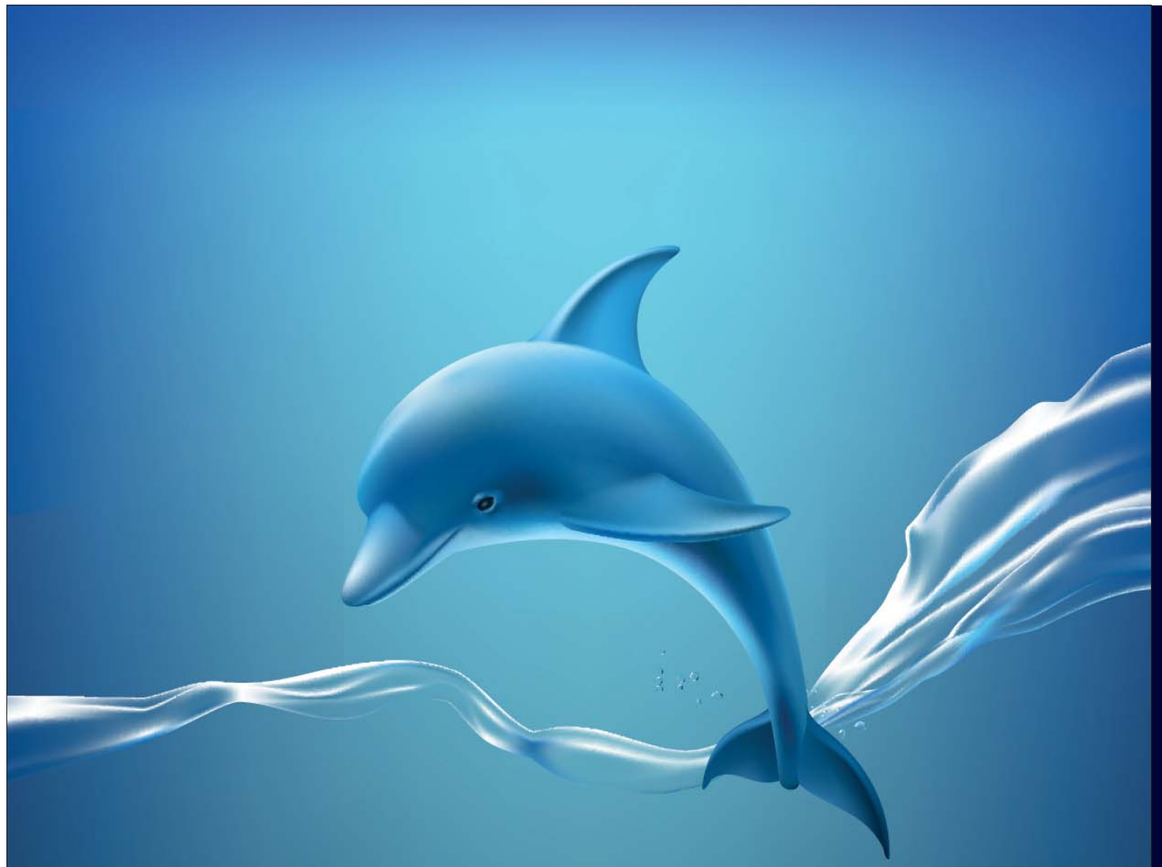
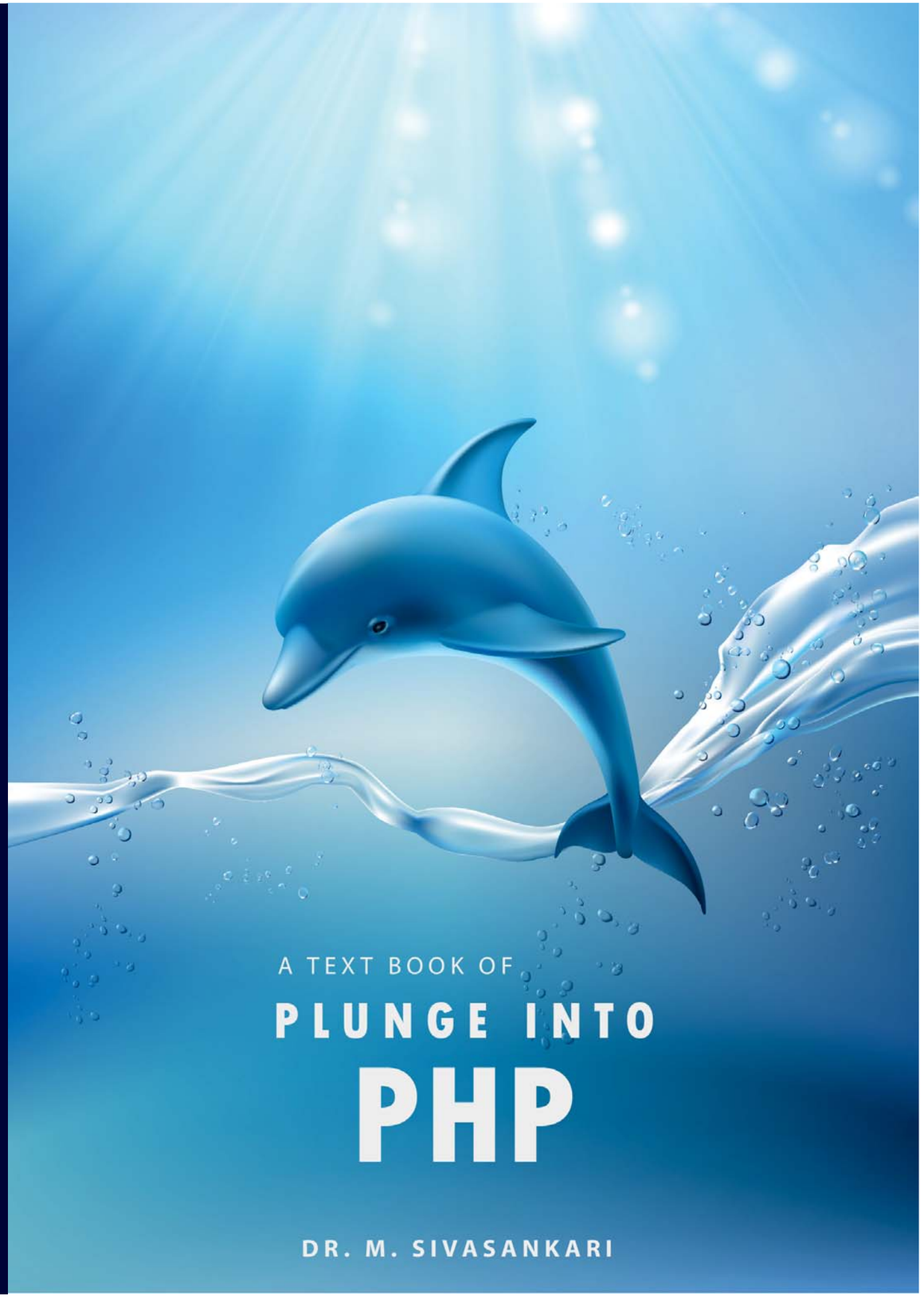


Dr.M.Sivasankari earned her Ph.D degree in Computer Science from MS University, Tirunelveli. She was a meritorious student both in academics and non academics during her college days and she won a number of prizes and medals in competitions conducted inside and outside of the college.

On top of that she is a Hat-trick World Record Holder with 40 different awards. She completed her M.Phil (Computer Science) at St.Xavier's College, Palayamkottai, Tirunelveli and M.Sc (Computer Science) & B.Sc (Computer Science) at G.Venkataswamy Naidu College, Kovilpatti. She is currently working as an Assistant Professor of Computer Applications at Don Bosco College of Arts and Science, Keela Eral. She has been actively playing a vital role in organizing State, National, International level Competitions for the college. She has been honored and felicitated in many occasions. She is being invited as a Resource Person for many programs related to her discipline. She is an orator. She has published a number of technical papers at National & International Conferences.

ISBN
No.

PLUNGE INTO PHP - Dr. M. SIVASANKARI



A TEXT BOOK OF

PLUNGE INTO PHP

B.E. / B.Tech. STUDENTS

As Per Anna University Syllabus

(Computer Science and Engineering) - Regulation – 2017

B.Sc(CS) / B.Sc(IT) / BCA

Manonmaniam Sundaranar University Syllabus

Dr. M. SIVASANKARI

Assistant Professor, Department of Computer Applications

Don Bosco College of Arts and Science,

Keela Eral.

To the Author

First Edition : November 2020

ISBN No. :

Price : Rs.

Publishers

Dr. M. Sivasankari
Assistant Professor, Department of Computer Applications
Don Bosco College of Arts and Science,
Keela Eral.

Contact for Copies:

Printed By :
M/s. Vinayaga Traders, 330-A, PKSA Arumugam Road,
Sivakasi- 626 189, Cell : 9486357318

PHP

ABOUT THE AUTHOR

Dr.M.Sivasankari earned her Ph.D degree in Computer Science from MS University, Tirunelveli. She was a meritorious student both in academics and non academics during her college days and she won a number of prizes and medals in competitions conducted inside and outside of the college. On top of that she is a Hat-trick Guinness World Record Holder with 40 different awards. She completed her M.Phil(Computer Science) at St.Xavier's College, Palayamkottai, Tirunelveli and M.Sc (Computer Science) & B.Sc (Computer Science) at G.Venkataswamy Naidu College, Kovilpatti. She is currently working as an Assistant Professor of Computer Applications at Don Bosco College of Arts and Science, Keela Eral. She has been actively playing a vital role in organizing State, National, International level programmes for the college. She has been honored and felicitated in many occasions. She is being invited as a Resource Person for many programs related to her discipline. She's an orator. She has published a number of technical papers at National & International Conferences.

Dr.M.Sivasankari

PREFACE

This book is intended for beginners, intermediate level and for all those who want to learn or expand their knowledge in Php Program. A systematic approach has been followed from the beginning to the end. Most of the concepts of Php language are explained in detail with practical applications. Solved programs have also been provided to help the learners to master the programming language.

A simple approach is used to understand the various concepts of Php language. Each Program is thoroughly explained and output is also shown. Each and every program given in the book is perfectly working.

These exercises are meant to test your skills & understanding for solving the problems. If you study this book in a right spirit, you will become an expert in Php Programming. Best of Luck!

Utmost care has been taken to write the book in order to make it free of errors. However, if you come across any error, you can feel free to contact me. Your suggestions & feedback may kindly be sent to the following maid id - mvsivasankari@gmail.com.

Dr. M. SIVASANKARI

ACKNOWLEDGEMENT

I would like to thank all those who have encouraged me to write the book.

I express my deep sense of gratitude to my Professor Dr. A. Ranichitra, Assistant Professor of Computer Science for her thorough review of every topic discussed in the book. It was of great help in improving this book.

I dedicate this book to my family and I wholeheartedly thank them for their patience & support extended to me all the times.

FOREWORD

Dr.A.Ranichitra
Assistant Professor,
Department of Computer Science,
Sri S.Ramasamy Naidu Memorial College, Sattur

The book **"Programming with PHP and MYSQL"**is Great for beginners and the learners new to PHP. The writer starts with the basics of the language. This helps the beginner to catch up quickly and then goes step by step on how PHP works. This book allows the learners to start with the easy stuff like how to create and run simple PHP scripts, which helps to modify web pages. It covers all the basics of the language with simple examples for each concept. Complex concepts are broken down into simple steps to ensure that the learner can easily master PHP. Concepts are presented in a "to-the-point" style to cater to the busy individual. Topics are carefully selected to give broad exposure to PHP. Working through the programs gives a chance to see how everything works. The Leaner will be able to use PHP to interact with the database and the server.

Best Wishes and **Congratulations** to the Writer in bringing the concept to reach every learner.

(Dr A.RANICHITRA)

CONTENTS

PROGRAMMING WITH PHP & MY SQL

UNIT-1

Introduction: introduction-open source PHP- PHP History- features-variables, statements operators, conditional statements- if- switch – nesting conditional- merging forms with conditional statements- loop- while- do-for-loop iteration with break and continue.

UNIT-2

Arrays and function: Arrays: creating an array- modifying array- processing array- grouping form with arrays- using array function creating user defined functions- using files- sessions cookies- executing external programs- creating sample application using PHP.

UNIT-3

File handling opening files using fopen- looping over file content with feof-reading text from a file using fgets - closing a file– reading character with fgets-reading whole file with file- get- content reading a file into an array with file. Check if a file exists- fscanf parse int- file- getting file information with stat- writing to a file- reading and writing binary files-locking files.

UNIT-4 : MYSQL

Effectiveness of MY SQL-MY SQL data types- creating and manipulation insertion-Updation and deletion of rows in tables- retrieved data – advanced data filtering data manipulation function- aggregate function – grouping data- sub queries - joining tables – set operators- full text searches.

UNIT-5 : PHP WITH MYSQL:

Working is MY SQL with PHP-database connectivity usage of MY SQL commands in PHP processing results. Sets of queries- handling errors-debugging and notice functions validating user input through data base layer and application layer- formatting query output with character-numeric-Data and time – sample data base applications.

UNIT 01
INTRODUCTION

Introduction to PHP

- PHP is an acronym for ―PHP- Hyper Text Pre-processor‖(earlier called Personal Home Page)
- PHP is a widely-used,open source scripting language.
- PHP is an HTML- embedded, server- side scripting language which is designed for web development
- It is also used as a general-purpose programming language.
- It was created by Rasmus Lerdorf in 1994& appeared in the market in 1995
- PHP/FI 2.0 & in turn quickly supplanted in 1997
- PHP 3.0 developed by Andi Gutmans & Zeev Suraski
- It was a complete rewrite of the original PHP/FI implementation
- PHP 4.0 was released in 2003.

SYNTAX

```
<?php
Code
?>
```

HTML- Hyper Text Mark-up language.

Example

```
<html>
,body>
<h1> My first PHP Page</h1>
<?php
        echo ‘ Hello World!’;
?>
    </body>
    </html>
```

Output

Hello World!

Features of PHP

There are many features given by PHP. All features discussed below one by one.

Familiarity

If you are in programming background then you can easily understand the PHP syntax. And you can write PHP script because of most of PHP syntax inherited from other languages like C or Pascal.

Simplicity

PHP provides a lot of pre-define functions to secure your data. It is also compatible with many third- party applications, and PHP can easily integrate with other. In PHP script there is no need to include libraries like c, special compilation directives like Java, PHP engine starts execution from(<?) escape sequence and end with a closing escape sequence(?>). In PHP script, there is no need to write main function. And also, you can work with PHP without creating a class.

Efficiency

PHP 4.0 introduced resource allocation mechanisms and more pronounced support for object- oriented programming, in addition to session management features. It‘s eliminating unnecessary memory allocation.

Security

Several trusted data encryption options are supported in PHP‘s predefined function set. You can use a lot of third-party applications to secure our data, allowing for securing our application.

Flexibility

We can say that PHP is a very flexible language because of PHP is an embedded language you can embed PHP scripts with HTML, JAVA SCRIPT, WML, ML and many others. You can run your PHP Script any device like mobile phones, tabs, laptops, PC and other because of PHP script execute on the server then after sending to the browser of your device.

Open Source

PHP is an open source programming language so you can download freely there is no need to buy a licence or anything.

Object Oriented

PHP has added some object-oriented programming features, and object-oriented programming became possible with PHP 4. With the introduction of PHP 5, the PHP developers have really beefed up the object-oriented features of PHP, resulting in both more speed and added features.

Embedding PHP in HTML

- PHP lets your embed commands in regular HTML pages.
- These embedded PHP commands are enclosing within special start & and tags which are read by the interpreter when it parses the page.

Note: Parses- to divide & identify the parts & their relations to each other.

Example

```
<?php
... PHP code....
?>
->PHP &HTML can be combined
<html>
<head><base font face =‖Arial‖></head>
<body>
<h2>Hi,we are the seniors </h2>
<?php
        //printout output
        echo‘<h2><i> we are intelligent student </i></h2>‘;
?>
</body>
</html>
```

WRITING STATEMENTS & COMMANDS

- A PHP script consists of one or more statements echo statement ending in a semicolon
- Black lines & outside the tags are ignored by the parses
- Only the code between the tags is read & executed.

Example

```
<? php
        //this is single line command
        /* & this is multiline command*/
?>
```

STORING VALUES IN VARIABLES

- Variables are the building blocks of any programming language.
- Variables are containersfor storing information.
- PHP supports a number of different variables types.

-Boolean
-Integers
-Floating point numbers
-Strings
-Arrays
-Objects
-Resources
-Nulls

Declaring (Creating) PHP Variables

In PHP, a variable starts with the $ sign, followed by the name of the variable.

Rules for PHP Variables

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

Example

$income, $day

Example

```
<html>
<head><base font face=’Arial’></head>
<body>
<h3>This is variable</h3>
<?php
```

```
$answer =’Hai’;
echo $answer;
?>
</body>
</html>
```

Output

This is variable
Hai

ASSIGNING & USING VARIABLES

- To assign a value to a variable use the assignment operator the equality (=) simple.
- This operator assigns a value (the right side of the equation) to a variable (the left side).
- To use a variable value in our script, simply call the variable by name, and PHP will substitute its value at run time.

Example

```
<?php
        $today=’June 27 2018’;
        echo ‘today is $ today’;
?>
```

Output

Today is June 27 2018

Saving Form Input in Variables

- Forms have always been one of the quickest and easiest ways to add interactivity to your web site.
- PHP can simplify the task of processing web-based forms substantially, by providing a simple mechanism to read user data submitted through a form into PHP variables.

Example

Consider the following sample form:

```
<html>
<head></head>
<body>
<form action="message.php" method="post">
Enter your message: <input type="text" name="msg" size="30">
<input type="submit" value="Send">
```

```
</form>
</body>
</html>
```

Detecting the Data Type of a Variable

- Variables can store data of different types, and different data types can do different things.
- PHP offers the gettype() function, which accepts a variable or value as argument.
- PHP supports the following data types

PHP String

A String is a sequence of characters like ―Hello ―World‖. A string can be any text inside quotes. We can use single or double quotes.

