[1] . '</td>';
                echo '<td>' . $row[2] . '</td>';
                echo '</tr>';
        }
                echo '</table>';
        }
        else
```

```
	{
		echo 'No rows found!';}
		mysql_free_result($result);
		mysql_close($connection);
	}
?>
</body>
</html>
```

Using PHP to perform and process a MySQL query involves several steps, which the following explains.

Step 1: To begin communication with the MySQL database server, you first need to open a connection to the server by the **mysql_connect()** function. The **mysql_connect()** function requires **three parameters**: the host name, username and password of MySQL to get access. If the function is successfully connected, it returns a link identifier, as **$connection**. This identifier is used throughout the script when communicating with the database.

Step 2: The next step is to select a database for use with the **mysql_select_db()** command, and then send the server a query through the **mysql_query()** function.

Step 3: The result is assigned to the variable **$result**. The number of rows in the result set is obtained from the **mysql_num_rows()** function. Individual field values can then be accessed as array elements.

Step 4: Each result set occupies some amount of memory. Once processing is completed, use the **mysql_free_result()** function to free up the used memory for other purposes. And close the connection with a call to **mysql_close().**

5.2 DATABASE CONNECTIVITY

In PHP, connections to the MySQL server are opened via the mysql_connect() function, which accepts three different arguments: the hostname, username and the corresponding password of the MySQL server.

Example:

```
<?php
	$connection = mysql_connect('mydbserver', 'guest', 'pass');
	if ($connection)
	{
		echo 'Connected!';
	}
	else
	{
		echo 'Could not connect!';
	}
?>
```

If a connection can be established, the mysql_connect() function returns a *link identifier,* which is used by other functions to communicate with the server. If a connection cannot be established, the function returns false and, an error message indicating the cause of failure will be printed to the output device

Example:

```
<?php
	$connection = @mysql_connect('mydbserver', 'guest', 'pass');
	echo $connection ? 'Connected!' : 'Could not connect!';
?>
```

It's good programming practice to explicitly close the MySQL connection once you finish using it. This is accomplished by calling the **mysql_close()** function, which closes the link and returns the used memory to the system.

Example:

```
<?php
        $connection = @mysql_connect('mydbserver', 'guest', 'pass');
        if ($connection)
        {
                mysql_close($connection);
        }
?>
```

5.2.1 Using Persistent Connections

When dealing with high-traffic sites, the constant opening and closing of MySQL server connections can often prove a drain on resources. In such situations, it is possible to obtain a reduction in overhead, as well as some performance gain, by replacing the call to **mysql_connect()** with a call to **mysql_pconnect()**.

The mysql_pconnect() function opens a **"persistent"** connection to the server. Such a persistent connection does not automatically end when the script creating it ends; rather, it remains available for use by other scripts requesting an equivalent connection to the MySQL server.

Example:

```
<?php
        $connection = mysql_pconnect('mydbserver', 'myuser', 'mypass') or die ('Unable to connect!');
        echo $connection ? 'Persistent connection open!' : 'Could not open persistent connection!';
?>
```

5.3 USAGE OF MYSQLCOMMANDS IN PHP

Once a connection has been opened, the next step is to select a database for use. This is done with the mysql_select_db() function, as given below

```
<?php
mysql_select_db('mydb'),
?>
```

Once the database has been selected, it becomes possible to execute queries on it. In PHP, MySQL queries are handled via the mysql_query() function,

Example:

```
<?php
        $result = mysql_query('SELECT * FROM items WHERE price < 10.00');
?>
```

Depending on the type of query, the return value of mysql_query() differs:

■ If the query is a data-retrieval query—for example, a SELECT or SHOW query—then mysql_query() returns a resource identifier pointing to the query's result set, or false on failure. The resource identifier can then be used to process the records in the result set.

■ If the query is a data manipulation query—for example, an INSERT or UPDATE query—then mysql_query() returns true if the query succeeds, or false on failure. The result-set processing functions outlined in the next section can now be used to extract data from the return value of mysql_query().

5.4 **PROCESSING RESULT SETS OF QUERIES**

The return value of a successful mysql_query() invocation can be processed in a number of different ways, depending on the type of query executed.

5.4.1 Queries Which Return Data

For SELECT-type queries, a number of techniques exist to process the returned data. They are
mysql_fetch_row()
mysql_fetch_assoc()
mysql_fetch_object()

mysql_fetch_row():

The simplest is the mysql_fetch_row() function, which returns each record as a numerically indexed PHP array. Individual fields within the record can then be accessed using standard PHP-array notation. The following example illustrates this:

```
<?php
// open connection to MySQL server
 $connection = mysql_connect('localhost', 'guest', 'pass') ↵ or die ('Unable to connect!');

// select database for use
mysql_select_db('db2') or die ('Unable to select database!');

// create and execute query
 $query = 'SELECT itemName, itemPrice FROM items'; $result = mysql_query($query) or die ('Error in query: $query. ' . mysql_error());

// check if records were returned
if (mysql_num_rows($result) > 0)
{
        // iterate over record set      // print each field
        while($row = mysql_fetch_row($result))
        {
                echo $row[0] . " - " . $row[1] . "\n";
        }
}
else
{
 // print error message
 echo 'No rows found!';
}

// once processing is complete // free result set
mysql_free_result($result);

// close connection to MySQL server
 mysql_close($connection);
 ?>
```

Notice, in the previous listing, how the call to mysql_fetch_row() is wrapped in a mysql_num_rows() conditional test. The mysql_num_rows() function returns the number of records in the result set and comes in handy to check whether the query returned any records at all.

mysql_fetch_assoc():

mysql_fetch_assoc() function is used to represent each row as an associative array of field-value pairs, a minor variation of the previously used technique.

Example:

```
<?php
 // open connection to MySQL server
$connection = mysql_connect('localhost', 'guest', 'pass') or die ('Unable to connect!');

// select database for use
mysql_select_db('db2') or die ('Unable to select database!');

// create and execute query
$query = 'SELECT itemName, itemPrice FROM items';
$result = mysql_query($query) or die ('Error in query: $query. ' . mysql_error());

if (mysql_num_rows($result) > 0)     // check if records were returned
{
        while($row = mysql_fetch_assoc($result))   // iterate over record set
        {
                echo $row['itemName'] . " - " . $row['itemPrice'] . "\n";    // print each field
        }
}
else
{
        echo 'No rows found!';          // print error message
}

mysql_free_result($result);             // once processing is complete // free result set

mysql_close($connection);               // close connection to MySQL server
?>
```

mysql_fetch_object():

In this case, field values are accessed using the field name instead of the index. There's also the mysql_fetch_object() function, which returns each row as an object, with properties corresponding to the field names. Here is an example:

Example:

```
<?php
// open connection to MySQL server
$connection = mysql_connect('localhost', 'guest', 'pass') or die ('Unable to connect!');

// select database for use
mysql_select_db('db2') or die ('Unable to select database!');

// create and execute query
$query = 'SELECT itemName, itemPrice FROM items';
$resu lt = mysql_query($query) or die ('Error in query: $query. ' . mysql_error());

if (mysql_num_rows($result) > 0)     // check if records were returned
{
        while($row = mysql_fetch_object($result))           // iterate over record set
        {
              echo $row->itemName . " - " . $row->itemPrice . "\n";   // print each field
         }
```

```
}
else
{
        echo 'No rows found!';                  // print error message
}

mysql_free_result($result);    // once processing is complete // free result set

mysql_close($connection);    // close connection to MySQL server
?>
```

In this case, each $row object is created with properties corresponding to the field names in that row. Row values can be accessed using standard $object-> property notation.

5.4.2 Queries That Alter Data

You can also use PHP's MySQL API for queries that don't return a result set, for example, INSERT or UPDATE queries. Consider the following example, which demonstrates by asking for user input through a form, and then INSERT-ing that data into the database:

```
<html>
<head> </head>
<body>
<?php
if (!$_POST['submit'])
{
        ?>
        <form action="<?=$_SERVER['PHP_SELF']?>" method="post">
                Item name: <input type="text" name="name">
                Item price: <input type="text" name="price">
                <input type="submit" name="submit">
        </form>
<?php
}
else
{
        $name = (trim($_POST['name']) == '') ? die ('ERROR: Enter a name') :
                mysql_escape_string($_POST['name']);

        $price = (trim($_POST['price'] == '') || !is_numeric($_POST['price'])) ? die ('ERROR: Enter a
           price') : $_POST['price'];

        $connection = mysql_connect('localhost', 'guest', 'pass') or die ('Unable to connect!');

        mysql_select_db('db2') or die ('Unable to select database!');

        $query = "INSERT INTO items (itemName, itemPrice) VALUES ('$name', '$price')";