Example

```
<?php
    $x= 'Hello World!';
    echo $x;
?>
```

PHP Integer

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

Example

```
<?php
    $x=5985;
    echo $x;
    ?>
```

Output

```
5985
```

PHP Float

A float (floating point number) is a number with a decimal point or a number in exponential form.

Example

```
<?php
        $x= “10.56”;
        var_dump($x);
?>
```

Explanation

In the following example $x is a float. The PHP var_dump() function returns the data type and value.

PHP Boolean

- A Boolean represents two possible states: TRUE or FALSE.
- Booleans are often used in conditional testing.

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

PHP Object

- An object is a data type which stores data and information on how to process that data.
- In PHP, an object must be explicitly declared.
- First, we must declare a class of object. For this we use the class keyword.

Example

```
<?php
class car       {
        function car() {
                $this->model=’BMW’;
        }
}
//create an object
$herbie = new car();
// show object ptoperties
```

```
echo $herbie->model;
?>
```

PHP NULL value

Null is a special data type which can have only one value: NULL

A variable of data type NULL is a variable that has no value assigned to it.

Example

```
<?php
    $x='Hello World';
    $x=null;
    var_dump($x);
?>
```

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-strings()	Checks if a variable or value string
Is-number()	Check is 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 variables or value is an integer
Is-null	Check if a variables is an array
Is-objects()	Checks if a variable is an objects

Example

The following example illustrates this:

```
<?php
    // define variables
    $auth = true;
    $age = 27;
    $name = 'Bobby';
    $temp = 98.6;
    // returns "string"
```

```
echo gettype($name);
// returns "boolean"
echo  gettype($auth);
// returns "integer"
echo  gettype($age);
// returns "double"
echo gettype($temp);
?>
```

A NOTE ON STRINGS VALUES

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

Example

```
<? php
        $identity =’JamesBond’;
        $car=’BMW’;
        $sentence=’$identity drives a $car’;
        echo $sentence;
?>
```

Output

James Bond drives a BMW

OPERATORS

- Operators are the glue that let you do something useful with them.
- Operators are used to perform operations on variables and values.
- PHP comes with over 15 operators including operators for

*Assignment

*Arithmetic

*String

*Comparison

*logical operators

ARITHMETIC OPERATORS

- The PHP arithmetic operators are used with numeric values to perform common arithmetical operations such as addition, subtraction, multiplication, division etc.
- To perform mathematical operations on variables, use the standard arithmetic operators.

Example

```
<?php
        $num1=101;
        $num2=5;
        $sum=$num1+$num2;
        $sub=$num1-$num2;
        $multiply=$num1*$num2;
        $divide=$num1/$num2;
        $module=$num1%$num2;
?>
```

USING STRING OPERATORS

To add strings together use the string concatenation operators represents by a period(.).

Example

```
<?php
        $str1='John';
        $str2='example.com';
        $email=$str1'@':$str2;
?>
```

USING COMPARISON OPERATOR

- The PHP comparison operators are used to compare two values (number or string)
- To test whether two variables are different use anyone of PHP's many comparison operators.

Example

```
<?php
        $mean=29;
        $median=40;
        $mode=29;
        $result= ($mean<$median);
        $result= ($mean>$median);
        $result= ($mean<=$mode);
```

```
        $result= ($mean==$mean);
        $result= ($mean!=$mode);
    ?>
```

THE = = = OPERATOR

An important comparison operator in PHP 4.0 is the === operator which enable you to test both for equality & type.

Example

```
<?php
    // define two variables
    $str = '14';
    $int = 14;
    // returns true
    // since both variables contain the same value
    $result = ($str == $int);
    // returns false
    // since the variables are not of the same type
    // even though they have the same value
    $result = ($str === $int);
?>
```

USING LOGICAL OPERATORS

- The PHP logical operators are used to combine conditional statements.
- To linking together related conditions in a simple & elegant manner use one of PHP‘s four logical operators.

```
*logical AND
*logical OR
*logical NOT
*logical XOR
```

Example

```
<?php
    $user =‘joe‘;
    $pars =‘try m3‘;
    $save =1;
    $status =1;
    $result = ( $user ==‘joe‘) && ($pars ==’try m3‘);
```

```
        $result =! ($save ==1);
?>
```

AUTO-INCREMENT AND AUTO DECREMENT OPERATOR

- The auto-increment operator is a PHP operator designed to automatically increment the value of variables it an attached to by. It is represented by addition symbol.

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

Example

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

ADDING DECISION WITH CONDITIONAL STATEMENTS

A conditional enables you to test whether a specific condition is true or false and to perform different actions on the basic‘s types of conditional statements both which are discussed with the following sections.

USING THE IF ( ) STATEMENT

In PHP, the simplest form of conditional statement is the if() statement, which looks like this:

Syntax

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

Example

```
<?php
        if ($temp >= 100)
        {
                echo 'Very hot!';
```

```
        }
?>
```

Explanation

The argument to if() here is a conditional expression, which evaluates toeither true or false. If the statement evaluates to true, all PHP code within the curlybraces is executed; if not, the code within the curly braces is skipped and the linesfollowing the if() construct are executed.

IF ELSE STATEMENT

PHP also offers the if-else() construct, used to define an alternate block of code that gets executed when the conditional expression in the if() statement evaluates as false. This is good for ―either-or‖ situations, as illustrated in the following:

Syntax

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

Example

```
<?php
if ($temp >= 100)
{
        echo 'Very hot!';
}
else
{
        echo 'Within
limits';
}
?>
```

IF-ELSEIF-ELSE () STATEMENT

PHP also provide you with a way of handling multiple possibilities the if-else if- else () construct. This construct consists listing a number of possible result one after another and specifying the action to be taken for each.

Syntax

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

Example

```
<?php
if ($country == 'UK')
{
        $capital = 'London';
}
elseif ($country == 'US')
{
        $capital = 'Washington';
}
elseif ($country == 'FR')
{
        $capital = 'Paris';
}
else
{
        $capital = 'Unknown';
}
?>
```

The if-elseif-else () control structure assigns a different value to the $capital variable, depending on the country code. As soon as one of the if() branches within the block is found to be true, PHP will execute the corresponding code, skip the remaining if() statements in the block, and jump immediately to the lines following the entire if-elseif-else() block.

USING THE SWITCH ( ) STATEMENT

- The switch statement is used to perform different actions based on different conditions.
- Use the switch statement to select one of many blocks of code to be executed.
- Here a switch () statement evaluate a conditional expressions or decision variables depending on the result of the evaluation an appropriate case () instead.
- If no matches can be found a default block is executed instead.

Syntax

```
<? php
```

```
switch (conditional variable)
{
        case label 1:
                do this;// code to be executed if n=label1;
                break;
        case  label2:
                do this; // code to be executed if n=label2;
                break;
        case label n:
                do this; // code to be executed if n=label n;
                break;
                ……
        default:
                        code to be executed if n is different from all
        labels;
}
?>
```

Example

```
<? php
switch ($country)
{
        case ‘UK‘:
                $capital =‘London‘;
                break;
        case ‘US‘:
                $capital =‘washing ton‘;
                break;
        default:
                $capital =‘unknown‘;
}
?>
```

TERNARY OPERATOR

- Ternary Operator can perform the same operation in a single line as compared to conditional statement which uses multiple lines. Therefore, it reduces the length of your code.

- In ternary operator, if condition statement is true then statement 1 will execute otherwise statement 2 will execute.

Syntax

(condition)?(statement1):(statement2);

Example

```
<?php$msg =$dial count> 10? Cannot connect after 10 attempts:
_Dialing...';?>
```

NESTING CONDITIONAL STATEMENT

To handle multiple conditions, you can ―nest‖ conditional statements inside each other**.**

Example

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

REPEATING LOCATION WITH LOOP

Loops are used to execute the same block of code again and again, as long as a certain condition is true.

WHILE ( ) LOOP

- The while loop executes a block of code as long as the specified condition is true.
- The conditional expression specified evaluates to true, the loop will continue to execute.
- When the conditional becomes false the loop will be broken.

Syntax

```
<? php
while (condition is true)
{
        do this;// code to be executed
}
?>
```

Example

```
<?php
        $num =11;
        $upper limit =10;
        $lower limit =1;
        while ($lower limit <= $upper limit)
        {
                echo –$num X $lower limit =‖, ($num*$lower limit);
                $lower limit++;
        }
?>
```

DO ( ) LOOP

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

Syntax

```
<? php
do
{
        do this; // code to be executed;
}while (condition is true);
?>
```

Example

```
<?php
        $num =11;
        $UL =10;
        $LL =12;
        do
        {
                echo –$num X $LL =‖, ($num + $LL);
                $LL++;
        }
        while ($LL <= $UL);
?>
```

FOR LOOP ( )

- PHP offers for loop is used when you know in advance how many times the script should run.

Parameters

- Initialized counter –Initialize the loop counter value.

- Conditional test – Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- Increment/Decrement Counter – increases the loop counter value.

Syntax

```
<? php
for (initialized counter; conditional test; Increment/Decrement
counter);
{
        do this;
}
?>
```

Example

```
<?php
        for ($ x=0; $x<=10; $x++)
        {
                echo ‘ the number is: $x<br>’;
        }
?>
```

CONTROLLING LOOP ITERATION WITH BREAK AND CONTINUE

a. break

- The break keyword is used to exit a loop when it encounters an unexpected situation.
- It is advisable to check the division and use the break statement to exit the loop as soon as it becomes equal to zero.

Example

```
<?php
for ($x=-10; $x<=10; $x++)
{
        if ($x == 0) { break; }
        echo '100 / ' . $x . ' = ' . (100/$x);
}
?>
```

b. Continue

- The continue keyword is used to skip a particular iteration of the loop and move to the next iteration immediately.
- This statement can be used to make the execution of the code within the loop block dependent on particular circumstances.

Example

```
<?php
        for ($x=10; $x<=100; $x++)
        {
                if (($x % 12) == 0)
                {
                        echo "$x ";
                }
                else
                {
                        continue;
                }
        }
?>
```

- - - ☙ ☙ ☙ ☙ ☙ - - -

UNIT 02
USING ARRAYS AND CUSTOM FUNCTIONS

Definition of Array

- An array is complex variable that enables variables, if comes in handy when you need to store and represent can best be thought of as –container‖ variable, which can contain one or more values.

- An array stores multiple values in one single variable.

Syntax

array();

Example

```
<? php
        $flavour = array (_strawberry‘, _grape‘, _vanilla‘,  _caramel‘,
_chocolate‘);
?>
```

✓ ***an array alternative way to define an array***

Example

```
<?php
        $flavours [0] = _strawberry‘;
        $flavours [1] = _grape‘;
        $flavours [2] = _vanilla‘;
        $flavours [3] = _caramel‘;
        $flavours [4] = _chocolate‘;
?>
```

MODIFYING ARRAY ELEMENTS

To add an element to an array assigns a value using the next available index number or key.

Example

```
<?php
        $flavours [5] = _Mango‘;
        $flavours [] = _Mango‘;
```

```
        $fruits [_pink‘] = _peach‘;
?>
```

2.2.1. MODIFY AN ELEMENT OF AN ARRAY

To assign a value to the corresponding scalar variable if you wanted to replaces ―flavour‖ strawberry with ―blue berry‖ in the flavours array created previously script.

Example

We are use the following

```
<? php
        $flavours [0] = _blue berry‘;
?>
```

PROCESSING ARRAYS WITH LOOPS

To interactively process the data in a PHP array, loop over it using any of the loop construct.

Example

```
<html>
<head></head>
<body>
Today's shopping list:
<ul>
<?php
        // define array
        $shoppingList = array('eye of newt', 'wing of bat', 'tail of
        frog');
        // loop over it
        // print array elements
        for ($x = 0; $x <sizeof($shoppingList); $x++)
        {
                echo "<li>$shoppingList[$x]";
        }
?>
</ul>
```

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

FOREACH ( ) LOOP

- ✓ PHP 4.0 for the purpose of iterating over an array: the foreach () loop.
- ✓ This loop runs once for each element of the array, moving forward through the array each element iteration on each run.
- ✓ The statements within the only braces are executed and the currently selected array element is made available through a temporary loop unlike a for () loop, a for each () loop. Does not need a counter or a call to size of (); it keeps of its position in the array automatically.

Example

```
<html>
<head><\head>
<body>
Today shopping list
<ul>
<?php
//define array
$shopping list =array (_eye of newt‘, _using of bat‘, _tail of frog‘);
//loop over it
foreach ($ shopping list a $ item)
        echo –<LI> $item‖;
?>
<\ul>
<\body>
<\html>
```

CREATING USER DEFINED FUNCTIONS

Definition of Function

A function is simply a set of program statement that perform a specific false and that can be called, or executed, from anywhere in your program.

Besides the built-in PHP functions, it is possible to create your own functions:

- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute automatically when a page loads.
- A function will be executed by a call to the function.

DEFINING AND INVOKING FUNCTIONS

- In PHP, functions are defined using the special function keyword.
- This keyword is followed by the name of the function (which must conform to the standard naming rules for variables in PHP), a list of arguments (optional) in parentheses, and the function code itself, enclosed in curly braces.
- This function code can be any legal PHP code—it can contain loops, conditional statements, or calls to other functions.
- In the previous example, the function is named displayShakespeareQuote() and only contains a call to PHP's echo()function.
- Calling a user-defined function is identical to calling a built-in PHP function like size of() or die()—simply invoke it by using its name.
- If the function is designed to accept input values, the values can be passed to it during invocation in parentheses

Example

```
<?php
// define a function
function displayShakespeareQuote()
{
        echo 'Some are born great, some achieve greatness, and some have
greatness thrust upon them';
}
// invoke a function
displayShakespeareQuote();
?>
```

USING ARGUMENTS AND RETURN VALUES

a. Arguments

It is possible to create functions that accept different values from the main program and operate on these values to return different more pertinent result on each invocation these values are called ***arguments*** and they add a whole new level of power and flexibility to your code.

b. Return values

Usually, when a function is invoked, it generates a ***return value***. This return value is explicitly set within the function with the return statement.

Example

```
<?php
// define a function
function getTriangleArea($base, $height)
{
        $area = $base * $height * 0.5;
        return $area;
}
// invoke a function
echo 'The area of a triangle with base 10 and height 50 is '
.getTriangleArea(10, 50);
?>
```

Using Arrays with Argument Lists and Return Values

PHP fully supports passing arrays to functions in the argument list and returning arrays from functions with the return statement.

Example

```
<?php
// define a function
// with a single-argument list
function addDomainToUsername($u, $d)
{
        // create empty result array
        $resultArray = array();
        // process input array
        // add domain to username and place in result array
```

```
        foreach ($u as $element)
        {
                $resultArray[] = $element . '@' . $d;
        }
        // return result array
        return $resultArray;
}
// define variables
$users = array('john', 'jim', 'harry');
// send array as argument to function
// receive result array
$newUsers = addDomainToUsername($users, 'guess.me.domain');
?>
```

GLOBAL & LOCAL VARIABLES

a. LOCAL

The variables declared within a function are called local variables to that function and has its scope only in that particular function.

b. GLOBAL

- The variables declared outside a function are called global variables.
- A global variable can be accessed in any part of the program.
- To use a variable from the main program inside a function use the ***GLOBAL*** keyword before the variables name inside the function definition.

Example

```
<? php
$item =65;
$employee =125;
functionadditems ()
{
        GLOBAL $item;
        $item =$item+100;
}
function addemployee ()
```

```
{
        $employee =$employee+2000;
}
additems ();
addemployee ();
?>
```

USING FILES, SESSIONS COOKIES AND EXTERNAL PROGRM

READING DATA FROM A FILE

- ✓ To begin with let consider the process of opening a file and reading its contents.
- ✓ Create and run the following PHP script (remember to alter the value of $file variables to an actual file on your system that is a readable by the web server)

Example

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

1. This file can be created with the fopen () function, which accepts two arguments the name and path to the file and a string indicating the mode in which the file us to be opened (r for read)

2. If the fopen() function is successful it returns a file handle $fn which can be used for further information which the file fread() functions, which read the file and places its contains into a variable fread() is the number of bytes to be read you can usually obtain this information.
3. Close the file once you‘re done with the file it a good idea to does it with fclose().

WRITING DATA OF A FILE

The steps involved in writing data to a file are almost identical to those involved inreading it: open the file and obtain a file handle, use the file handle to write data to it,and close the file. There are two differences:

1. You must fopen() the file in write mode ('w' for write).
2. Instead of using the fread() function to read from the file handle, use thefwrite() function to write to it.

Example

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

TESTING FILE ATTRIBUTES

PHP also comes with a bunch of functions that enable you to test the status of a file.

This Table shows useful PHP file functions**.**

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

Example

```
<?php
// set file
$file = $_GET['file'];
// check if file exists
echo file_exists($file) ? 'File exists' : 'File does not exist';
// check if file is executable
echo is_executable($file) ? 'File is executable' :'File is not
executable';
// check if file is readable
echo is_readable($file) ? 'File is readable' : 'File is not readable';
// check if file is writable
echo is_writable($file) ? 'File is writable' : 'File is not writable';
// print file size
echo 'File size is ' .filesize($file) . ' bytes';
```

```
// print file owner
echo 'File owner is ' .fileowner($file);
// print file type
echo 'File type is ' . filetype($file);
?>
```

OBTAINING DIRECTORS LISTINGS

The task of iterating over one or moredirectories and processing the file list within each.

To meet this requirement, PHPoffers a comprehensive set of directory manipulation functions, which enabledevelopers to read and parse an entire directory listing.

To demonstrate, consider the following simple example, which lists all the filesin the directory /bin:

```
<?php
// initialize counter
$count = 0;
// set directory name
$dir = "/bin";
// open directory and parse file list
if (is_dir($dir))
{
        if ($dh = opendir($dir))
        {
                // iterate over file list
                // print filenames
                while (($filename = readdir($dh)) !== false)
                {
                        if (($filename != ".") && ($filename != ".."))
                        {
                                $count++;
                                echo $dir . "/" . $filename . "\n";
                        }
                }
        // close directory
        closedir($dh);
        }
}
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 thedirectory, and return a list of file names one after another.
- ✓ The closedir() function is used to destroy the directory handle.

CREATING A SESSION AND REGITERING SESSION VARIABLES

SESSION

- ❖ A session is a global variable stored on the server
- ❖ Each session is assigned a unique ID which us used to retrieve stored value.
- ❖ Session have the capacity to stored relatively large data compared to cookies.
- ❖ The session values are automatically deleted when the browser is closed.

In PHP the session_start () functions used to create a check session and generate a session ID once a session has been created it becomes possible to register any number of session variables which can store textual.

Example

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

DESTROYING A SESSION

- ✓ To remove all global session variables and destroy the session, usesession_unset() and session_destroy();

- ✓ To destroy an extant session—for example, on user logout—reset the $_SESSIONarray, and then use the session_destroy() function to erase session data.

Example

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

COOKIES

- ❖ A cookie is a small file with the maximum size of 4kb that the web server store on the client computer. Each time the same computer requests a page with a browser, it will send the cookie too.
- ❖ With PHP, you can both create and retrieve cookie values.
- ❖ A cookie created by a user can only be visible them.other users cannot see its value

2.14.1 *SETTING COOKIES*

In PHP cookies are set with the setcookie() function which accepts six arguments the cookie name its value its expiry data its path and domain and a Boolean flag indicating its security status only the first argument is required all the rest are optional. The set cookies () function return true if successful by checking for this you can only verify if the cookie was sent to the browser or not.

Syntax

setcookie(name, value, expire,path, domain,secure,httponly);

Only the name parameter is required. All other parameters are optional.

Example

```
<? php
        $ref =setcookie (_user name‘, _admin‘, mktime () +86400
        _/‘;
        if (!$ref)
         {
                echo –unable to set cookie‖;
        }
?>
```

RETRIEVING COOKIE DATA

Once cookie has been set for the domain it becomes available in the special $cookie associated array and its value may be accessed using standard array notation.

Example

```
<?php
if ($-COOKIE [_user name‘])
        echo –welcome back, $-COOKIE [_user name‘];
else
        echo _is this your first time here? Take our guided for! ‖;
?>
```

DELETING COOKIE

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

Example

```
<?php
        setcookie (_user name‘, _null‘, mktime ()-10000, _/‘);
?>
```

STORING DATA IN COOKIES

- ✓ Cookies allow web sites to store client-specific information in a file on the clientsystem, and retrieve this information on an as-needed basis.
- ✓ Cookies are typicallyused to bypass the stateless nature of the HTTP protocol, by using the client‘sdisk as a storage area for persistent data; however, they‘re dependent on the clientbrowser being configured to accept cookies.

RULES

When dealing with cookies, you should be aware of some ground rules:

1. Because cookies are used to record information about your activities ona particular site, they can only be read by the site that created them.

2. A single domain cannot set more than 20 cookies, and each cookie islimited to a maximum size of 4KB.

3. A cookie usually possesses five types of attributes. Table shows the lists them.

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 HTTPconnection

DELETING WITH DATE AND TIMES

PHP comes with some fairly powerful function to create and format time and date. Stamps the following sections discuss some

1) retrieving current date and time the easiest way to do this is with the get date ( ) function which returns an associative array containing current data and time to try it out create and return the following script.

```
<?php
        $current =get data [];
        $current-time =$current [_hours‘] –:‖
        $current [_minutes‘] —:‖ $current [_seconds‘] —:‖
        $current-data =$current [_day‘]._.‘ $current [_mon‘].‘.‘
        $current [_year‘];
        echo –it is now $current-time on $current-date‖;
```

```
?>
```

EXECUTING EXTERNAL PROGRAMS

- To run an external program from your PHP script, place the program commandline within backticks (``).
- The output of the command can also be assigned toa variable for further use within the script.
- Try the following example, which runsthe UNIX du command (to calculate disk usage) and places the resulting output ina PHP variable:

Example

```
<?php
        $output = `/bin/du -s /tmp/`;
        echo $output;
?>
```

- - - ☙ ☙ ☙ ☙ ☙ - - -

UNIT 03
FILE HANDLING FUNCTIONS

PHP fopen() function Definition and Usage

If fopen() fails, it returns FALSE and an error on failure. You can hide the error output by address _@‘ infront of the function name.

Syntax

fopen(filename, mode, includepath, context);

Parameter	Description
File name	Required. specifies the file or URL to open
Mode	Required. Specifies the type of access you require to the file/stream Possible values: • –r‖ (read only. Starts at the beginningof the file) • –r+‖ (read/write. Starts at thebeginning of thefile) • —w‖ (write only. Opens and clears the contents of file; or creates a new file ifit doesn‘t exist) • —w+‖ (read/write. Opens and clears the contents of file; or creates a new file ifit doesn‘t exist) • —a‖ (write only. Opens and writes tothe end of the file; or creates a new file if it doesn‘t exist) • –a+‖ (read/write. Preserve file contentby writing to the end of the file) • —x‖ (write only. Creates a new file. Returns FALSE and an error if filealready exists) • —x+‖ (read/write. Creates a new file. Returns FALSE and an error if filealready exists)
Include _ path	Optional. Set this parameter to _1‘ if you want tosearch for the file in the include _ path (in php.ini)
Context	Optional. Specifies the context of the file handle. Context is a set of options that can modify the behavior of a stream

Tips and Notes

NOTE: when writing to a text file, be sure to use the correct line-ending character! Unix systems use\n, windows systems use \r\n, and Macintosh systems use \r as the line ending character. Windows offers a translation flag (_t‘) which will translate \n to \r\n when working with the file. You can also use _b‘ to force binary mode. To use these flags, specify either _b‘ or _t‘ as the last character of the mode.

Example

```
<?php
    $file=fopen(–test.txt‖, –r‖);
    $file=fopen(–/home/test/test.t‖,
    ‖r‖);
    $file=fopen(—/home/test/test.tx
    t‖, —w+‖);
    $file=fopen(http://www.example.com/, –r‖);
    $file=fopen(ftp://user:password@example.com/test.txt,
    —w‖);
?>
```

PHP feof() function Definition and Usage

- The feof() function checks if the —end-of-file‖ (EOF) has been reached.
- This function returns TRUE if an error occurs, or if EOF has been reached. Otherwise it returns FALSE.

Syntax

feof(file);

Parameter	Description
File	Required. Specifies the open file to check

Example

```
<?php
        $file = fopen(―test.txt‖, ―r‖);
        //output a line of the file until the end is reached
        $line = fgets($file);
        while (!feof($file))
        {
        echo $line. ‖<br/>‖;
        $line = fgets($file);
        }
        fclose($file);
?>
```

The output of the code above will be

Hello, this is a test file.

There are three lines here.

This is the last line.

PHP fgets() function Definition and Usage

- The fgets() function returns a line from an open file.
- The fgets() function stops returning on a new line, at the specified length, or at EOF, whichever comes first.
- This function returns FALSE on failure.

Syntax

```
fgets(file, length);
```

Parameter	Description
File	Required. Specifies the file to read from
Length	Optional. Specifies the number of bytes to read. Default is 1024 bytes.

Example 1

```
<?php
    $file = fopen(―test.txt‖, ―r‖);
    echo fgets($file);
    fclose($file);
?>
```

The output of the code above will be

Hello, this is a test file.

Example 2

Read a file line by line:

```
<?php
    $file = fopen(―test.txt‖, ―r‖);
    while (! feof($file))
    {
        echo fgets($file). ―<br/>‖;
    }
    fclose($file);
?>
```

The output of the code above will be

Hello, this is a test file.
There are three lines here.
This is the last line.

PHP fclose() function Definition and Usage

- The fclose() function closes an open file
- This function returns TRUE on success or FALSE on failure.

Syntax

```
fclose(file);
```

Parameter	Description
File	Required. Specifies the file to close

Example

```
<? php
    $file = fopen (–test.txt‖, –r‖);
    //some
    code    to
    be
    executed
    fclose($fi
    le);
?>
```

PHP fgetc() function Definition and Usage

The fgetc() function returns a single character from an open file.

Syntax

```
fgetc(file);
```

Parameter	Description
file	Required. Specifies the file to check

Tips and Notes

Note: This function is slow and should not be used on large files. If you need to read one character at a time from a large file, use fgets() to read data one line at a time and then process the line one character at a time with fgetc().

Example 1

```
<?php
        $file = fopen(―test.txt‖, ―r‖);
        echo fgetc($file);
        fclose($file);
?>
```

The output of the code above will be:

H

Example2

Read file character by character

```
<?php
        $file = fopen(―test.txt‖, ―r‖);
        while(! feof($file))
        {
                echo fgetc($file);
        }
        fclose($file);
?>
```

The output of the code above will be

Hello, this is a test file.

PHP file_get_contents() function

This function is the preferred way to read the contents of a file into a string. Because it will use memory mapping techniques, if this is supported by the server, to enhance (improve) performance.

Syntax

file_get_contents (path, include_path, context, start, max length);

Parameter	Description
Path	Required. Specifies the file to read
Include-path	Optional. Set this parameter to _1‘ if you want to search for the file in the include-path (in php.ini) as well
Context	Optional. Specifies the context of the file handle. Context is a set of options that can modify the behaviour of a stream. Can be skipped by using NULL.
Start	Optional. Specifies where in the file to start reading. This parameter was added in PHP 5.1
Max-length	Optional. Specifies how many bytes to read. Thisparameter was added in PHP 5.1

Tips and Notes

Tip: this function is binary-safe (meaning that both binary data, like images, and character data can be written with this function).

Example

```
<?php
    echo file-get-contents(—test.txt‖);
?>
```

The output of the code above will be

```
This is a test file with test
text.
```

PHP fscanf() function

Definition and Usage

- The fscanf() function parses the input from an open file according to the specified format.

Syntax

```
fscanf(file, format, mixed);
```

Parameter	Description
File	Required. Specifies the file to check
Format	Required. Specifies the format. Possible format values: • %%-returns a percentsign • %b-binarynumber • %c-the character according to the ASCII value • %d- signed decimalnumber • %e-unsigned decimalnumber • %f-floating-point number (localsettings aware) • %F- floating-point number (notlocal settingsaware) • %o-octalnumber • %s-string • %x-hexadecimalnumber(lowercase letters) • %X- hexadecimalnumber(uppercase letters) Additional format values. These are placed between the % and the letter (example %.2f): □ +(force both + and - in front of numbers. By default, only negative numbers are marked) • _ (specifies what to use as padding. Default is space. Must be used together with the width specifies. Example:%'x20s (this uses —x‖ aspadding) • - (left-justifies the variablevalue) • [0-9] (specifies the minimum widthheld to the variablevalue) • [0-9] (specifies the number ofdecimal digits maximum stringlength) • Note: if multiple additional format values are used, they must be in the same order as above.

Tips and Notes

Note: any whitespace in the format string matches any whitespace in the input stream. This means that a tab (\t) in the format string can match a single space character in the input stream.

PHP fseek() function Definition and Usage

- The fseeek() function seeks in an open file.
- This function moves the file pointer from its current position to a new position, forward or backward, specified by the number of bytes.
- This function returns 0 on success, or -1 on failure. Seeking past EOF will not generate an error.

Syntax

```
fseek(file, offset, whence);
```

Parameter	Description
File	▪ Required. Specifies the open file to seek in
Offset	▪ Required. Specifies the new position (measured ▪ in bytes from the beginning of the file)
Whence	▪ Optional. (added PHP 4). Possible values: ▪ SEEK_SET-set position equal tooffset. Default ▪ SEEK_CUR-set position to current location plusoffset ▪ SEEEK_END-set position to EOF plus offset (to move to a position beforeEOF, ▪ the offers must be a negative value)

Tips and Notes

Tip: find the current position by using ftell()!

Example

```
<?php
    $file = fopen(―test.txt‖, ―r‖);
    //read first line fgets($file)
```

```
//move back to beginning of file fseek($file,
0);
        ?>
```

PHP copy() function Definition and Usage

- The copy()function copies a file
- This function returns TRUE on success FALSE on failure.

Syntax

```
copy(file, to-file);
```

Parameter	Description
File	Required. Specifies the file to copy
To-file	Required. Specifies the file to copy to

Tips and Notes

Note: if the destination file already exists, it will be overwritten.

Example

```
<?php
        echo copy(―source.txt‖, ―target.txt‖);
 ?>
```

The output of the code above will be

```
1
```

PHP unlink() function (or) delete()

Definition and Usage

- The unlink() function deletes a file.
- This function returns TRUE on success, or FALSE on failure.

Syntax

unlink(filename, context)

Parameter	Description
Filename	Required. Specifies the file to delete.
Context	Optional. Specifies the context of the file handle. Context is a set of options that can modify the behaviour of a stream

Example

```
<?php
        $file =―test.txt‖;
        if (!unlink($file))
        {
                echo (―Error deleting $file‖);
        }else {
                echo (―deleted $file‖);
        }
?>
```

PHP fread( ) function Definition and

Usage

- The fread() reads from an open file.
- The function will stops at the end of the file or when it reaches the specified length, whichever comes first.
- This function returns the read string, or FALSE on failure.

Syntax

fread(file, length);

Parameter	Description
File	Required. Specifies the open file to read from
Length	Required. Specifies the maximum number of bytes to read

Tips and Notes

Tip: this function is binary-safe (meaning that both binary data, like images, and character data can be written with this function)

Example 1

Read 10 bytes from file:

```
<?php
        $file = fopen("test.txt", "r");
        fread($file, "10");
        fclose ($file);
?>
```

Example 2

```
<?php
 $file=fopen("test.txt", "r");
 fread($file, filesize ("test.txt"));
 fclose($file);
?>
```

PHP fwrite( ) writes to an open file

- The function will stop at the end of the file or when it reaches the specified length, whichever comes first.
- This function returns the number of bytes written or FALSE on failure.

Syntax

```
fwrite(file, string, length);
```

Parameter	Description
File	Required. Specifies the open file to write to
String	Required. Specifies the string to write to the open File
Length	Optional. Specifies the maximum number of bytes to write.

Tips and Notes

Tip: This function is binary-safe (meaning that both binary data, like images, and character data can be written with this function).

Example

```
<?php
    $file = fopen(—test.txt‖, —w‖);
    echo fwrite ($file, —Hello World. Testing!‖);
    fclose($file);
?>
```

The output of the code above will be

Hello World. Testing!

PHP flock( ) function Definition and Usage

- The flock() function locks or releases a file.
- This function returns TRUE on success or FALSE on failure.

Syntax

```
flock(file, lock, block);
```

Parameter	Description
File	Required. Specifies an open file to lock or release

Lock	Required. Specifies what kind of lock to use. Possible values • LOCK_SH-shared lock (reader).Allow other processer to access thefile. • LOCK_EX- exclusive lock (writer).Prevent other processes from accessing thefile • LOCK_UN-release a shared or exclusive lock • LOCK_NB-avoids blocking other processes whilelocking
Block	Optional. Set to 1 to block other processes while Locking

Tips and Notes

Note: these locks only apply to the current PHP process. Other processes can modify or delete a PHP- locked file if permissions allow.

Note: flock() is mandatory under windows.

Tip: the lock is released also by fclose(), which is called automatically when script is finished.

Example