        $result = mysql_query($query) or die ("Error in query: $query. " . mysql_error());

        echo 'New record inserted with ID ' . mysql_insert_id() . '<br \>';

        echo mysql_affected_rows() . ' record(s) affected';

        mysql_close($connection);
}
```

```
?>
</body>
</html>
```

The previous example has three new functions:

■ The mysql_escape_string() function escapes special characters in the user input, so it can be safely entered into the database. If the magic_quotes_gpc setting in your PHP configuration file is enabled, you might need to first call stripslashes() on the user input before calling mysql_escape_string(), to avoid characters getting escaped twice.

■ The mysql_insert_id() function returns the ID generated by the previous INSERT query (useful only if the table into which the INSERT occurs contains an AUTO_INCREMENT field).

■ The mysql_affected_rows() function returns the total number of rows affected by the last operation.

5.5 HANDLING ERRORS

PHP's MySQL API also comes with some powerful error-tracking functions that can reduce debugging time. The following example, deliberate error in the SELECT query string:

```
<?php
 // open connection to MySQL server
$connection = mysql_connect('localhost', 'guest', 'pass') or die ('Unable to connect!');

mysql_select_db('db2') or die ('Unable to select database!');          // select database for use

$query = 'SELECT FROM items';     // create and execute query
$result = mysql_query($query);

if(!$result)                // if no result
{
        echo 'MySQL error ' . mysql_errno() . ': ' . mysql_error();    // print MySQL error message
        mysql_close($connection);
 }
?>
```

The mysql_errno() function displays the error code returned by MySQL if there's an error in your SQL statement, while the mysql_error() function returns the actual error message. Turn these both on, and you'll find they can significantly reduce the time you spend fixing bugs

5.6 USING ANCILLARY FUNCTIONS

In addition to the general functions, PHP's MySQL API comes with a number of ancillary functions that may be used to find out more about the databases and tables on the MySQL server or to obtain server status information.

List of functions:

mysql_get_server_info()	Returns the version number of the MySQL server
mysql_get_proto_info()	Returns the version number of the MySQL protocol
mysql_get_client_info()	Returns the version number of the MySQL client
mysql_get_host_info()	Returns information on the MySQL host

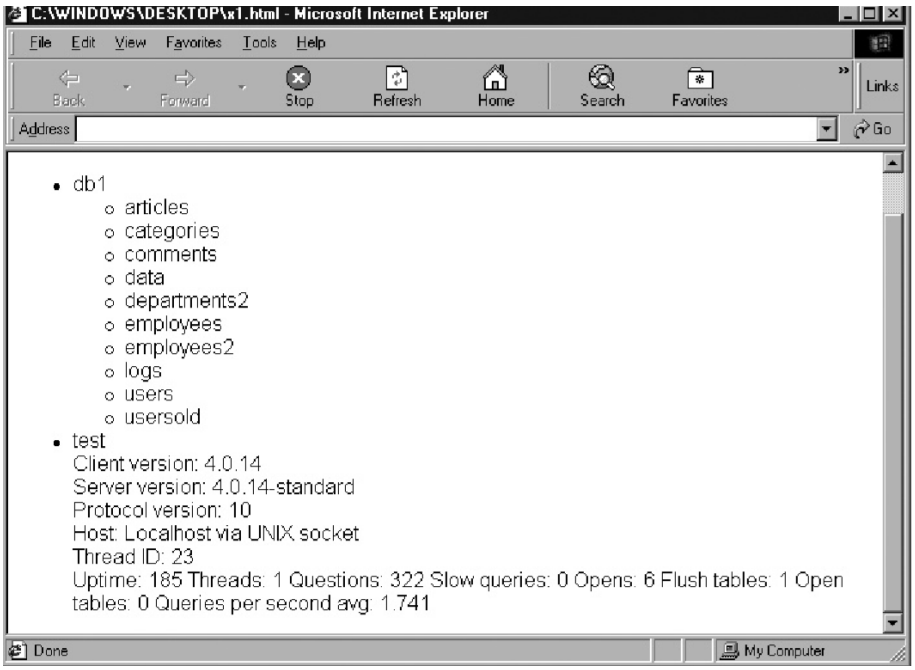
mysql_thread_id()	Returns the thread ID for the current MySQL connection
mysql_list_dbs()	Returns a list of databases available on the MySQL server
mysql_list_tables()	Returns a list of tables available in a specified MySQL database
mysql_list_fields()	Returns information about the fields of a specified MySQL table
mysql_stat()	Returns status information about the MySQL server
mysql_info()	Returns information about the last executed query
mysql_db_name()	Returns a name of a database from the list generated by mysql_list_dbs()
mysql_tablename()	Returns a name of a table from the list generated by mysql_list_tables()
mysql_ping()	Tests the server connection

Example:

```
<html>
 <head>
        <basefont face="Arial">
</head>
<body>
   <?php
      // open connection
      $connection = mysql_connect('localhost', 'guest', 'pass') or die ('Unable to connect!');

     // get list of available databases and tables
    $dbs = mysql_list_dbs($connection);
    echo 'Available databases and tables:'; echo '<ul>';

    for ($x=0; $x<mysql_num_rows($dbs); $x++)
   {
        $db = mysql_db_name($dbs, $x);              // print database name
        echo '<li>' . $db . '</li>';

        // for each database, get list of tables within it
        $tables = mysql_list_tables($db, $connection);
        echo '<ul>';

        for ($y=0; $y<mysql_num_rows($tables); $y++)      // iterate over table list
        {
              echo '<li>' . mysql_tablename($tables, $y) . '</li>';            // print table name
         }

         echo '</ul>';
}

// get version and host information
```

```
echo "Client version: " . mysql_get_client_info() . "<br />";
echo "Server version: " . mysql_get_server_info() . "<br />";
echo "Protocol version: " . mysql_get_proto_info() . "<br />";
echo "Host: " . mysql_get_host_info() . "<br />";
echo "Thread ID: " . mysql_thread_id() . "<br />";

$status = mysql_stat();          // get server status
echo $status;
mysql_close($connection);    // close connection
 ?>
</body> </html>
```

The below figure illustrates what the output might look like. The first part of this script is fairly simple: it runs the mysql_list_dbs() function to get a list of databases, and then it iterates over the list and runs the mysql_list_tables() function to retrieve the list of tables inside each. Next, the mysql_get_*_info() functions provide the client version number, the MySQL version number, the version number of the special MySQL clientserver protocol used for communication between the two, the current hostname, how it is connected to the MySQL server, and the connection thread ID.



Finally, new in PHP 4.3.0, is the mysql_stat() function, which returns a string containing status information on the MySQL server (including information on server uptime, open tables, queries per second, and other statistical information).

5.7 SETTING INPUT CONSTRAINTS AT THE DATABASE LAYER

When it comes to maintaining the integrity of your database, a powerful tool is provided by the database system itself: the capability to restrict the type of data entered into a field or make certain fields mandatory, using field definitions or constraints.

- Using the NULL Modifier
- Using the UNIQUE Modifier
- Using Field Data Types

5.7.1 Using the NULL Modifier

MySQL enables you to specify whether a field is allowed to be empty or if it must necessarily be filled with data, by placing the NULL and NOT NULL modifiers after each field definition. This is a good way to ensure that required fields of a record are never left empty, because MySQL will simply reject entries that do not have all the necessary fields filled in.

Example: mysql> CREATE TABLE products (-> id int(4), -> name varchar(50) ->);
mysql> INSERT INTO products VALUES (NULL, NULL);

Output of the above query
mysql> SELECT * FROM products;

ID	NAME
NULL	NULL

Now, look what happens if you make the name field mandatory
Example: mysql> CREATE TABLE products (-> id int(4), -> name varchar(50) NOT NULL ->);
mysql> INSERT INTO products VALUES (NULL, NULL);

Output of the above query
ERROR 1048: Column 'name' cannot be null

Thus, while the NOT NULL modifier can help reduce the incidence of empty or incomplete records in a database, it is not a comprehensive solution. It needs to be supplemented by application-level verification to ensure that empty strings are caught before they get to the database.

5.7.2 Using the UNIQUE Modifier

Using MySQL's built-in validation mechanisms has an important advantage: it makes it easy to perform certain types of validation that would be lengthy and time-consuming to write code for. Consider, for example, the situation of ensuring that a particular field contains only unique values. MySQL makes it possible to do this, simply by attaching a UNIQUE modifier to the field.

Example: mysql> CREATE TABLE users (-> username VARCHAR(50) NOT NULL UNIQUE->);
mysql> INSERT INTO users (username) VALUES ('tim');
mysql> INSERT INTO users (username) VALUES ('jon');

Now, if you attempt to enter another record with the value *tim* in the username field, MySQL will reject your entry with an error:

mysql> INSERT INTO users (username) VALUES ('tim');

ERROR 1062: Duplicate entry 'tim' for key 1

If you had to perform this type of validation at the application layer, the only way to do it would be to select all the records in the table, scan the username field to obtain a list of all values present in it, and check the user's input against each to eliminate duplication. Needless to say, this is expensive, both in terms of CPU cycles and time. Fortunately, the UNIQUE modifier renders it unnecessary.

5.7.3 Using Field Data Types

Checking for mandatory and unique values are just small pieces of a much bigger picture. It's also necessary to make sure that the data being entered is of the correct type—after all, you don't want string values in a numeric field or decimal values in a timestamp field. To this end, MySQL also requires you to specify the type of data a particular field can hold at the time of defining a table. Input that does not match the named data type is automatically converted into a more acceptable, though incorrect, value.

Example: mysql> CREATE TABLE items (-> id INT(2) NOT NULL, -> price INT(4) NOT NULL ->);
mysql> INSERT INTO items (id, price) VALUES (1, 'five');
mysql> SELECT * FROM items;

id	price
1	0

In this case, because the price field has been constrained to only store integers, the string *five* has been converted into a *0* and saved. Of course, this isn't perfect. Sure, you were able to avoid storing a string instead of a number, but you also simply replaced one problem with another: the field now contains a 0 instead of a valid price.

5.8 **VALIDATING INPUT AT THE APPLICATION LAYER**

When it comes to catching errors in user input, the best place to do this is at the point of entry—the application itself. That's why a good part of this chapter is devoted to showing you techniques you can use to catch common input errors and ensure that they don't get into your database.

- Checking for Required Values
- Restricting the Size of Input Data
- Checking the Type of Input Data
- Checking for Illegal Input Values
- Validating Dates
- Validating Multiple-Choice Input
- Matching Patterns

5.8.1 Checking for Required Values

One of the most common mistakes a novice programmer makes is forgetting to check for required field values. This can result in a database with numerous empty records, and these empty records can, in turn, affect the accuracy of your queries.

Example: mysql> CREATE TABLE users (-> username varchar(8) NOT NULL DEFAULT '', -> password varchar(8) NOT NULL DEFAULT '->) TYPE=MyISAM;

When inserting a record into this table, values must be specified for both username and password fields . Here's a script that enforces these constraints at the application level:

```
$username = (!isset($_POST['username']) || trim($_POST['username']) == "") ? die ('ERROR: Enter a username') : mysql_escape_string(trim($_POST['username']));

$password = (!isset($_POST['password']) || trim($_POST['password'] == "")) ? die ('ERROR: Enter a password') : mysql_escape_string(trim($_POST['password']));

$connection = mysql_connect('localhost', 'guest', 'pass') or die ('Unable to connect!');

$query = "INSERT INTO users (username, password) VALUES ('$username', '$password')";

$result = mysql_query($query) or die ("Error in query: $query. " . mysql_error());

mysql_close($connection);
```

The isset() function checks whether the named variable is set or not, and returns false if the variable has either not been set or assigned a NULL value. The trim() function removes the white space around the ends of the string, and then compares it with an empty string ("") to ensure that it contains at least one character.

If both tests return true, then the script proceeds to connect to the database and insert the record into the table. If either one returns false, the user clearly has not entered the corresponding form value, and the script terminates immediately, without even attempting to open a connection to the database.

5.8.2 Restricting the Size of Input Data

MySQL enables you to control the length of a particular field by adding a size modifier to the field data type. Now, the way MySQL works, values greater than the specified length are automatically truncated, with no notification or exception generated to let the user know about the change. This is disturbing, because it means that user data can easily get corrupted without the user's awareness.

Example: mysql> CREATE TABLE news (-> id INT (10) NOT NULL, -> title VARCHAR(50) NOT NULL ->)**;**

And here's the PHP script that replicates this constraint in a form:

```
$title = trim ($_POST['title']);
if (strlen($title) > 50)
{
        die ('ERROR: Title contains more than 50 characters');
}
```

To see this in action, try entering a string greater than 50 characters in the title field. When you submit the form, you'll see an error message, and the data will not be saved to the database until you correct the error. The code behind this is straightforward—just pass the user input to PHP's strlen() function, which returns the length of the string. You can then wrap this in an if() test to ensure that only strings under the specified limit pass muster.

5.8.3 Checking the Type of Input Data

An important test of user input involves checking the data type of input values against the database's expectations, and raising an error in the event of a mismatch.

Example: mysql> CREATE TABLE items (-> itemID INT(11) NOT NULL AUTO_INCREMENT, -> itemName VARCHAR(255) NOT NULL DEFAULT '',) TYPE=MyISAM;

Now, if you attempt to enter a string into any of the INT or FLOAT fields, MySQL will simply convert that string to a 0. At first glance, this might seem like an intelligent thing to do, because it avoids having to deal with error messages. However, it isn't, because the database now contains incorrect data. What is needed, then, is a way to verify the data type of a value before allowing it to be entered into the database. A useful PHP function to accomplish this is the is_numeric() function, demonstrated in the next

Example:

```
// check the itemName field
$itemName = (!isset($_POST['itemName']) || trim($_POST['itemName']) == "") ? die ('ERROR: Enter
        the item name') : mysql_escape_string(trim($_POST['itemName']));

// check the itemSPrice field
if(!isset($_POST['itemSPrice']) || trim($_POST['itemSPrice']) == "")
{
        die ('ERROR: Enter the item\'s selling price');
}
```

In this example, the first test is to ensure that the field is not empty. If this is true, the second test involves checking whether the value entered is a numeric string, with the is_numeric() function. Only if the user input passes both tests is it allowed to proceed into the database. In addition to the is_numeric()

function, you may also use PHP's character type extension to further test input before saving them to your database.

The important functions supported by this extension are listed in the below table.

Function	What It Does
ctype_alnum()	Check if a value contains only alphanumeric characters.
ctype_alpha()	Check if a value contains only alphabetic characters.
ctype_digit()	Check if a value contains only numeric characters.
ctype_print()	Check if a value contains only printable characters.
ctype_space()	Check if a value contains only white space characters.

5.8.4 Checking for Illegal Input Values

An application's particular business logic often demands custom validation routines of its own. To illustrate this, considerthe example of a form that asks the user to enter a positive two-digit number. Here, it is necessary to write a validation test to check if the user's input falls between 10 and 99 (both inclusive) and to display an error if it doesn't.

Example:

```
// check for presence of number
$num = (!isset($_POST['num']) || trim($_POST['num']) == "" || !is_numeric($_POST['num'])) ? die
        ('ERROR: Enter a number') : trim($_POST['num']);

// check for number range
if ($num < 10 || $num > 99)
{
        die ('ERROR: Enter a number between 10 and 99');
}
```

This type of custom validation can play an important role in avoiding common errors, such as the dreaded division-by-zero error.

5.8.5 Validating Dates

PHP hasa checkdate() function that provides an easy way to validate user-provided date values.

Example:

```
// check date
if (!checkdate($_POST['month'], $_POST['day'], $_POST['year']))
{
        die ('ERROR: Enter a valid date');
}
```

5.8.6 Validating Multiple-Choice Input

Checkboxes and drop-down lists are an important component of web forms, and it's often necessary to include validation for these controls in your PHP applications. Normally, the user's selections from these controls are submitted to the form processor in the form of an array, and it's necessary to use PHP's array functions to validate them.

Example:

```
// check the "hobbies" field for valid values
  $hobbies = ((sizeof($_POST['hobbies']) < 3) ? die ('ERROR: Please select at least 3 hobbies') :
  implode(',', $_POST['hobbies']));
}
```

Thus, it is convenient to use PHP's array functions—namely, the sizeof() function, which returns the number of elements in an array—to check whether the required number of options was selected.

Matching Patterns

Fortunately, PHP comes with these tools built in, with its support for regular expressions. Regular expressions (regex) are a powerful tool used in pattern-matching and substitution.. Depending on whether or not there's a match, appropriate action can be taken and appropriate program code executed. Regular expressions play an important role in the decision-making routines of web applications, and in complex find-and-replace operations.

The application needs to enforce the following constraints:

- The name may contain only uppercase (A–Z) or lowercase characters (a–z), with a minimum of three and a maximum of eight.

- The password may contain only lowercase characters (a–z) or integers (0–9), with a minimum of five and a maximum of eight.

- The e-mail address must conform to the standard *user@domain* format

5.9 FORMATTING QUERY OUTPUT WITH CHARACTER

It's essential that you know how to manipulate this string data and adjust it to fit the requirements of your application user interface. Both PHP and MySQL come equipped with numerous string manipulation functions.

- ➢ Concatenating String Values
- ➢ Padding String Values
- ➢ Dealing with Special Characters
- ➢ Altering String Case

Concatenating String Values

It's pretty simple—just string together the variables you want to concatenate using the PHP concatenation operation, a period (.). Concatenating fields from a MySQL result set is equally simple—just assign the field values to PHP variables and concatenate the variables together in the normal manner. To see how this works, consider the following table:

mysql> SELECT * FROM users;

username	fname	Lname
Xxx	AA	A
Yyy	BB	B
zzz	CC	C

MySQL comes with two built-in functions—CONCAT() and CONCAT_WS()—which can be used to glue fields together within the SQL query itself.

Padding String Values

The PHP trim() function, used to strip leading and trailing white space from string values prior to testing them for validity or inserting them into a database. However, PHP also comes with the str_pad() function, which does just the reverse: it pads strings to a specified length using either white space or a user-specified character sequence.
PHP code that demonstrates padding them:

Example: echo str_pad($row->name, 30, ' ', STR_PAD_LEFT) . '
';

The str_pad() function takes three parameters: the variable to be padded, the size it should be padded to, and the character to use for padding. By default, the function pads the string on the right side.

You can alter this default, however, by passing one of the constants STR_PAD_LEFT or STR_PAD_BOTH to the function as an optional fourth parameter. The PHP str_pad() function is functionally equivalent to MySQL's RPAD() and LPAD() functions, which pad a string from the right and left, respectively.

Example: mysql> SELECT RPAD(name, 20,'_'), LPAD(name, 20, '_') FROM ingredients LIMIT 0,2;

Altering String Case

If you need case manipulation, just reach for PHP's string manipulation API again. Four useful functions are here: strtolower(), which converts all characters in a string to lowercase; strtoupper(), which converts all characters to uppercase; ucfirst(), which converts the first character of a string to uppercase, and the useful ucwords(), which converts the first character of all the words in a string to uppercase.

Example:

```
while($row = mysql_fetch_object($result))
{
echo '<tr>';
echo '<td>' . ucfirst($row->fname) . ' ' . ↲
ucfirst($row->lname) . '</td>';
echo '<td>' . ucwords($row->addr) . '<br />' . ↲
strtoupper($row->city) . '</td>';
echo '<td>' . strtolower($row->email) . '</td>';
echo '</tr>';
}
```

mysql> SELECT CONCAT_WS('\n', UCASE(addr), UCASE(city)) AS address, LCASE(email) AS email FROM customers;

Dealing with Special Characters

When it comes to displaying large text blocks on a web page, a PHP developer must grapple with a number of issues. Special characters need to be protected, white space and line breaks must be preserved, and potentially malicious HTML code must be defanged. PHP comes with a number of functions designed to perform just these tasks.

```
if (mysql_num_rows($result) > 0)
{
while($row = mysql_fetch_object($result))
{
echo '<b>' . $row->title . '</b>';
echo '<p />'; echo $row->data;
echo '<p />';
}
```

The revised listing uses three new functions.

■ The htmlentities() function takes care of replacing special characters like ", &, <, and > with their corresponding HTML entity values. This function is useful to defang user-supplied HTML text and render it incapable of effecting the display or functionality of your web page. This function also translates these special characters and prevents them from being interpreted as HTML code by the browser.

■ Next, the wordwrap() function wraps text to the next line once it reaches a particular, user-defined size, by inserting the /n newline character at appropriate points in the text block (these are then converted

into HTML line breaks by the next function). This can be used to set artificial boundaries on the width of your text display area, and to maintain the integrity of your page layout.

■ Finally, the nl2br() function automatically preserves newlines in a text block, by converting them to HTML
 elements. This makes it possible to reproduce the original formatting of the text when it is displayed

5.10 Formatting Numeric Data

Just as you can massage string values into a number of different shapes, so, too, can you format numeric data. Both PHP and MySQL come with a full set of functions to manipulate integer and floating-point numbers, and to format large numeric values for greater readability.

Using Decimal and Comma Separators

When it comes to formatting numeric values in PHP, there are only **two functions**: number_format() and sprintf(). Of these, the former is easier to understand and use, so let's begin with that function. The number_format() function is used to display large numbers with comma and decimal separators. It can be used to control both the visibility and the appearance of the decimal digits, as well as the character used as the thousands separator.

Example:

```
while($row = mysql_fetch_object($result))
{
        echo '<tr>';
        echo '<td>' . $row->accountNumber . '</td>';
        echo '<td>' . $row->accountName . '</td>';
        echo '<td align=right>' . ↲
        number_format($row->accountBalance, 2, '.', ',') . '</td>';
        echo '</tr>';
}
```

Common function used to perform this type of number formatting is the sprintf() function, which enables you to define the format in which data is output.

Example:

```
<?php
        // returns 1.6666666666667
        print(5/3);
?>
```

As you might imagine, that's not very friendly. Ideally, you'd like to display just the significant digits of the result, so you'd use the sprintf() function, as in the following:

```
<?php
        // returns 1.67
        echo sprintf("%1.2f", (5/3));
?>
```

The PHP sprintf() function is similar to the sprintf() function that C programmers are used to. To format the output, you need to use *field templates*, templates that represent the format you'd like to display.

Template	-	**What It Represents**
%s	-	string
%d	-	decimal number
%x	-	hexadecimal number
%o	-	octal number

%f - float number

Examples of sprintf():