```
<?php
        $file = fopen(—test.txt‖, —w+‖);
        //exclusive lock
        if(flock ($file, LOCK_EX))
        {
                fwrite($file, —write something‖);
                //release lock
                flock($file, LOCK_UN);
        }else {
                echo –error locking file!‖;
        }

        fclose($file);
 ?>
```

- - - ☙ ☙ ☙ ☙ ☙ - - -

UNIT 04
MY SQL

INTRODUCTION

MySQL is the most popular Open Source Relational SQL Database Management System. MySQL is one of the best RDBMS being used for developing various web-based software applications. MySQL is developed, marketed and supported by MySQL AB, which is a Swedish company. This tutorial will give you a quick start to MySQL and make you comfortable with MySQL programming.

DATA TYPES

Properly defining the fields in a table is important to the overall optimization of your database. You should use only the type and size of field you really need to use.

For example, do not define a field 10 character wide, if you know you are only going to use 2 characters. These type of fields (or columns) are also referred to as data types, after the **type of data** you will be storing in those fields.

MySQL uses many different data types broken into ***three categories*** −

- Numeric
- Date and Time
- String Types.

Let us now discuss them in detail.

Numeric Data Types

MySQL uses all the standard ANSI SQL numeric data types, so if you're coming to MySQL from a different database system, these definitions will look familiar to you.

The following list shows the common numeric data types and their descriptions −

- **INT** − A normal-sized integer that can be signed or unsigned. If signed, the allowable range is from -2147483648 to 2147483647. If unsigned, the allowable range is from 0 to 4294967295. You can specify a width of up to 11 digits.
- **TINYINT** − A very small integer that can be signed or unsigned. If signed, the allowable range is from -128 to 127. If unsigned, the allowable range is from 0 to 255. You can specify a width of up to 4 digits.
- **SMALLINT** − A small integer that can be signed or unsigned. If signed, the allowable range is from -32768 to 32767. If unsigned, the allowable range is from 0 to 65535. You can specify a width of up to 5 digits.
- **MEDIUMINT** − A medium-sized integer that can be signed or unsigned. If signed, the allowable range is from -8388608 to 8388607. If unsigned, the allowable range is from 0 to 16777215. You can specify a width of up to 9 digits.
- **BIGINT** − A large integer that can be signed or unsigned. If signed, the allowable range is from -9223372036854775808 to 9223372036854775807. If unsigned, the allowable range is from 0 to 18446744073709551615. You can specify a width of up to 20 digits.
- **FLOAT(M,D)** − A floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 10,2, where 2 is the number of decimals and 10 is the total number of digits (including decimals). Decimal precision can go to 24 places for a FLOAT.

- **DOUBLE(M,D)** − A double precision floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 16,4, where 4 is the number of decimals. Decimal precision can go to 53 places for a DOUBLE. REAL is a synonym for DOUBLE.
- **DECIMAL(M,D)** − An unpacked floating-point number that cannot be unsigned. In the unpacked decimals, each decimal corresponds to one byte. Defining the display length (M) and the number of decimals (D) is required. NUMERIC is a synonym for DECIMAL.

DATE AND TIME TYPES

The MySQL date and time datatypes are as follows −

- **DATE** − A date in YYYY-MM-DD format, between 1000-01-01 and 9999-12-31. For example, December 30th, 1973 would be stored as 1973-12-30.
- **DATETIME** − A date and time combination in YYYY-MM-DD HH:MM:SS format, between 1000-01-01 00:00:00 and 9999-12-31 23:59:59. For example, 3:30 in the afternoon on December 30th, 1973 would be stored as 1973-12-30 15:30:00.
- **TIMESTAMP** − A timestamp between midnight, January 1st, 1970 and sometime in 2037. This looks like the previous DATETIME format, only without the hyphens between numbers; 3:30 in the afternoon on December 30th, 1973 would be stored as 19731230153000 (YYYYMMDDHHMMSS).
- **TIME** − Stores the time in a HH:MM:SS format.
- **YEAR(M)** − Stores a year in a 2-digit or a 4-digit format. If the length is specified as 2 (for example YEAR(2)), YEAR can be

between 1970 to 2069 (70 to 69). If the length is specified as 4, then YEAR can be 1901 to 2155. The default length is 4.

String Types

Although the numeric and date types are fun, most data you'll store will be in a string format. This list describes the common string datatypes in MySQL.

- **CHAR(M)** − A fixed-length string between 1 and 255 characters in length (for example CHAR(5)), right-padded with spaces to the specified length when stored. Defining a length is not required, but the default is 1.
- **VARCHAR(M)** − A variable-length string between 1 and 255 characters in length. For example, VARCHAR(25). You must define a length when creating a VARCHAR field.
- **BLOB or TEXT** − A field with a maximum length of 65535 characters. BLOBs are "Binary Large Objects" and are used to store large amounts of binary data, such as images or other types of files. Fields defined as TEXT also hold large amounts of data. The difference between the two is that the sorts and comparisons on the stored data are **case sensitive** on BLOBs and are **not case sensitive** in TEXT fields. You do not specify a length with BLOB or TEXT.
- **TINYBLOB or TINYTEXT** − A BLOB or TEXT column with a maximum length of 255 characters. You do not specify a length with TINYBLOB or TINYTEXT.
- **MEDIUMBLOB or MEDIUMTEXT** − A BLOB or TEXT column with a maximum length of 16777215 characters. You do not specify a length with MEDIUMBLOB or MEDIUMTEXT.

- **LONGBLOB or LONGTEXT** − A BLOB or TEXT column with a maximum length of 4294967295 characters. You do not specify a length with LONGBLOB or LONGTEXT.
- **ENUM** − An enumeration, which is a fancy term for list. When defining an ENUM, you are creating a list of items from which the value must be selected (or it can be NULL). For example, if you wanted your field to contain "A" or "B" or "C", you would define your ENUM as ENUM ('A', 'B', 'C') and only those values (or NULL) could ever populate that field.

AGGREGATE FUNCTIONS

Aggregate functions retrieve a single value after performing a calculation on a set of values.

Functions are COUNT( ), AVG( ), SUM( ), MIN( ), MAX( )

COUNT( )

The COUNT( ) function returns the number of rows that matches a specified criteria.

Syntax

SELECT COUNT(**column_name*****)*****FROM*** **table_name**

WHERE **condition*;***

Example

```
SELECT COUNT(ProductID)
FROM Products;
```

Note**:** NULL values are not counted.

AVG(*)*

The AVG( ) function returns the average value of a numeric column.

Syntax

***SELECT AVG(*column_name*)FROM* table_name**

WHERE* condition*;

Example

SELECT AVG(Price)FROM Products;

Note: NULL values are ignored.

SUM( )

The SUM( ) function returns the total sum of a numeric column.

Syntax

***SELECT SUM(*column_name*)FROM* table_name**

WHERE* condition*;

Example

SELECT SUM(Quantity)FROM OrderDetails;

Note: NULL values are ignored.

MIN( )

The MIN( ) function returns the smallest value of the selected column.

Syntax

SELECT MIN(*column_name*)FROM* table_name*WHERE* condition*;

Example

SELECT MIN(Price) AS SmallestPriceFROM Products;

MAX( )

The MAX() function returns the largest value of the selected column.

Syntax

***SELECT MAX(*column_name*)FROM* table_name**
WHERE* condition*;

Example

SELECT MAX(Price) AS LargestPrice FROM Products;

ADVANCED DATA FILTERING DATA MANIPULATION FUNCTION

4.4.1. Filtering Data Using the *OR* Operator

This can be achieved using the *WHERE* clause in combination with the *OR* operator.

For example, that we need to list all products in our sample database that cost less than $100 or greater than $200. The SQL statement to achieve this would read as follows:

```
SELECT * from products WHERE prod_price< 100 OR prod_price> 200;
```

The resulting output from executing the above SQL statement would contain all products *except* those priced between $100 and $200:

```
+-----------+---------------------------+-------------------------+------------+
| prod_code |        prod_name          |    prod_desc      |prod_price |
+-----------+---------------------------+-------------------------+------------+
|         3 | Microsoft 10-20 Keyboard | Ergonomic Keyboard         | 49 |
|         4 | EasyTech Mouse 7632        | Cordless Mouse                | 49 |
```

```
|       5 | Dell XPS 400                  | Desktop PC                  | 999 |
|       6 | Buffalo AirStation Turbo G | Wireless Ethernet Bridge    | 60 |
|       8 | Apple iPhone 8Gb              | Smart Phone                 | 399 |
+----------+--------------------------+-----------------------+-----------+
5 rows in set (0.00 sec)
```

Filtering Data Using the *AND* Operator

The *AND* operator selects rows based on the requirement that meet multiple requirements (as opposed to the "either or" approach of the *OR* operator).

For example, that we need to find a "Microsoft 10-20 Keyboard" that costs less than $30. To do so we would construct a SELECT statement as follows:

```
SELECT * from products WHERE prod_name = 'Microsoft 10-20
Keyboard' AND prod_price< 30;
```

Since we have no such keyboards in our table that meet the price criteria we get no results. If the customer decides to pay more we can change our search to find a suitable item:

```
SELECT * from products WHERE prod_name = 'Microsoft 10-20
Keyboard' AND prod_price< 50;
```

This time we find what the customer needs *and* the price she is willing to pay:

```
+----------+-----------------------+------------------+-----------+
| prod_code | prod_name             |    prod_desc       | prod_price |
+----------+-----------------------+------------------+-----------+
```

```
|         3 | Microsoft 10-20 Keyboard | Ergonomic Keyboard|         49 |
+-----------+-------------------------+-------------------+------------+
1 row in set (0.00 sec)
```

Combining *AND* and *OR* Operators

A SELECT statement with a *WHERE* clause can combine any number of *AND* and *OR* operators to create complex filtering requirements. For example, we can combine operators to find either a mouse or a keyboard that costs $49:

```
SELECT * from products WHERE prod_name = 'Microsoft 10-20
Keyboard' OR prod_name = 'EasyTech Mouse 7632' AND prod_price =
49;
+-----------+-------------------------+-------------------+------------+
| prod_code | prod_name               | prod_desc         | prod_price |
+-----------+-------------------------+-------------------+------------+
|         3 | Microsoft 10-20 Keyboard | Ergonomic Keyboard|         49 |
|         4 | EasyTech Mouse 7632      | Cordless Mouse    |         49 |
+-----------+-------------------------+-------------------+------------+
2 rows in set (0.00 sec)
```

Understanding Operator Precedence

When combining operators, it is important to understand something called operator precedence, which refers to the order in which operators in the same statement are evaluated.

By default, MySQL evaluates *AND* expressions before *OR* expressions regardless of whether the *OR* appears before the *AND* when reading the statement from left to right.

This means, for example, that the following SQL statement will evaluate the *AND* expression before it evaluates the *OR*:

```
SELECT prod_desc FROM products WHERE prod_name = 'WildTech
250Gb 1700' OR prod_name = 'Moto Razr' AND prod_price< 100;
```

To change the operator precedence (in this case to cause the *OR* expression to be evaluated first), simply surround the *OR* expression with parentheses as follows:

```
SELECT prod_desc FROM products WHERE (prod_name = 'WildTech
250Gb 1700' OR prod_name = 'Moto Razr') AND prod_price< 100;
```

The *OR* expression contained in the parentheses will now be executed before the *AND* expression.

Specifying a Range of Conditions using the *IN* Clause

The *IN* operator allows a range of filter criteria to be specified in a WHERE clause, all contained in parentheses and comma separated.

For example, imagine we need to list all products in our database that have a price of either $49, $100 or $999. Obviously, we could write a statement that uses a series of *OR* expressions. Whilst this would ultimately work a much quicker way is to provide a list of desired prices using the *IN* clause:

```
 SELECT * from products WHERE prod_price IN (49, 100, 999);
+-----------+-------------------------+------------------+------------+
| prod_code | prod_name               | prod_desc        | prod_price |
+-----------+-------------------------+------------------+------------+
```

```
|         3 | Microsoft 10-20 Keyboard | Ergonmoc Keyboard |         49 |
|         4 | EasyTech Mouse 7632      | Cordless Mouse    |         49 |
|         5 | Dell XPS 400             | Desktop PC        |        999 |
+-----------+--------------------------+-------------------+------------+
3 rows in set (0.00 sec)
```

Using the *NOT* Operator

The final operator to look at in this chapter is the *NOT* operator.

The *NOT* operator is used to negate the result of an expression and is of particular use when using the *IN* operator.

For example, we could very easily change our previous *IN* example so that it lists all the products in our table that do NOT cost $49, $100 or $999 simply by used a *NOT IN* operator combination:

```
mysql> SELECT * from products WHERE prod_price NOT IN (49, 100,
999);
+-----------+---------------------------+---------------------------+------------+
| prod_code | prod_name                 | prod_desc | prod_price |
+-----------+---------------------------+---------------------------+------------+
|         1 | WildTech 250Gb 1700       | SATA Disk Drive      |        120 |
|         2 | Moto Razr                 | Mobile Phone|        200 |
|         6 | Buffalo AirStation Turbo G | Wireless Ethernet Bridge|      60 |
|         7 | Apple iPod Touch          | Portable Music/
                                          Movie Player                    | 199 |
|         8 | Apple iPhone 8Gb          | Smart Phone                       | 399 |
+-----------+---------------------------+---------------------------+------------+
5 rows in set (0.00 sec)
```

GROUPING DATA

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

GROUP BY Syntax

***SELECT* column_name(s)*FROM* table_name**
***WHERE* condition**
***GROUP BY* column_name(s)**
***ORDER BY* column_name(s);**

SQL GROUP BY Examples

The following SQL statement lists the number of customers in each country:

Example
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;

SUB QUERIES

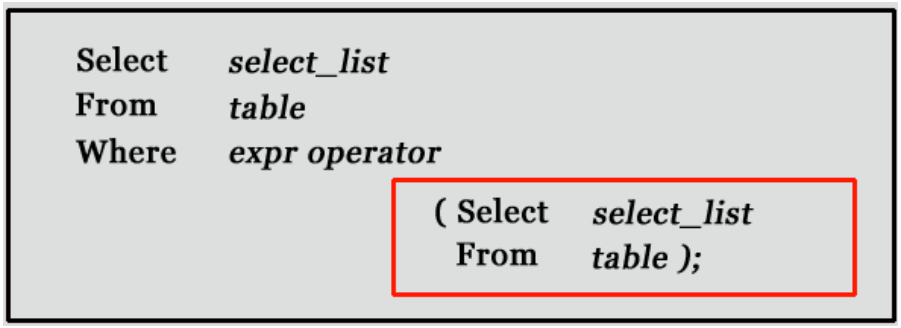
A subquery is a SQL query nested inside a larger query.

A subquery may occur in:

- - A SELECT clause
- - A FROM clause
- - A WHERE clause

- In MySQL subquery can be nested inside a SELECT, INSERT, UPDATE, DELETE, SET, or DO statement or inside another subquery.
- A subquery is usually added within the WHERE Clause of another SQL SELECT statement.
- We can use the comparison operators, such as >, <, or =. The comparison operator can also be a multiple-row operator, such as IN, ANY, SOME, or ALL.
- A subquery can be treated as an inner query, which a SQL query placed as a part of another query is called as outer query.
- The inner query executes first before its parent query so that the results of the inner query can be passed to the outer query.

Subquery Syntax



- The subquery (inner query) executes once before the main query (outer query) executes.
- The main query (outer query) use the subquery result.

Subquery syntax as specified by the SQL standard and supported in MySQL

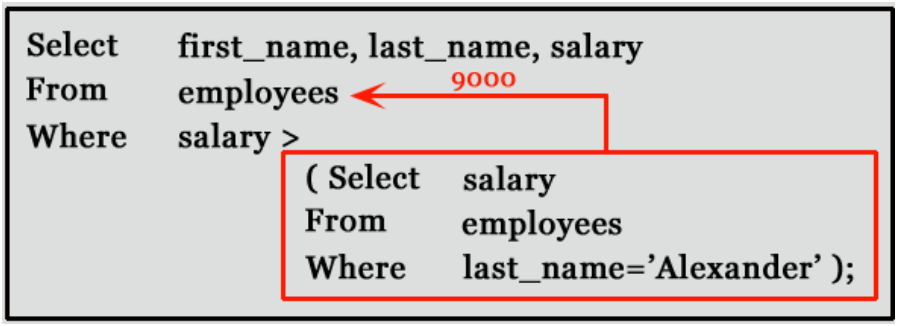
```
DELETE FROM t1
   WHERE s11 > ANY
   (SELECT COUNT(*) /* no hint */ FROM t2
```

```
WHERE NOT EXISTS
 (SELECT * FROM t3
 WHERE ROW(5*t2.s1,77)=
   (SELECT 50,11*s1 FROM t4 UNION SELECT 50,77 FROM
    (SELECT * FROM t5) AS t5)));
```

A subquery can return a scalar (a single value), a single row, a single column, or a table (one or more rows of one or more columns). These are called scalar, column, row, and table subqueries.

MySQL Subquery Example

Using a subquery, list the name of the employees, paid more than 'Alexander' from emp_details .




```
mysql> SELECT first_name,last_name, salary FROM emp_details
      WHERE salary >(SELECT salary FROM emp_details
                  WHERE first_name='Alexander');
+------------+-----------+----------+
| first_name | last_name | salary |
+------------+-----------+----------+
| Steven     | King      | 24000.00 |
| Neena      | Kochhar | 17000.00 |
| Lex        | De Haan | 17000.00 |
| RABI       | Chandra | 15000.00 |
| Ana        | King      | 17000.00 |
+------------+-----------+----------+
```

5 rows in set (0.00 sec)

Subqueries: Guidelines

There are some guidelines to consider when using subqueries:

- A subquery must be enclosed in parentheses.
- Use single-row operators with single-row subqueries, and use multiple-rowoperators with multiple-row subqueries.
- If a subquery (inner query) returns a null value to the outer query, the outer query will not return any rows when using certain comparison operators in a WHERE clause.

TYPES OF SUBQUERIES

- The Subquery as Scalar Operand
- Comparisons using Subqueries
- Subqueries with ALL, ANY, IN, or SOME
- Row Subqueries
- Subqueries with EXISTS or NOT EXISTS
- Correlated Subqueries
- Subqueries in the FROM Clause

MySQL Subquery as Scalar Operand

A scalar subquery is a subquery that returns exactly one column value from one row. A scalar subquery is a simple operand, and you can use it almost anywhere a single column value or literal is legal. If the subquery returns 0 rows then the value of scalar subquery expression in NULL and if the subquery returns more than one row thenMySQL returns an error.

There is some situation where a scalar subquery cannot be used. If a statement permits only a literal value, you cannot use a subquery. For example, LIMIT requires literal integer arguments, and LOAD DATA INFILE requires a literal string file name. You cannot use subqueries to supply these values.

Example: MySQL Subquery as Scalar Operand

```
mysql> SELECT employee_id, last_name,
(CASE WHEN department_id=(
SELECT department_id from departments WHERE
location_id=2500)
THEN 'Canada' ELSE 'USA' END)
location FROM employees;
+-------------+-------------+----------+
| employee_id | last_name | location |
+-------------+-------------+----------+
|         100 | King        | USA      |
|         101 | Kochhar     | USA      |
|         102 | De Haan     | USA      |
|         103 | Hunold      | USA      |
|         104 | Ernst       | USA      |
|         105 | Austin      | USA      |
| - - - - - - - - - - - - - - - - - - -|
| - - - - - - - - - - - - - - - - - - -|
107 rows in set (0.00 sec)
```

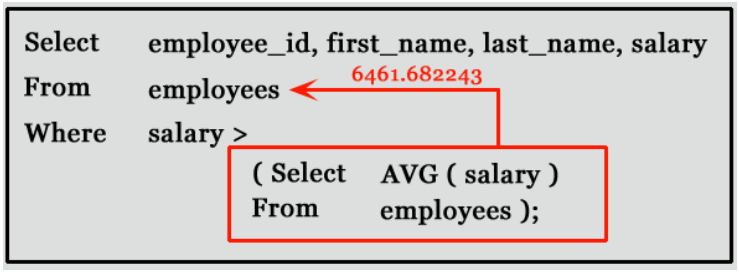
MySQL Subqueries: Using Comparisons

A subquery can be used before or after any of the comparison operators. The subquery can return at most one value. The value can be the result of an arithmetic expression or a column function. SQL then compares the value that results from the subquery with the value on the

other side of the comparison operator. You can use the following comparison operators:

Operator	Description
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
!=	Not equal to
<>	Not equal to
<=>	NULL-safe equal to operator

For example, suppose you want to find the employee id, first_name, last_name, and salaries for employees whose average salary is higher than the average salary throughout the company.




```
mysql> SELECT employee_id,first_name,last_name,salary
      FROM employees WHERE salary >
                  (SELECT AVG(SALARY) FROM employees);
+-------------+-------------+------------+----------+
| employee_id | first_name |last_name | salary  |
+-------------+-------------+------------+----------+
|         100 | Steven      | King       | 24000.00 |
|         101 | Neena       | Kochhar    | 17000.00 |
|         102 | Lex         | De Haan    | 17000.00 |
|         103 | Alexander | Hunold| 9000.00 |
|         108 | Nancy       | Greenberg | 12000.00 |
```

```
|        109 | Daniel      | Faviet| 9000.00 |
|        120 | Matthew     | Weiss      | 8000.00 |
|        121 | Adam        | Fripp| 8200.00 |
|        122 | Payam       | Kaufling| 7900.00 |
|- - - - - - - - - - - - - - - - - - - - - - - - -|
|- - - - - - - - - - - - - - - - - - - - - - - - -|
+-------------+-------------+------------+----------+
51 rows in set (0.00 sec)
```

MySQL Subqueries with ALL, ANY, IN, or SOME

We can use a subquery after a comparison operator, followed by the keyword ALL, ANY, or SOME.

a. ALL Operator

The ALL operator compares value to every value returned by the subquery. Therefore, ALL operator (which must follow a comparison operator) returns TRUE if thecomparison is TRUE for ALL of the values in the column that the subquery returns.

Syntax

operand comparison_operator ALL (subquery)

NOT IN is an alias for <> ALL. Thus, these two statements are the same:

Code

```
SELECT c1 FROM t1 WHERE c1 <>ALL(SELECT c1 FROM t2);

SELECT c1 FROM t1 WHERE c1 NOTIN(SELECT c1 FROM t2);
```

Example: MySQL Subquery, ALL operator

The following query selects the department with the highest average salary. The subquery finds the average salary for each department, and then the main query selects the department with the highest average salary.

```
mysql> SELECT department_id, AVG(SALARY)
FROM EMPLOYEES GROUP BY department_id
HAVING AVG(SALARY)>=ALL
(SELECT AVG(SALARY) FROM EMPLOYEES GROUP BY
department_id);
+---------------+--------------+
| department_id | AVG(SALARY)  |
+---------------+--------------+
|            90 | 19333.333333 |
+---------------+--------------+
1 row in set (0.00 sec)
```

Note: Here we have used ALL keyword for this subquery as the department selected by the query must have an average salary greater than or equal to all the average salaries of the other departments.

b. ANY Operator

The ANY operator compares the value to each value returned by the subquery. Therefore, ANY keyword (which must follow a comparison operator) returns TRUE if the comparison is TRUE for ANY of the values in the column that the subquery returns.

Syntax

operand comparison_operator ANY (subquery)

Example

MySQL Subquery, ANY operator

The following query selects any employee who works in the location 1800. The subquery finds the department id in the 1800 location, and then the main query selects the employees who work in any of these departments.

Departments table

```
mysql> SELECT first_name, last_name,department_id
FROM employees WHERE department_id= ANY
(SELECT DEPARTMENT_ID FROM departments WHERE
location_id=1800);
+------------+-----------+---------------+
| first_name | last_name | department_id |
+------------+-----------+---------------+
| Michael    | Hartstein |            20 |
| Pat        | Fay       |            20 |
+------------+-----------+---------------+
2 rows in set (0.00 sec)
```

Note: We have used ANY keyword in this query because it is likely that the subquery will find more than one department in 1800 location. If you use the ALL keyword instead of the ANY keyword, no data is selected because no employee works in all departments‘s of 1800 location

c. IN Operator

When used with a subquery, the word IN (equal to any member of the list) is an alias for = ANY. Thus, the following two statements are the same:

Code

```
SELECT c1 FROM t1 WHERE c1 =ANY(SELECT c1 FROM t2);

SELECT c1 FROM t1 WHERE c1 IN(SELECT c1 FROM t2);
```

Copy

The word SOME is an alias for ANY. Thus, these two statements are the same:

Code

```
SELECT c1 FROM t1 WHERE c1 <>ANY(SELECT c1 FROM
t2);

SELECT c1 FROM t1 WHERE c1 <>SOME(SELECT c1 FROM
t2);
```

MySQL Row Subqueries

A row subquery is a subquery that returns a single row and more than one column value. You can use = , >, <, >=, <=, <>, !=, <=> comparison operators. See the following examples:

Code

```
SELECT*FROM table1 WHERE(col1,col2)=(SELECT col3,
col4 FROM table2 WHERE id =10);

SELECT*FROM table1 WHEREROW(col1,col2)=(SELECT
col3, col4 FROM table2 WHERE id =10);
```

For both queries,

- if the table table2 contains a single row with id = 10, the subquery returns a single row. If this row has col3 and col4 values equal to the col1 and col2 values of any rows in table1, the WHERE expression is TRUE and each query returns those table1 rows.
- If the table2 row col3 and col4 values are not equal the col1 and col2 values of any table1 row, the expression is FALSE and the query returns an empty result set. The expression is unknown (that is, NULL) if the subquery produces no rows.
- An error occurs if the subquery produces multiple rows because a row subquery can return at most one row.

Example: MySQL Row Subqueries

In the following examples, queries showsdifferent result according to above conditions:

employees table:

```
mysql> SELECT first_name
FROM employees
WHERE ROW(department_id, manager_id) = (SELECT
department_id, manager_id FROM departments WHERE
location_id = 1800);
+------------+
| first_name |
+------------+
| Pat        |
+------------+
1 row in set (0.00 sec)
```

Code

```
mysql>SELECT first_name FROM employees WHERE ROW
(department_id,manager_id)=(SELECTdepartment_id,manager_id FROM
departments WHERE location_id=2800);

Empty set(0.00 sec)
```

Code

```
mysql>SELECT first_name FROM employees WHERE
ROW(department_id,manager_id)=(SELECTdepartment_id, manager_id
FROM departments WHERE location_id=1700);

ERROR 1242(21000): Subquery returns more than 1row
```

MySQL Subqueries with EXISTS or NOT EXISTS

The EXISTS operator tests for the existence of rows in the results set of the subquery. If a subquery row value is found, EXISTS subquery is TRUE and in this case NOT EXISTS subquery is FALSE.

Syntax

SELECT column1 FROM table1 WHERE EXISTS (SELECT * FROM table2);

In the above statement, if table2 contains any rows, even rows with NULL values, the EXISTS condition is TRUE. Generally, an EXISTS subquery starts with SELECT *, but it could begin with SELECT 'X', SELECT 5, or SELECT column1 or anything at all. MySQL ignores the SELECT list in such a subquery, so it makes no difference.

Example: MySQL Subqueries with EXISTS

From the following tables (employees) find employees (employee_id, first_name, last_name, job_id, department_id) who have at least one person reporting to them.

employees table:

```
SELECT employee_id, first_name, last_name, job_id, department_id
FROM employees E
WHERE EXISTS (SELECT * FROM employees WHERE manager_id =
E.employee_id);
+-------------+------------+-----------+---------+---------------+
| employee_id | first_name | last_name | job_id |department_id |
+-------------+------------+-----------+---------+---------------+
|         100 | Steven     | King              | AD_PRES |          90 |
|         101 | Neena      | Kochhar           | AD_VP   |          90 |
|         102 | Lex        | De Haan           | AD_VP   |          90 |
|         103 | Alexander  |Hunold             | IT_PROG |          60 |
```

108	Nancy	Greenberg	FI_MGR	100
114	Den	Raphaely	PU_MAN	30
120	Matthew	Weiss	ST_MAN	50
121	Adam	Fripp	ST_MAN	50

18 rows in set (0.02 sec)

Example: MySQL Subqueries with NOT EXISTS

NOT EXISTS subquery almost always contains correlations. Here is an example:From the following table (departments and employees) find all departments (department_id, department_name) that do not have any employees.

departments table:

```
mysql> SELECT department_id, department_name
FROM departments d
WHERE NOT EXISTS (SELECT * FROM employees WHERE
department_id = d.department_id);
```

department_id	department_name
120	Treasury
130	Corporate Tax
140	Control And Credit
150	Shareholder Services
160	Benefits
170	Manufacturing
180	Construction
190	Contracting
200	Operations

```
+---------------+---------------------+
16 rows in set (0.00 sec)
```

MySQL Correlated Subqueries

A correlated subquery is a subquery that contains a reference to a table (in the parent query) that also appears in the outer query. MySQL evaluates from inside to outside.

Correlated subquery syntax:

SELECT column1, column2,…
FROM table2 outerr
WHERE column1 operator
FROM table2WHERE expr1 –Outer.expr2);

Example - 1: MySQL Correlated Subqueries

Following query find all employees who earn more than the average salary in their department.

employees table:

```
mysql> SELECT last_name, salary, department_id FROM employees outer
WHERE salary > (SELECT AVG(salary) FROM employees WHERE department_id = outer.department_id);
+-----------+----------+---------------+
| last_name | salary | department_id |
+-----------+----------+---------------+
| King      | 24000.00 |          90 |
| Hunold|  9000.00 |           60 |
| Ernst     |  6000.00 |           60 |
| Greenberg | 12000.00 |          100 |
| Faviet|  9000.00 |          100 |
```

| Raphaely | 11000.00 | 30 |
| Weiss | 8000.00 | 50 |
Fripp	8200.00	50
+----------+----------+---------------+
38 rows in set (0.02 sec)

4.8.6 MySQL Subqueries in the FROM Clause

Subqueries work in a SELECT statement's FROM clause.

Syntax

SELECT ... FROM (subquery) [AS] name ...

Every table in a FROM clause must have a name, therefore the [AS] name clause is mandatory. Any columns in the subquery select list must have unique names.

Example

MySQL Subqueries in the FROM Clause

We have the following table tb1.

mysql> CREATE TABLE tb1 (c1 INT, c2 CHAR(5), c3 FLOAT); Query OK, 0 rows affected (0.73 sec)

Let insert some values into tb1.

mysql> INSERT INTO tb1 VALUES (1, '1', 1.0);
Query OK, 1 row affected (0.11 sec)
mysql> INSERT INTO tb1 VALUES (2, '2', 2.0);
Query OK, 1 row affected (0.07 sec)
mysql> INSERT INTO tb1 VALUES (3, '3', 3.0);
Query OK, 1 row affected (0.03 sec)

mysql> select * from tb1;

```
+------+------+------+
| c1 | c2 | c3  |
+------+------+------+
|   1 | 1   |   1 |
|   2 | 2   |   2 |
|   3 | 3   |   3 |
+------+------+------+
3 rows in set (0.00 sec)
```

Here is how to use a subquery in the FROM clause, using the example table (tb1) :

```
mysql> SELECT sc1, sc2, sc3
FROM (SELECT c1 AS sc1, c2 AS sc2, c3*3 AS sc3 FROM tb1) AS sb
WHERE sc1 > 1;
+------+------+------+
| sc1 | sc2 | sc3  |
+------+------+------+
|   2 | 2   |   6 |
|   3 | 3   |   9 |
+------+------+------+
2 rows in set (0.02 sec)
```

JOINING TABLES

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Let's look at a selection from the "Orders" table:

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

Then, look at a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Country
2	Raju	Jothi	Germany
37	Priya	Muruga	India
77	Mani	Ponmalar	India

Notice that the "CustomerID" column in the "Orders" table refers to the "CustomerID" in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.

Then, we can create the following SQL statement (that contains an INNER JOIN), that selects records that have matching values in both tables:

EXAMPLE

```
SELECT Orders.OrderID, Customers.CustomerName,
Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.Custo
merID;
```

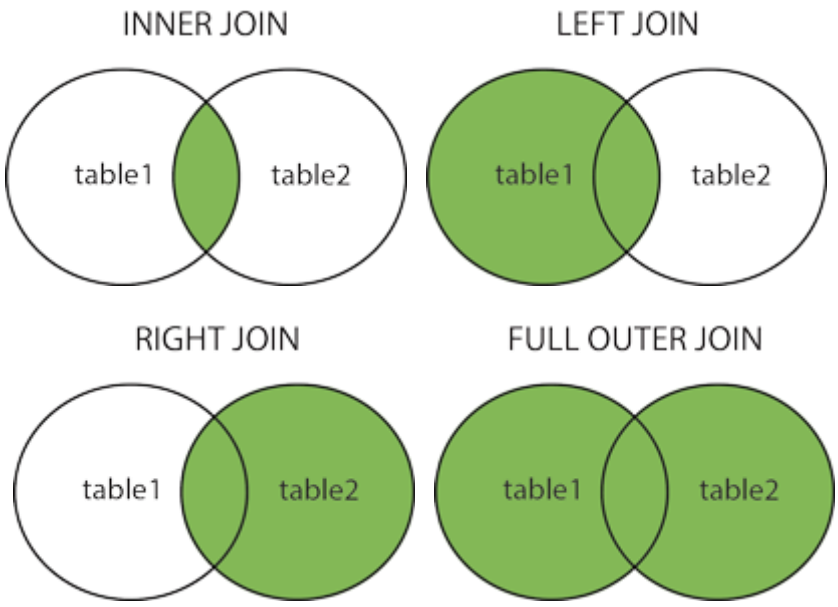
and it will produce something like this:

OrderID	CustomerName	OrderDate
10308	Raju	9/18/1996
10365	Priya	11/27/1996
10383	Mani	12/16/1996

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL

- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table

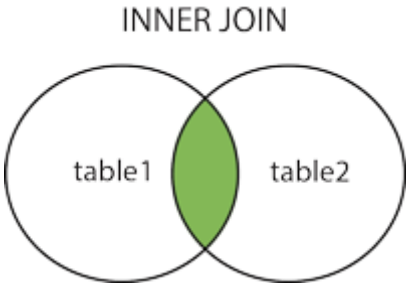


SQL INNER JOIN Keyword

The INNER JOIN keyword selects records that have matching values in both tables.

INNER JOIN Syntax

SELECT *column_name(s)*
FROM *table1*
INNER JOIN *table2*
ON *table1.column_name = table2.column_name*;



The following SQL statement selects all orders with customer information:

Example

SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID =
Customers.CustomerID;

Note: The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns. If there are records in the "Orders" table that do not have matches in "Customers", these orders will not be shown!

JOIN Three Tables

The following SQL statement selects all orders with customer and shipper information:

Example

SELECT Orders.OrderID, Customers.CustomerName,
Shippers.ShipperName

```
FROM ((Orders
INNER JOIN Customers ON Orders.CustomerID =
Customers.CustomerID)
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);
```

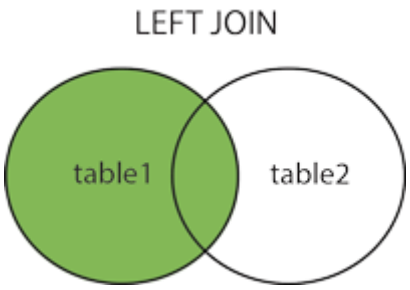
SQL LEFT JOIN

The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.

LEFT JOIN Syntax

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

Note: In some databases LEFT JOIN is called LEFT OUTER JOIN.



SQL LEFT JOIN Example

The following SQL statement will select all customers, and any orders they might have:

Example

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID =
```

```
Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Note: The LEFT JOIN keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders).

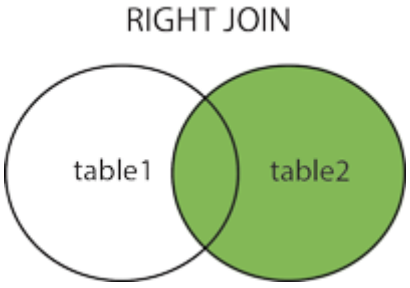
SQL RIGHT JOIN

The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.

RIGHT JOIN Syntax

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

***Note*:** In some databases RIGHT JOIN is called RIGHT OUTER JOIN.



SQL RIGHT JOIN Example

The following SQL statement will return all employees, and any orders they might have placed:

```
SELECT Orders.OrderID, Employees.LastName,
Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID =
```

```
Employees.EmployeeID
ORDER BY Orders.OrderID;
```

Note: The RIGHT JOIN keyword returns all records from the right table (Employees), even if there are no matches in the left table (Orders).

SQL FULL OUTER JOIN

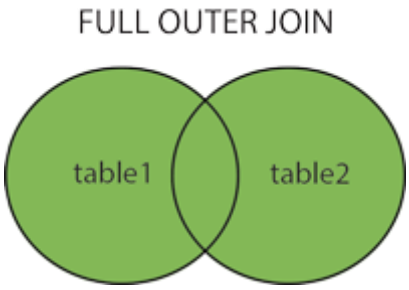
The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

Note: FULL OUTER JOIN can potentially return very large result-sets!

Tip: FULL OUTER JOIN and FULL JOIN are the same.

FULL OUTER JOIN Syntax

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```



SQL FULL OUTER JOIN Example

The following SQL statement selects all customers, and all orders:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.C
```

```
ustomerID
ORDER BY Customers.CustomerName;
```

Note: The FULL OUTER JOIN keyword returns all matching records from both tables whether the other table matches or not. So, if there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

4.10.5. SQL Self JOIN

A self JOIN is a regular join, but the table is joined with itself.

Self JOIN Syntax

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

T1 and *T2* are different table aliases for the same table.