```
<?php
        // returns 00003
        echo sprintf("%05d", 3);
        // returns $25.99
        echo sprintf("$%2.2f", 25.99);
        // returns ****56
        printf("%'*6d", 56);
?>
```

5.11 Formatting Dates and Times

We can use PHP's mktime() function to obtain a UNIX timestamp for any arbitrary date/time value. However, because the timestamp returned by mktime() does not resemble traditional date/time displays, it is usually necessary to format this timestamp, so it is understandable to humans. This is particularly true in web applications, where dates and times are frequently displayed in human-readable, rather than machine-readable, form. To this end, PHP offers the date() function, which accepts two arguments: one or more *format specifiers,* which indicates how the timestamp should be formatted, and the timestamp itself.

Example:

```
echo date("h:i a d M Y", mktime());
echo date("d F Y", mktime(0, 0, 0, 04, 27, 2003));
echo date("H:i", mktime());
```

Specifier	-	**What It Means**
d	-	Day of the month; numeric
D	-	Day of the week; short string
F	-	Month of the year; long string
H	-	Hour; numeric 12-hour format
H	-	Hour; numeric 24-hour format
i	-	Minute; numeric
l	-	Day of the week; long string
L	-	Boolean indicating whether it is a leap year
m	-	Month of the year; numeric
M	-	Month of the year; short string
s	-	Seconds; numeric
T	-	Timezone
Y	-	Year; numeric
z	-	Day of the year; numeric

Example:

```
while($row = mysql_fetch_object($result))
{
        echo '<tr>';
        echo "<td>$row->name</td><td>" . ↲
        date("d M Y", $row->dob) . "</td>";
        echo '</tr>';
}
```

MySQL isn't far behind either: the RDBMS comes with powerful DATE_ FORMAT() and TIME_FORMAT() functions to manipulate the display of date and time values until they're exactly the way you want them. As with the PHP date() function, format specifiers are used to control the appearance of the output.

Below is the DATE_FORMAT() and TIME_FORMAT() functions.

Symbol	-	**What It Means**
%a	-	Short weekday name (Sun, Mon . . .)
%b	-	Short month name (Jan, Feb . . .)
%d	-	Day of the month
%H	-	Hour (01, 02 . . .)
%I	-	Minute (00, 01 . . .)
%j	-	Day of the year (001, 002 . . .)
%m	-	2-digit month (00, 01 . . .)
%M	-	Long month name (January, February . . . .)
%p	-	AM/PM
%r	-	Time in 12-hour format
%S	-	Second (00, 01 . . .)
%T	-	Time in 24-hour format
%w	-	Day of the week (0,1 . . .)
%W	-	Long weekday name (Sunday, Monday . . .)
%Y	-	4-digit year

Here are some examples demonstrating these in action:

1) mysql> **SELECT DATE_FORMAT(NOW(), '%W, %D %M %Y %r');**
   Thursday, 18th November 2004 12:07:55 PM
2) mysql> **SELECT DATE_FORMAT(19980317, '%d/%m/%Y');**
   17/03/1998
3) mysql> **SELECT DATE_FORMAT("20011215101030", "%H%i hrs on %a %d %M %y");**
   1010 hrs on Sat 15 December 01
4) mysql> **SELECT TIME_FORMAT(19690609140256, '%h:%i %p');**
   02:02 PM

Using the DATE_FORMAT() function, you can perform date formatting within your SQL query itself, without needing PHP's date() function.

Function	-	**What It Does**
DAYOFWEEK()	-	Returns a number (1 to 7) representing the day of the week for a date
DAYOFMONTH()	-	Returns the day component (1 to 31) of a date
DAYOFYEAR()	-	Returns a number (1 to 366) representing the day of the year for a date
DAYNAME()	-	Returns the weekday name for a date
HOUR()	-	Returns the hour component (0–23) of a time
MINUTE()	-	Returns the minute component (0–59) of a time
MONTH()	-	Returns the month component (1 to 12) for a date
MONTHNAME()	-	Returns the month name for a date
QUARTER()	-	Returns the quarter (1–2) in which a date falls
WEEK()	-	Returns the week number (0–53) for a date
YEAR()	-	Returns the year component (1000–9999) of a date

Here are some examples of these in action:

```
mysql> SELECT DAYOFMONTH(NOW()), DAYOFYEAR('1979-01-02');
mysql> SELECT DAYNAME(NOW()), MONTHNAME(NOW()), YEAR(NOW());
```