SQL Self JOIN Example

The following SQL statement matches customers that are from the same city:

```
SELECT A.CustomerName AS CustomerName1,
B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID<>B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

SET OPERATORS

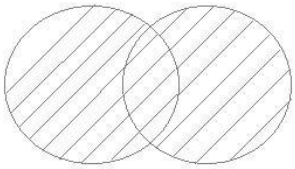
SQL supports few Set operations which can be performed on the table data. These are used to get meaningful results from data stored in the table, under different special conditions.

In this tutorial, we will cover 4 different types of SET operations, along with example:

1. UNION
2. UNION ALL
3. INTERSECT
4. MINUS

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.



Example of UNION

The **First** table,

ID	Name
1	abhi
2	adam

The **Second** table,

ID	Name
2	adam
3	Chester

Union SQL query will be,

SELECT * FROM First

UNION

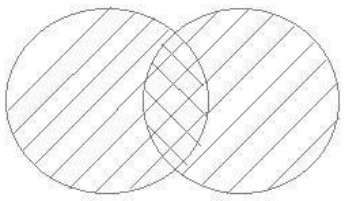
SELECT *FROM Second;

The resultset table will look like,

ID	NAME
1	abhi
2	adam
3	Chester

UNION ALL

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



Example of Union All

The **First** table,

ID	NAME
1	abhi
2	adam

The **Second** table,

ID	NAME
2	Adam
3	Chester

Union All query will be like,

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

The result set table will look like,

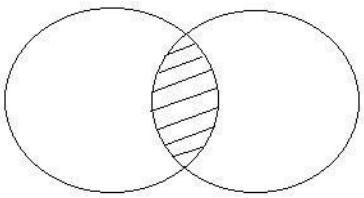
ID	NAME
1	Abhi
2	Adam
2	Adam
3	Chester

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 data type must be same.

NOTE: MySQL does not support INTERSECT operator.



EXAMPLE OF INTERSECT

The First table

ID	NAME
1	abhi
2	adam

The Second table

ID	NAME
2	adam
3	Chester

Intersect query will be

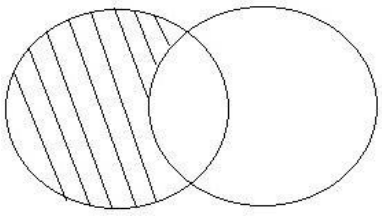
```
SELECT *FROM First
INTERSECT
SELECT *FROM Second;
```

The result set table will look like

ID	NAME
2	adam

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.



Example

The First table

ID	NAME
1	Abhi
2	adam

The Second table,

ID	NAME
2	adam
3	Chester

Minus query will be

SELECT * FROM First

MINUS

SELECT * FROM Second;

The result set table will look like

ID	NAME
1	abhi

Data Manipulation Language (DML)

Data Manipulation Language (DML) statements or commands are used for managing data within tables. Some commands of DML are:

Some commands of DML are:

- SELECT – retrieve data from the a database
- INSERT – insert data into a table
- UPDATE – updates existing data within a table
- DELETE – deletes all records from a table, the space for the records remain
- MERGE – UPSERT operation (insert or update)
- CALL – call a PL/SQL or Java subprogram
- LOCK TABLE – control concurrency

INSERT

The insert statement is used to add new row to a table.

Syntax

INSERT INTO <table name> VALUES (<value 1>…..< value n>);

Example

INSERT INTO STUDENT VALUES(1001,‘Ram‘);

The inserted values must match the table structure exactly in the number of attributes and the data type of each attribute. Character type values are always enclosed in single quotes; number values are never in quotes;data values are often (but not always) in the format _yyyy-mm-dd‘

(for example. _2006-11-30‘);

UPDATE

The update statement is used to change values that are already in a table.

UPDATE <table name> SET <attribute>=<expression>WHERE <condition>;

Example

UPDATE STUDENT SET Name=‘Amar‘ WHERE StudID=1001;

Theupdate expression can be a constant, any computed value, or even the result of a SELECT statement that returns a single row and a single column.

DELETE STATEMENT
The delete statement deletes row(s) from a table.

Syntax
DELETE FROM <table name> WHERE <condition>;

EXAMPLE

DELETE FROM STUDENT WHERE Studid=1001;

If the WHERE clause is omitted, then every row of the table is deleted that matches with thespecified condition.

SELECT STATEMENT

The SELECT statement is used to form queries for w=extracting information out of the database.

Syntax
SELECT <attribute>…..<attribute n> FROM <table name>;

Example
SELECT StudID, Name FROM STUDENT;

The SQL CREATE TABLE Statement

The CREATE TABLE statement is used to create a new table in a database.

SYNTAX

```
CREATE TABLE table_name (
   column1 datatype,
   column2 datatype,
   column3 datatype,
  ....
);
```

The column parameters specify the names of the columns of the table.

The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

SQL CREATE TABLE EXAMPLE

The following example creates a table called "Persons" that contains five columns: PersonID, LastName, FirstName, Address, and City:

```
CREATE TABLE Persons (
    PersonID int,
    LastNamevarchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

The PersonID column is of type int and will hold an integer.

The LastName, FirstName, Address, and City columns are of type varchar and will hold characters, and the maximum length for these fields is 255 characters.

The empty "Persons" table will now look like this:

Tip: The empty "Persons" table can now be filled with data with the SQL INSERT INTO statement.

Create Table Using Another Table

A copy of an existing table can also be created using CREATE TABLE.

The new table gets the same column definitions. All columns or specific columns can be selected.

If you create a new table using an existing table, the new table will be filled with the existing values from the old table.

Syntax

CREATE TABLE *new_table_name* AS
SELECT *column1, column2,...*
FROM *existing_table_name*
WHERE.... ;

The following SQL creates a new table called "TestTables" (which is a copy of the "Customers" table):

CREATE TABLE TestTable AS
SELECT customername, contactname
FROM customers;

The SQL INSERT INTO Statement

The INSERT INTO statement is used to insert new records in a table.

Syntax

It is possible to write the INSERT INTO statement in two ways.

The first way specifies both the column names and the values to be inserted:

INSERT INTO *table_name* (*column1, column2, column3, ...*)
VALUES (*value1, value2, value3,...*);

If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. The INSERT INTO syntax would be as follows:

***INSERT INTO* table_name**
VALUES (*value1, value2, value3, ...*);

INSERT INTO Example

The following SQL statement inserts a new record in the "Customers" table:

INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');

The SQL UPDATE Statement

The UPDATE statement is used to modify the existing records in a table.

UPDATE Syntax

UPDATE *table_name*
SET *column1 = value1, column2 = value2, ...*
WHERE *condition*;

Note: Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated!

UPDATE Table

The following SQL statement updates the first customer (CustomerID = 1) with a new contact person *and* a new city.

EXAMPLE

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;
```

The SQL DELETE Statement

The DELETE statement is used to delete existing records in a table.

Syntax

DELETE FROM *table_name* WHERE *condition*;

Note: Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

SQL DELETE Example

The following SQL statement deletes the customer "AlfredsFutterkiste" from the "Customers" table:

DELETE FROM Customers WHERE CustomerName='AlfredsFutterkiste';

Select Data/ Retrieved Data From a MySQL Database

The SELECT statement is used to select data from one or more tables:

SELECT column_name(s) FROM table_name

or we can use the * character to select ALL columns from a table:

SELECT * FROM table_name

FULL TEXT SEARCH

Full-Text Search in MySQL server lets users run full-text queries against character-based data in MySQL tables. You must create a full-text

index on the table before you run full-text queries on a table. The full-text index can include one or more character-based columns in the table.

FULLTEXT is the index type of full-text index in MySQL.

InnoDB or MyISAM tables use Full-text indexes.

Full-text indexes can be created only for VARCHAR, CHAR or TEXT columns.

A FULLTEXT index definition can be given in the CREATE TABLE statement or can be added later using ALTER TABLE or CREATE INDEX.

Large data sets without FULLTEXT index is much faster to load data into a table than to load data into a table which has an existing FULLTEXT index. Therefore create the index after loading data.

Syntax

MATCH (col1,col2,col3...) AGAINST (expr [search_modifier])

- col1, col2, col3 - Comma-separated list that names the columns to be searched
- AGAINST() takes a string to search, and an optional modifier that indicates what type of search to perform.
- The search string must be a string value. The value is constant during query evaluation.

There are three types of full-text searches:

- Natural Language Full-Text Searches
- Boolean Full-Text searches
- Query expansion searches

Note: Some words are ignored in full-text searches.

- The minimum length of the word for full-text searches as of follows:
    - Three characters for InnoDB search indexes.
    - Four characters for MyISAM search indexes.
- Stop words are words that are very common such as 'on', 'the' or 'it', appear in almost every document. These types of words are ignored during searching.

Natural Language Full-Text Searches

Natural language full-text search interprets the search string as a free text (natural human language) and no special operators are required.

Full-text searches are natural language searches if the IN NATURAL LANGUAGE MODE modifier (see the following syntax) is given or not.

MATCH() function searches a string against a text collection (A set of one or more columns included in a FULLTEXT index.).

For each row in the table, MATCH() returns a relevance value; that is, a similarity measure between the search string (given as the argument to AGAINST() function) and the text in that row in the columns named in the MATCH() list.

The basic format of a natural Language null-text searches mode query is as follows:

Code

```
SELECT*FROMtable_nameWHEREMATCH(col1, col2)
AGAINST('search terms'INNATURALLANGUAGEMODE)
```

Example

```
mysql> CREATE TABLE tutorial (
```

```
id INT UNSIGNED AUTO_INCREMENT NOT NULL
PRIMARY KEY,
title VARCHAR(200),
description TEXT,
FULLTEXT(title,description)
) ENGINE=InnoDB;
```

Query OK, 0 rows affected (2.40 sec)

Let search the string 'left right' in description field:

```
mysql> SELECT * FROM tutorial WHERE
MATCH(title,description) AGAINST ('left right' IN NATURAL
LANGUAGE MODE);
```

By default, the search is case-insensitive. To perform a case-sensitive full-text search, use a binary collation for the indexed columns. For example, a column that uses the latin1 character set of can be assigned a collation of latin1_bin to make it case sensitive for full-text searches.

When MATCH() is used in a WHERE clause, as in the example shown earlier, the rows returned are automatically sorted with the highest relevance first.

- Relevance values are nonnegative floating-point numbers.
- Zero relevance means no similarity.
- Relevance is computed based on -
    - the number of words in the row
    - the number of unique words in that row
    - the total number of words in the collection
    - the number of documents (rows) that contain a particular word.

The following example shows how to retrieve the relevance values explicitly:

```
mysql> SELECT id, MATCH(title,description) AGAINST ('left
right' IN NATURAL LANGUAGE MODE) AS score FROM
tutorial;
+----+--------------------+
| id | score              |
+----+--------------------+
|  1 |                  0 |
|  2 |                  0 |
| 3 | 0.45528939366340637 |
|  4 |                  0 |
| 5 | 0.8331640362739563 |
|  6 |                  0 |
+----+--------------------+
6 rows in set (0.00 sec)
```

Count matches

To count matches, you can use a query like this:

```
mysql> SELECT COUNT(*) FROM tutorial WHERE
MATCH(title,description) AGAINST ('left right' IN NATURAL
LANGUAGE MODE);
+----------+
| COUNT(*) |
+----------+
|        2 |
1 row in set (0.03 sec)
```

Boolean Full-Text Searches

A boolean search interprets the search string using the rules of a special query language.

The string contains the words to search for. It can also contain operators that specify requirements such that a word must be present or absent in matching rows, or that it should be weighted higher or lower than usual.

Certain common words (stopwords) are omitted from the search index and do not match if present in the search string.

MySQL can perform boolean full-text searches using the IN BOOLEAN MODE modifier. With this modifier, certain characters have special meaning at the beginning or end of words in the search string.

The basic format of a boolean mode query is as follows:

Code

```
SELECT*FROMtable_nameWHEREMATCH(col1, col2)
AGAINST('search terms'INBOOLEANMODE)
```

Characteristics of Boolean Full-Text searches

Do not use the 50% threshold that applies to MyISAM search indexes.

- Do not automatically sort rows in order of decreasing relevance.
- Boolean queries against a MyISAM search index can work even without a FULLTEXT index.
- The minimum and maximum word length full-text parameters apply:
    - For InnoDB search indexes, innodb_ft_min_token_size and innodb_ft_max_token_size
    - for MyISAM search indexes, ft_min_word_len and ft_max_word_len.
- InnoDB full-text search does not support the use of multiple operators on a single search word.

The boolean full-text search supports the following operators:

Operator	Description	Example
+	A leading plus sign indicates that a word must be present in each row that is returned.	'+join +union' Find rows that contain both words. '+join union' Search rows that contain the word 'join', but rank rows higher if they also contain 'union'
-	A leading minus sign indicates that a particular word must not be present in any of the rows that are returned. The - operator acts only to exclude rows that are otherwise matched by other search terms.	'+join -union' Find rows that contain the word 'join' but not 'union'.
(no operator)	By default, the word is optional, but the rows that contain it are rated higher.	'join -union' Search rows that contain at least one of the two words.
><	These two operators are used to change a word's contribution to the relevance value that is assigned to a row. The > operator increases the contribution and the < operator decreases it.	'+join +(>left <right)' Find rows that contain the words 'join' and 'left' or 'join' and 'right' (in any order), but rank 'join left' higher than 'join right'.
( )	Parentheses group words into subexpressions. Parenthesized groups can be nested.	
~	A leading tilde acts as a negation operator, causing the word's contribution to the row's relevance to be negative.	'+join ~left' Find rows that contain the word 'join', but if the row also contains the word 'left', rate it lower than if row does not.
*	The asterisk serves as the truncation (or wildcard) operator. Unlike the other operators, it is appended to the word to be affected. Words match if	'join*' Find rows that contain words such as 'join', 'joins', 'joining'

	they begin with the word preceding the * operator.	etc.
"	A phrase that is enclosed within double quote (—"‖) characters matches only rows that contain the phrase literally, as it was typed.	"'left join"' Find rows that contain the exact phrase "let join".

Example: Boolean Full-Text Searches

In the following query, the query retrieves all the rows that contain the word 'Joins' but not 'right'.

```
mysql> SELECT * FROM tutorial WHERE MATCH(title,description)
AGAINST ('+Joins -right' IN BOOLEAN MODE);
```

- - - ☙ ☙ ☙ ☙ ☙ - - -

UNIT 05
USING MYSQL AND PHP TOGETHER

Introduction

PHP has included support for MySQL since version 3.*x*, although the procedureto activate this support has varied widely between versions. PHP 4.*x* included a setof MySQL client libraries, which were activated by default. PHP 5.*x* no longerbundles these libraries, however, due to licensing issues, so you need to obtain andinstall them separately. Then, you need to explicitly activate the MySQL extension—ext/mysql—by adding the --with-mysql option to PHP‘s configure script.

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, andthen use PHP to connect to the server, retrieve a set of results, and format them fordisplay on a web page. The sample table used here consists of a single table nameditems, which holds a list of products and their prices. Here are the SQL queriesneeded to create and initialize this table:

CREATE TABLE items (itemID int(11) NOT NULL auto_increment, itemName varchar(255) NOT NULL default '',itemPrice float NOT NULL default '0',PRIMARY KEY (itemID)) TYPE=MyISAM;

INSERT INTO items VALUES (1, 'Paperweight', '3.99');
INSERT INTO items VALUES (2, 'Key ring', '2.99');

INSERT INTO items VALUES (3, 'Commemorative plate', '14.99');

INSERT INTO items VALUES (4, 'Pencils (set of 4)', '1.99');

INSERT INTO items VALUES (5, 'Coasters (set of 3)', '4.99');

You can enter these commands either interactively or noninteractively throughthe MySQL client program. Once entered, run a SELECT query to ensure that thedata has been successfully imported.

mysql>**SELECT * FROM items;**

```
+--------+--------------------+-----------+
| itemID |        itemName | itemPrice |
+--------+--------------------+-----------+
| 1 |      Paperweight             | 3.99 |
| 2 |      Key ring                | 2.99 |
| 3 | Commemorative plate      | 14.99 |
| 4 |      Pencils (set of 4)      | 1.99 |
| 5 |      Coasters (set of 3)      | 4.99 |
+--------+--------------------+-----------+
```

5 rows in set (0.00 sec)

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

```
<html>
<head></head>
<body>
<?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 * 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)
{
        // print HTML table
```

```
	echo '<table width=100% cellpadding=10 cellspacing=0
border=1>';
	echo'<tr><td><b>ID</b></td><td><b>Name</b></td><td><b>Pri
ce</b></td></tr>';
	// iterate over record set
	// print each field
	while($row = mysql_fetch_row($result))
	{
		echo '<tr>';
		echo '<td>' . $row[0] . '</td>';
		echo '<td>' . $row[1] . '</td>';
		echo '<td>' . $row[2] . '</td>';
		echo '</tr>';
	}
echo '</table>';
}
else
{
	// print error message
	echo 'No rows found!';
}
// once processing is complete
// free result set
mysql_free_result($result);
// close connection to MySQL server
mysql_close($connection);
?>
</body>
</html>
```

Managing Database Connections

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

Here are some examples:

```
<?php
// open connection to MySQL server
$connection = mysql_connect('mydbserver', 'guest', 'pass');
// print status message
if ($connection)
{
	echo 'Connected!';
}
```

```
else
{
        echo 'Could not connect!';
}
?>
```

Another example

```
<?php
// open connection to MySQL server
$connection = @mysql_connect('mydbserver', 'guest', 'pass');
// print status message
echo $connection ? 'Connected!' : 'Could not connect!';
?>
```

Normally, the link to the server remains open for the lifetime of the script,and is automatically closed by PHP once the script completes executing. Thatsaid, just as it‘s good manners to close the doors you open, it‘s good programmingpractice to explicitly close the MySQL connection once you finish using it.

This is accomplished by calling the mysql_close() function, which closesthe link and returns the used memory to the system. Here is an example:

```
<?php
// open connection to MySQL server
$connection = @mysql_connect('mydbserver', 'guest', 'pass');
if ($connection)
{
        // close connection
        mysql_close($connection);
}
?>
```

PERFORMING QUERIES

- 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, which accepts a databasename as argument.
- It can optionally also accept a link identifier; if this is notspecified, the function defaults to using the last opened connection. Here‘s anexample of how it may be used:

```
<?php
// select the database "mydb"
mysql_select_db('mydb');
?>
```

- Once the database has been selected, it becomes possible to execute queries onit.
- In PHP, MySQL queries are handled via the mysql_query() function, whichaccepts a query string and a link identifier and sends it to the server represented bythe link identifier.
- If no link identifier is specified, the last opened link is used asthe default. Here is an example:

```
<?php
// execute query
$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 SHOWquery—then mysql_query() returns a resource identifier pointing tothe query‘s result set, or false on failure. The resource identifier can thenbe used to process the records in the result set.

■ If the query is a data manipulation query—for example, an INSERT orUPDATE 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 beused to extract data from the return value of mysql_query().

Processing Result Sets

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

Queries Which Return Data

- For SELECT-type queries, a number of techniques exist to process the returneddata. The simplest is the mysql_fetch_row() function, which returns eachrecord as a numerically indexed PHP array.
- Individual fields within the recordcan then be accessed using standard PHP-array notation.

The following exampleillustrates 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);
?>
```

Queries That Alter Data

- You can also use PHP‘s MySQL API for queries that don‘t return a result set, forexample, INSERT or UPDATE queries.

```
<html>
<head>
<basefont face="Arial">
</head>
<body>
<?php
if (!$_POST['submit'])
{
        // form not submitted
?>
```

```
<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
{
        // get form input
        // escape input values for greater safety
        $name = (trim($_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'];
        // open connection
        $connection = mysql_connect('localhost', 'guest', 'pass') or die
('Unable to connect!');
        // select database
        mysql_select_db('db2') or die ('Unable to select database!');
        // create query
        $query = "INSERT INTO items (itemName, itemPrice) VALUES
('$name', '$price')";
        // execute query
        $result = mysql_query($query) or die (''Error in query: $query. " .
mysql_error());
        // print ID of inserted record
        echo 'New record inserted with ID ' .mysql_insert_id() . '<br \>';
        // print number of rows affected
        echo mysql_affected_rows() . ' record(s) affected';
```

```
    // close connection
    mysql_close($connection);
}
?>
</body>
</html>
```

The previous example has three new functions:

- The mysql_escape_string() function escapes special characters(like quotes) in the user input, so it can be safely entered into the database.If the magic_quotes_gpc setting in your PHP configuration file isenabled, you might need to first call stripslashes() on the user inputbefore calling mysql_escape_string(), to avoid characters gettingescaped twice.
- The mysql_insert_id() function returns the ID generated by theprevious INSERT query (useful only if the table into which the INSERToccurs contains an AUTO_INCREMENT field).
- The mysql_affected_rows() function returns the total number ofrows affected by the last operation.All these functions come in handy when dealing with queries that alter the database.

Handling Errors

- PHP‘s MySQL API also comes with some powerful error-trackingfunctions that can reduce debugging time. Take a look at the following example,which contains a deliberate error in the SELECT query string:

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

- The mysql_errno() function displays the error code returned by MySQLif there‘s an error in your SQL statement, while the mysql_error() functionreturns the actual error message.
- Turn these both on, and you‘ll find they cansignificantly reduce the time you spend fixing bugs.

Setting Input Constraints at the Database Layer

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

Using the NULL Modifier

- MySQL enables you to specify whether a field isallowed to be empty or if it must necessarily be filled with data, by placing theNULL and NOT NULL modifiers after each field definition.
- This is a good wayto ensure that required fields of a record are never left empty, because MySQL willsimply reject entries that do not have all the necessary fields filled in.
- Here‘s anexample of this in action:

mysql>**CREATE TABLE products (->id int(4),->name varchar(50)->);**
Query OK, 0 rows affected (0.06 sec)

Here, the name field can hold NULL values, which means the following INSERT will go unchallenged,

mysql>**INSERT INTO products VALUES (NULL, NULL);**

Query OK, 1 row affected (0.06 sec)
and create the following nonsense entry in the table:
mysql>**SELECT * FROM products;**

```
+------+------+
| id | name |
+------+------+
| NULL | NULL |
+------+------+
```

1 row in set (0.11 sec)

Now, look what happens if you make the name field mandatory:
mysql>**CREATE TABLE products (->id int(4),->name varchar(50) NOT NULL->);**
Query OK, 0 rows affected (0.05 sec)

mysql>**INSERT INTO products VALUES (NULL, NULL);**
ERROR 1048: Column 'name' cannot be null

Of course, because MySQL makes a distinction between a NULL value andan empty string (''), the following record—which is also meaningless—wouldbe accepted.

mysql>**INSERT INTO products VALUES ('', '');**
Query OK, 1 row affected (0.05 sec)

mysql>**SELECT * FROM products;**

+------+------+

| id | name |

+------+------+

| 0 | |

+------+------+

1 row in set (0.00 sec)

Thus, while the NOT NULL modifier can help reduce the incidence of emptyor incomplete records in a database.

Using the UNIQUE Modifier

Using MySQL‘s built-in validation mechanisms has an important advantage: itmakes it easy to perform certain types of validation that would be lengthy andtime-consuming to write code for.

Consider, for example, the situation of ensuringthat a particular field contains only unique values. MySQL makes it possible todo this, simply by attaching a UNIQUE modifier to the field, as in the followingexample:

mysql>**CREATE TABLE users (username VARCHAR(50) NOT NULL UNIQUE);**

Query OK, 0 rows affected (0.06 sec)

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

Query OK, 1 row affected (0.06 sec)

mysql>**INSERT INTO users (username) VALUES ('jon')**;

Query OK, 1 row affected (0.00 sec)

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

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

ERROR 1062: Duplicate entry 'tim' for key 1

Using Field Data Types

- MySQL also requires you to specify the typeof data a particular field can hold at the time of defining a table. Input that does notmatch the named data type is automatically converted into a more acceptable, thoughincorrect, value.

Here‘s an example of this:

mysql>**CREATE TABLE items (->id INT(2) NOT NULL,->price INT(4) NOT NULL->);**

Query OK, 0 rows affected (0.05 sec)

mysql>**INSERT INTO items (id, price) VALUES (1, 'five');**

Query OK, 1 row affected (0.00 sec)

mysql>**SELECT * FROM items;**

```
+----+-------+
| id | price |
+----+-------+
| 1 | 0 |
+----+-------+
```

1 row in set (0.05 sec)

In this case, because the price field has been constrained to only storeintegers, the string *five* has been converted into a *0* and saved.

VALIDATING INPUT AT THE APPLICATION LAYER

- When it comes to catching errors in user input, the best place to do this is at thepoint of entry—the application itself. That‘s why a good part of this chapter isdevoted to showing you techniques you can use to catch common input errors andensure that they don‘t get into your database.

Checking for Required Values

- This can result in a database with numerous emptyrecords, and these empty records can, in turn, affect the accuracy of your queries.

mysql>**CREATE TABLE users (**
->**username varchar(8) NOT NULL DEFAULT '',**
->**password varchar(8) NOT NULL DEFAULT ''**
->**) TYPE=MyISAM;**
Query OK, 0 rows affected (0.05 sec)

When inserting a record into this table, values must be specified for bothusername and password fields (this is reinforced by the use of NOT NULLconstraints on these fields). Here‘s a script that enforces these constraints at the application level

```
<html>
<head>
<basefont face="Arial">
</head>
<body>
<?php
if (!$_POST['submit'])
{
        // form not submitted
?>
<form action="<?=$_SERVER['PHP_SELF']?>" method="post">
Username: <input type="text" name="username">
<br />
Password: <input type="password" name="password">
<br /><br />
<input type="submit" name="submit" value="Sign Up">
```

```
</form>
<?php
}
else
{
        // form submitted
        // check the username field
        $username = ↲
        (!isset($_POST['username']) || trim($_POST['username']) == "")?
        die ('ERROR: Enter a username') :
        mysql_escape_string(trim($_POST['username']));
        // check the password field
        $password = (!isset($_POST['password']) ||
        trim($_POST['password'] == ""))? die ('ERROR: Enter a password')
        : mysql_escape_string(trim($_POST['password']));
        // connect to database
        // open connection
        $connection = mysql_connect('localhost', 'guest', 'pass') or die
('Unable to connect!');
        // select database
        mysql_select_db('db2') or die ('Unable to select database!');
        // create query
        $query = "INSERT INTO users (username, password)VALUES
('$username', '$password')";
        // execute query
        $result = mysql_query($query)or die ("Error in query: $query. " .
mysql_error());
        // close connection
        mysql_close($connection);
}
?>
```

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

Restricting the Size of Input Data

- MySQL enables you to control the length of a particularfield 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 notificationor exception generated to let the user know about the change.
- This is disturbing, because it means that user data can easily get corruptedwithout the user‘s awareness.

```
mysql>CREATE TABLE news (
->id INT (10) NOT NULL,
->title VARCHAR(50) NOT NULL
->);
Query OK, 0 rows affected (0.05 sec)
```

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

```
<html>
<head>
<basefont face="Arial">
</head>
<body>
<?php
if (!$_POST['submit'])
{
// form not submitted
?>
<form action="<?=$_SERVER['PHP_SELF']?>" method="post">
Title: <input type="text" name="title">
<br />
```

```
<input type="submit" name="submit" value="Save">
</form>
<?php
}
else
{
// form submitted
// trim the title field
$title = trim ($_POST['title']);
// check its length
if (strlen($title) > 50)
{
        die ('ERROR: Title contains more than 50 characters');
}
// connect to database
// save record
}
?>
</body>
</html>
```

Checking the Type of Input Data

You‘ve already seen how MySQL automatically ―corrects‖ values that don‘t matchthe data type specified in the table definition. Often, the assumptions MySQL makeswhen performing these corrections aren‘t true, and the corrected (but incorrect)values subsequently affect the integrity of your database. Therefore, an importanttest of user input involves checking the data type of input values against thedatabase‘s expectations, and raising an error in the event of a mismatch.

To see an example of this, consider the following table definition:

```
mysql>CREATE TABLE items (
->itemIDINT(11) NOT NULL AUTO_INCREMENT,
->itemNameVARCHAR(255) NOT NULL DEFAULT '',
->itemSPrice FLOAT NOT NULL DEFAULT '0',
->itemCPrice FLOAT NOT NULL DEFAULT '0',
->itemQuantityINT(11) NOT NULL DEFAULT '0',
->PRIMARY KEY (itemID)
->) TYPE=MyISAM;
Query OK, 0 rows affected (0.07 sec)
```

```
<html>
<head>
<basefont face="Arial">
</head>
<body>
<?php
if (!$_POST['submit'])
{
// form not submitted
?>
<form action="<?=$_SERVER['PHP_SELF']?>" method="post">
Item name:
<br />
<input type="text" name="itemName">
<br />
Item sale price:
<br />
<input type="text" name="itemSPrice">
<br />
Item cost price:
```

```
<br />
<input type="text" name="itemCPrice">
<br />
14
Item quantity:
<br />
<input type="text" name="itemQuantity">
<br/><br />
<input type="submit" name="submit" value="Enter Data">
</form>
<?php
}
else
{
        // form submitted
        // 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');
        }
        elseif(!is_numeric(trim($_POST['itemSPrice'])))
        {
                die ('ERROR: Enter numeric value for the item\'s selling
        price');
        }
        else
```

```
        {
                $itemPrice = floatval(trim($_POST['itemSPrice']));
        }
        // check the itemCPrice field
        if(!isset($_POST['itemCPrice']) || trim($_POST['itemCPrice']) ==
""")
        {
                die ('ERROR: Enter the item\'s cost price');
        }
        elseif (!is_numeric(trim($_POST['itemCPrice'])))
        {
                die ('ERROR: Enter numeric value for the item\'s cost
        price');
        }
        else
        {
                $itemCost = floatval(trim($_POST['itemCPrice']));
        }
        // check the itemQuantity field
        if(!isset($_POST['itemQuantity']) || trim($_POST['itemQuantity'])
== "")
        {
                die ('ERROR: Enter the quantity');
        }
        elseif (!is_numeric(trim($_POST['itemQuantity'])))
        {
                die ('ERROR: Enter numeric value for quantity');
        }
        else
        {
                $itemQuantity = intval(trim($_POST['itemQuantity']));
```

```
        }
        // connect to database
        // save record
}
?>
</body>
</html>
```

Checking for Illegal Input Values

In addition to the tests listed in previous sections, an application‘s particular businesslogic 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 between10 and 99 (both inclusive) and to display an error if it doesn‘t. Take a look at thenext script, which demonstrates what the code for such a validation test wouldlook like:

```
<html>
<head>
<basefont face="Arial">
</head>
<body>
<?php
if (!$_POST['submit'])
{
// form not submitted
?>
<form action="<?=$_SERVER['PHP_SELF']?>" method="post">
Enter any positive two-digit number:
<input type="text" name="num" size="2">
<br />
```

```
<input type="submit" name="submit" value="Check">
</form>
<?php
}
else
{
// form submitted
// 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');
}
}
?>
</body>
</html>
```

This type of custom validation can play an important role in avoiding commonerrors, such as the dreaded division-by-zero error. Harking back to the example inthe previous section, assume you have a table containing the following data,

mysql>**SELECT * FROM items;**

```
+--------+----------+------------+------------+--------------+
| itemID | itemName | itemSPrice | itemCPrice | itemQuantity |
+--------+----------+------------+------------+--------------+
| 1      | Syringe  | 10         | 5          | 200          |
| 2      | Swab     | 1          | 0.25       | 1000         |
```

```
| 3        | Pump            | 95                | 0        | 5 |
+--------+----------+------------+------------+--------------+
3 rows in set (0.00 sec)
```

and you'd like to calculate the percentage profit on each item using the formula
Percentage Profit = (Profit/Cost Price) * 100. You'd probably
need to run a SELECT query like this:

mysql>**SELECT itemName, (((itemSPrice - itemCPrice)/itemCPrice) * 100) AS percentProfit FROM items;**

```
+----------+---------------+
| itemName | percentProfit |
+----------+---------------+
| Syringe          | 100 |
| Swab             | 300 |
| Pump             | NULL |
+----------+---------------+
3 rows in set (0.00 sec)
```

Here's anexample of what that test might have looked like:

```
<html>
<head>
<basefont face="Arial">
</head>
<body>
<?php
if (!$_POST['submit'])
{
// form not submitted
?>
```

```
<form action="<?=$_SERVER['PHP_SELF']?>" method="post">
Item name:
<br />
<input type="text" name="itemName">
<br />
Item sale price:
<br />
<input type="text" name="itemSPrice">
<br />
Item cost price:
<br />
<input type="text" name="itemCPrice">
<br />
Item quantity:
<br />
<input type="text" name="itemQuantity">
<br/><br />
<input type="submit" name="submit" value="Enter Data">
</form>
<?php
}
else
{
// form submitted
// check the itemCPrice field
$itemCost = ( ↲
!isset($_POST['itemCPrice']) || trim($_POST['itemCPrice']) == "") ↲
? die ('ERROR: Enter the item\'s cost price') :↲
(!is_numeric(trim($_POST['itemCPrice']))) ↲
? die ('ERROR: Enter numeric value for the item\'s cost price') :↲
floatval(trim($_POST['itemCPrice']));
```

```
// check if itemCPrice field is equal to zero
if($itemCost == 0)
{
die ('ERROR: Please enter an item cost price greater ↲
than zero');
}
// connect to database
// save record
}
?>
```

VALIDATING DATES

Dates often play an important role in an application‘s business logic, and usersare prone to errors when entering these values. Luckily, PHP comes with acheckdate() function that provides an easy way to validate user-provideddate values.

To see how this works, consider the following simple script, which asks theuser to enter a date, and then tests it for validity:

```
<html>
<head>
<basefont face="Arial">
</head>
<body>
<?php
if (!$_POST['submit'])
{
// form not submitted
?>
<form action="<?=$_SERVER['PHP_SELF']?>" method="post">
Day <input type="text" name="day" size="2">
```

```
Month <input type="text" name="month" size="2">
Year <input type="text" name="year" size="2">
<br />
<input type="submit" name="submit" value="Check">
</form>
<?php
}
else
{
// form submitted
// check date
if (!checkdate($_POST['month'], $_POST['day'], $_POST['year'])) ↲
{
die ('ERROR: Enter a valid date');
}
}
?>
</body>
</html>
```

Validating Multiple-Choice Input

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

```
<html>
<head>
<basefont face="Arial">
</head>
```

```
<body>
<?php
if (!$_POST['submit'])
{
// form not submitted
?>
<form action="<?=$_SERVER['PHP_SELF']?>" method="post">
Username:
<br />
<input type="text" name="username">
<p />
Password:
<br />
<input type="password" name="password">
<p />
Date of Birth:
<br />
Month <input type="text" name="month" size="2">
Day <input type="text" name="day" size="2">
Year <input type="text" name="year" size="4">
<p />
Hobbies (select at least <b>three</b>):
<br />
<input type="checkbox" name="hobbies[]" value="Sports">Sports
<input type="checkbox" name="hobbies[]" value="Reading">Reading
<input type="checkbox" name="hobbies[]" value="Travel">Travel
<input type="checkbox" name="hobbies[]" value="Television">Television
<input type="checkbox" name="hobbies[]" value="Cooking">Cooking
<p />
Subscriptions (Select at least <b>two</b>):
<br />
```

```
<select name="subscriptions[]" multiple>
<option value="General">General Newsletter</option>
<option value="Members">Members Newsletter</option>
<option value="Premium">Premium Newsletter</option>
</select>
<p />
<input type="submit" name="submit" value="Sign Up">
</form>
<?php
}
else
{
// form submitted
// validate "username", "password" and "date of birth" fields
$username = (!isset($_POST['username']) || ↲
trim($_POST['username']) == "") ↲
? die ('ERROR: Enter a username') : trim($_POST['username']);
$password = (!isset($_POST['password']) ↲
|| trim($_POST['password'] == "")) ↲
? die ('ERROR: Enter a password') : trim($_POST['password']);
if (!checkdate($_POST['month'], $_POST['day'], $_POST['year']))
{
die ('ERROR: Enter a valid date');
}
// check the "hobbies" field for valid values
$hobbies = ((sizeof($_POST['hobbies']) < 3) ?↲
die ('ERROR: Please select at least 3 hobbies') :↲
implode(',', $_POST['hobbies']));
// check the "subscriptions" field for valid values
$subscriptions = ((sizeof($_POST['subscriptions']) < 2) ?↲
die ('ERROR: Please select at least 2 subscriptions') :↲
```

```
implode(',', $_POST['subscriptions']));
// connect to database
// save record
}
?>
</body>
</html>
```

Formatting Character Data

- A lot of your MySQL data is going to be stored as strings or text blocks, in CHAR,VARCHAR, or TEXT fields.
- It‘s essential that you know how to manipulate this stringdata and adjust it to fit the requirements of your application user interface.

Concatenating String Values

- String together the variables you want to concatenate using the PHP concatenation
  operation, a period (.).
- Concatenating fields from a MySQL result set is equallysimple—just assign the field values to PHP variables and concatenate the variablestogether in the normal manner.

To see how this works, consider the following table:

mysql>**SELECT * FROM users;**

```
+-----------+---------+----------+
| username | fname | lname |
+-----------+---------+----------+
| matt | Matthew | Johnson |
| har56 | Harry| Thompson |
| kellynoor | Kelly | Noor |
| jimbo2003 | Jim | Doe |
```

```
| x         | Xavier          | Belgudui |
+-----------+---------+----------+
5 rows in set (0.00 sec)
```

Now, assume you need to concatenate the first- and last-name fields into a single value (a common requirement).

```
<html>
<head></head>
<body>
<?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 fname, lname FROM users';
$result = mysql_query($query) or die ('Error in query: $query. ' . mysql_error());
// check if records were returned
if (mysql_num_rows($result) > 0)
{
// print HTML table
echo '<ul>';
// iterate over record set
// print each field
while($row = mysql_fetch_object($result))
{
// prints in format "last-name, first-name"
echo '<li>' . $row->lname . ', ' . $row->fname;
}
```

```
echo '</ul>';
}
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);
?>
</body>
</html>
```

PADDING STRING VALUES

- PHP trim() function, used to strip leading andtrailing white space from string values prior to testing them for validity or insertingthem 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 whitespace or a user-specified character sequence.
- This can come in handy if you needto artificially elongate string values for display or layout purposes.

Here‘s a table containing string values of differing lengths:

mysql>**SELECT * FROM ingredients;**

```
+----------------+
| name |
+----------------+
```

```
| cinnamon |
| ginger |
| red pepper |
| cloves |
| peas |
| tender coconut |
+----------------+
6 rows in set (0.00 sec)
```

And here‘s some PHP code that demonstrates padding them:

```
<html>
<head></head>
<body>
<pre>
<?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 name FROM ingredients";
$result = mysql_query($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($row = mysql_fetch_object($result))
```

```
{
// prints " name"
echo str_pad($row->name, 30, ' ', STR_PAD_LEFT) . '<br />';
}
}
else
{
// print error message
echo 'No rows found!';
}
// once processing is complete
// free result set
mysql_free_result($result);
// close connection to MySQL server
mysql_close($connection);
?>
</pre>
</body>
</html>
```

The str_pad() function takes ***three parameters***:

1. The variable to be padded, thesize it should be padded to, and the character to use for padding.
2. By default, thefunction pads the string on the right side. You can alter this default, however, bypassing one of the constants STR_PAD_LEFT or STR_PAD_BOTH to the functionas an optional fourth parameter.
3. 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.

The following snippets demonstrate how these functions work:

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

```
+----------------------+----------------------+
| RPAD(name, 20,'_') | LPAD(name, 20, '_') |
+----------------------+----------------------+
| cinnamon____________|____________cinnamon |
+----------------------+----------------------+
| ginger______________|______________ginger |
+----------------------+----------------------+
2 rows in set (0.00 sec)
```

A word of caution: if the total length specified in the RPAD() and LPAD()function call is less than the length of the field value, the value will be truncated.

The next snippet illustrates this:

mysql>**SELECT RPAD(name, 5, '_') FROM ingredients WHERE**
name = 'cinnamon';

```
+----------------------+
| RPAD(name, 5, '_') |
+----------------------+
| cinna |
+----------------------+
1 row in set (0.00 sec)
```

Padding string values

PHP‘s str_pad() function, however, does not truncate strings inequivalent situations.

Altering String Case

Four useful functions are here: strtolower(), which converts all characters ina string to lowercase; strtoupper(), which converts all characters to uppercase;ucfirst(), which converts the first character of a string to uppercase, and theuseful ucwords(), which converts the first character of all the words in a stringto uppercase.

The following example demonstrates these functions, using them on thedifferent fields of the following table:

mysql>**SELECT * FROM customers;**

```
+-------+---------+----------------+----------+--------------------------+
| fname | lname | addr                    | city          | email |
+-------+---------+----------------+----------+--------------------------+
| David | Johnson | 18 mcgooplace,ray road | boston | David_Johnson@CORPMAIL.DOM |
| Flora | Bharti | 239/a harkrishnabldg,j b marg | hyderabad| bharti@MyOwnCompany.in |
| joe | cool | 15 hill view,east end road | yorktown | joecool@guess.it |
+-------+---------+----------------+----------+--------------------------+
3 rows in set (0.00 sec)
```

Here‘s the code that reformats all this data to a more consistent casing style:

```
<html>
<head></head>
<body>
<?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 * FROM customers";
$result = mysql_query($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
echo '<table border=1 cellpadding=10>';
echo '<tr><td>Name</td><td>Mailing Address</td>↲
<td>Email Address</td></tr>';
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>';
}
echo '</table>';
}
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);
?>
</body>
</html>
```

In the query string itself, by using
MySQL‘s UCASE() and LCASE() functions. The following snippet illustrates this:

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

```
+--------------------------------+---------------------------+
| address                        | email |
+--------------------------------+---------------------------+
| 18 MCGOO PLACE, RAY ROAD | || BOSTON |
david_johnson@corpmail.dom |
| 239/A HARKRISHNA BLDG, J B MARG | || HYDERABAD|
bharti@myowncompany.in |
| 15 HILL VIEW, EAST END ROAD | || YORKTOWN | joecool@guess.it|
+--------------------------------+---------------------------+
3 rows in set (0.11 sec)
```

Dealing with Special Characters

- Special characters need to be protected,white space and line breaks must be preserved, and potentially malicious HTMLcode must be defanged. PHP comes with a number of functions designed toperform just these tasks.

Repeat Business

MySQL also provides a REPEAT() function, which can be used to display a string field multiple times. Here‘s an example:

```
mysql>SELECT REPEAT('ho ', 5);
+------------------+
| REPEAT('ho ', 5) |
+------------------+
| ho hohohoho |
+------------------+
1 row in set (0.00 sec)
```

PHP‘s equivalent function is the str_repeat() function.

To illustrate, consider a table containing large blocks of text data, like the following one:

```
mysql>SELECT id, data FROM newsdata LIMIT 0,1;
+----+-----------------------------------------------------------------------+
| id | data |
+----+-----------------------------------------------------------------------+
| 1 | Recently, I put together a Web site and the public actually liked |
my <html>&<javascript>. People... |
+----+-----------------------------------------------------------------------+
1 row in set (0.00 sec)
```

Now, here‘s how you‘d normally retrieve and display this information ina web page:

```
<html>
<head></head>
<body>
<font face="Arial" size="-1">
<?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 title, data FROM newsdata";
$result = mysql_query($query) or die ('Error in query: $query. ' .
mysql_error());
// check if records were returned
if (mysql_num_rows($result) > 0)
{
// iterate over record set
while($row = mysql_fetch_object($result))
{
echo '<b>' . $row->title . '</b>';
echo '<p />';
echo $row->data;
echo '<p />';
}
}
else
{
// print error message
echo 'No rows found!';
}
// once processing is complete
// free result set
mysql_free_result($result);
// close connection to MySQL server
mysql_close($connection);
?>
```

```
</font>
</body>
</html>
```

FORMATTING NUMERIC DATA

Both PHP and MySQL come with a full set of functionsto manipulate integer and floating-point numbers, and to format large numericvalues 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().

The number_format() function is used to display large numbers with commaand decimal separators.
It can be used to control both the visibility and the appearanceof the decimal digits, as well as the character used as the thousands separator.

To see how this works, consider the following table:

```
mysql>SELECT accountNumber, accountName, ↵
accountBalance FROM accounts;
+--------------+-------------+---------------+
| accountNumber | accountName | accountBalance |
+--------------+-------------+---------------+
| 1265489921          | James D | 2346.00000 |
| 2147483647          | Timothy J | 56347.50000 |
| 5739304575 | Harish K       | 996564.87500 |
| 2173467271 | Kingston X    | 634238.00000 |
| 2312934021 | Sue U          | 34.67000 |
| 1248954638 | Ila T          | 5373.81982 |
| 2384371001 | Anil V          | 72460.00000 |
```

```
| 9430125467 | Katrina P        | 100.00000 |
| 1890192554 | Pooja B          | 17337.11914 |
| 2388282010 | Sue U            | 388883.12500 |
| 2374845291 | Jacob N          | 18410.00000 |
+---------------+-------------+----------------+
11 rows in set (0.05 sec)
```

Here‘s a PHP script that displays this information on a web page, usingnumber_format() to display account balances with two decimal places andcommas as thousand separators:

```
<html>
<head></head>
<body>
<?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 accountNumber, accountName, accountBalance↲
FROM accounts";
$result = mysql_query($query) ↲
or die ('Error in query: $query. ' . mysql_error());
// check if records were returned
if (mysql_num_rows($result) > 0)
{
echo '<table border=1 cellpadding=10>';
echo '<tr><td>Number</td><td>Name</td><td>Balance</td></tr>';
// iterate over record set
```

```
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>';
}
echo '</table>';
}
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);
?>
</body>
</html>
```

Note:

- You‘ve already used the ***echo() function*** extensively to display output.
- However, echo()doesn‘t let you format output in any significant manner, forexample, you can‘t write 1 as 00001.00. So, another common function used to performthis type of number formatting is

the ***sprintf() function,*** which enables you todefine the format in which data is output.

Consider the following 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 thatC programmers are used to.

To format the output, you need to use *field templates*,templates that represent the format you‘d like to display. Common field templates are listed as below.

Template What It Represents

%s string

%d decimal number

%x hexadecimal number

%o octal number

%f float number

Common Field Templates Supported by the sprintf() Function

Here are a few more examples of sprintf() in action:

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

To see a real-world example of sprintf() usage, consider the followingnumber-heavy MySQL table:

mysql>**SELECT * FROM stocks;**

```
+--------+-------------+------------+------------+------------+-----------+
| symbol | qty         | buy        | sell       | high       | low |
+--------+-------------+------------+------------+------------+-----------+
| HGTY | 17000.0000 | 289.9786 | 195.7474 | 315.7643 | 187.9540|
| HDYS | 5.8701 | 19000.2734 | 21759.6465 | 21759.6465 | 18639.2988|
| IWIK | 2174733.0000 | 868.0000 | 870.0000 | 891.0000 | 800.0000|
+--------+-------------+------------+------------+------------+-----------+
```

3 rows in set (0.00 sec)

Here‘s the PHP script that formats this mass of data into something more readable:

```
<html>
<head></head>
<body>
```

Target Selection

The sprintf() function returns the result of output formatting, while theprintf() function prints the result directly to the standard output device.

```
<?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 * FROM stocks";
$result = mysql_query($query) ↲
or die ('Error in query: $query. ' . mysql_error());
// check if records were returned
if (mysql_num_rows($result) > 0)
{
echo '<table border=1 cellpadding=10>';
echo '<tr><td>Stock</td><td>Purchase value</td>';
echo '<td>Sale value</td><td>Profit/Loss</td>';
echo '<td>High/Low</td></tr>';
// iterate over record set
// format and print numeric data
while($row = mysql_fetch_object($result))
{
echo '<tr>';
echo '<td>' . $row->symbol . '</td>';
printf('<td align=right>%s</td>', ↲
number_format($row->qty * $row->buy));
printf('<td align=right>%s</td>', ↲
number_format($row->qty * $row->sell));
printf('<td align=right>%s</td>', ↲
number_format($row->qty * ($row->sell - $row->buy)));
printf('<td align=right>%s / %s</td>', ↲
number_format($row->high), number_format($row->low));
```

```
echo '</tr>';
}
echo '</table>';
}
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);
?>
</body>
</html>
```

Formatting Currency Values

- This function is designed specifically for use with currency

Rounding Off

If you have a decimal value that you need to round up or down, you can doit using either PHP or MySQL. MySQL offers the CEIL() and FLOOR()functions, while PHP offers the round(), ceil(), and floor() functions.

Take a look at the following examples to see how these functions work:

```
mysql>SELECT CEIL(12.052),FLOOR(12.052);
+--------------+--------------+
| ceil(12.052) | floor(12.052)|
+--------------+--------------+
| 13           | 12|
```

```
+--------------+--------------+
1 row in set (0.00 sec)

<?php
// returns 13
echo ceil(12.052);
// returns 12
echo floor(12.052);
// returns 12.1
// the second argument specifies ↲
// the number of decimals to round to
echo round(12.052, 1);
?>
```

Formatting Dates and Times

PHP offers the date() function,which accepts two arguments: one or more *format specifiers*, which indicates howthe timestamp should be formatted, and the timestamp itself (optional; PHP assumesthe current time if this second argument is not provided).

To see a few examples of the date() function in action, create and run thefollowing script:

```
<?php
// retrieve current date and time
// prints a date and time like "09:18 pm 19 Jun 2004"
echo date("h:i a d M Y", mktime());
// returns just the date "27 April 2003"
echo date("d F Y", mktime(0, 0, 0, 04, 27, 2003));
// returns the time in 24-hr format "21:18"
echo date("H:i", mktime());
?>
```

Let‘s see an example of this in action. Consider the following database table,which holds a list of users and their birth dates:

mysql>**SELECT * FROM birthdays;**

```
+-------+------------+
| name | dob |
+-------+------------+
| raoul | 1978-06-04 |
| luis | 1970-11-17 |
| larry | 1971-08-19 |
| moe | 1992-01-23 |
+-------+------------+
```

4 rows in set (0.00 sec)

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

Common Format Specifiers Supported by the date() Function

Now, create and run a PHP script to retrieve these dates and format them intomore readable values:

```
<html>
<head></head>
```

```
<body>
<?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 name, UNIX_TIMESTAMP(dob) AS dob FROM
birthdays';
$result = mysql_query($query) or die ('Error in query: $query. ' .
mysql_error());
// check if records were returned
if (mysql_num_rows($result) > 0)
{
// print HTML table
echo '<table border=1 cellpadding=10>';
// iterate over record set
// print each field
while($row = mysql_fetch_object($result))
{
echo '<tr>';
echo "<td>$row->name</td><td>" .↲
date("d M Y", $row->dob) . "</td>";
echo '</tr>';
}
echo '</table>';
}
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);
?>
</body>
</html>
```

Table demonstrates the specifiers supported by the DATE_FORMAT() and TIME_FORMAT() functions.

Here are some examples demonstrating these in action:

```
mysql>SELECT DATE_FORMAT(NOW(), '%W, %D %M %Y %r');
+-----------------------------------------+
| DATE_FORMAT(NOW(), '%W, %D %M %Y %r') |
+-----------------------------------------+
| Thursday, 18th November 2004 12:07:55 PM |
+-----------------------------------------+
1 row in set (0.22 sec)

mysql>SELECT DATE_FORMAT(19980317, '%d/%m/%Y');
+----------------------------------+
| DATE_FORMAT(19980317, '%d/%m/%Y') |
+----------------------------------+
| 17/03/1998 |
+----------------------------------+
1 row in set (0.00 sec)
mysql>SELECT DATE_FORMAT("20011215101030", "%H%ihrs on
%a %d %M %y");
```

```
+-----------------------------------------------------+
| DATE_FORMAT("20011215101030", "%H%ihrs on %a %d %M %y") |
+-----------------------------------------------------+
| 1010 hrs on Sat 15 December 01 |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

Formatting dates with the date() function

mysql>**SELECT TIME_FORMAT(19690609140256, '%h:%i %p');**

```
+---------------------------------------+
| TIME_FORMAT(19690609140256, '%h:%i %p') |
+---------------------------------------+
| 02:02 PM |
+---------------------------------------+
1 row in set (0.00 sec)
```

MySQL Date/Time Formatting Codes

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

```
<html>
<head></head>
<body>
<?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 name, DATE_FORMAT(dob, '%d %b %Y') ↲
AS dob FROM birthdays";
$result = mysql_query($query) ↲
or die ('Error in query: $query. ' . mysql_error());
// check if records were returned
if (mysql_num_rows($result) > 0)
{
// print HTML table
echo '<table border=1 cellpadding=10>';
// iterate over record set
// print each field
while($row = mysql_fetch_object($result))
{
echo '<tr>';
echo "<td>$row->name</td><td>$row->dob</td>";
echo '</tr>';
```

```
}
echo '</table>';
}
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);
?>
</body>
</html>
```

References

1. How to Do Everything with PHP & MySQL – Vikram Vaswani
2. www.tutorialpoint.com

- - - ☙ ☙ ☙ ☙ ☙ - - -