AUTHORS
MR. VIRAL B. POLISHWALA
MS. KEJAL C. VADZA

For the Students of TYBCA

PHP- MySQL

First Edition

:: Authors ::

Mr. Viral B. Polishwala
Asst. prof.
Sutex Bank College of
Computer Application & Science
Amroli, Surat

Ms. Kejal C. Vadza
Asst. prof.
Sutex Bank College of
Computer Application & Science
Amroli, Surat

About the Authors

Asst. Prof. Viral B. Polishwala has completed his M.C.A. from Department of South Gujarat University, VNSGU, Surat in 2004. Currently he is working as Asst. Professor in Sutex Bank College of Computer Application and Science, Amroli, Surat. He has total 1.3 years of experience in teaching field. He has covered many subjects of B.C.A. stream starting from Introduction to Computer, Data Structure, Web Designing, Information System, PHP-MySQL. He is actively involved in guiding TYBCA students for their final year project work. He has also worked as System Admin cum Programmer at SVNIT(Deemed University), Surat for 4.5 years. As well as he has worked as software engineer for 1 year at CoVisible Solutions(India) Pvt. Ltd.

Asst. Prof. Kejal C. Vadza has completed her M.C.A. from Department of Computer Science, VNSGU, Surat and M.phil. from The Global Open University, Nagaland. Currently she is working as Asst. Professor in Sutex Bank College of Computer Application and Science, Amroli, Surat. She has total 3.5 years of experience in teaching field. She has covered many subjects of B.C.A. stream starting from Introduction to Computer, Computer Programming and Programming Methodology, Organization Behavior, Information System, Object Oriented Programming, JAVA, Information System Analysis & Design and PHP-MySQL. She is actively involved in guiding TYBCA students for their final year project work. She has also served as an ISO Auditor & Lecturer at Laxmi Vidhyapith Institute of Computer Application and science, Sarigam, Vapi. She has leaded her team of Kho-Kho at state level competition.

Dedicated to...

I, Viral B. Polishwala, would like to dedicate this book to my loving wife Megha ,Vrishti(My Daughter), My Parents Shri. Bhagwandasbhai and Smt. Ramilaben and Polishwala family.

I, Kejal C. Vadza, would like to dedicate this book to my loving husband Chintan, Shri. Jagdishchandra Patel & Family and Rameshbhai Vadza & Family.

Acknowledgement

Authors of this book sincerely thank Dr. Ravi Gulati Prof. of Department of Computer Science, VNSGU, Surat, initially gave his expert advice in writing book and corrected us in all phases of book writing. This book is also result of constant motivation from our family, colleagues, friends and well wishers. We received great inspiration and constant encouragement from all of them. Special mention is made of Shri. J.B. Shah who gave us valuable help.

We sincerely thank mr. Apurva Shah, publisher of B.S.Shah Publication for Publishing book and spread our knowledge of "PHP MySQL".

And how should we forget to acknowledge efforts made by us and to say thanks to each other, We would like to appreciate each other.

This book is merely dedicated to B.C.A. students as it is written as per their academic syllabus.

Mr. Viral B. Polishwala
Asst. Prof.
Sutex Bank College of
Computer Application & Science,
Amroli, Surat.

Ms. Kejal C. Vadza
Asst. Prof.,
Sutex Bank College of
Computer Application & Science,
Amroli, Surat.

:: Preface ::

The web has democratized design—both the process of design and the attendant responsibilities. Knowingly or not, we have been engaging in design throughout our lives, but without such broad implications. When we choose whether to wear brown shoes or black, the effect of our choice is limited to the people we encounter throughout the day. When we design a Web site, we design for everyone. For this we are using certain languages like PHP, ASP, ASP.Net etc.

This book provides a broad and through introduction to PHP and MySQL. PHP is the engine behind millions of dynamic web applications. Its broad feature set, approachable syntax, flexibility with MySQL database and support for different operating systems and web servers have made it an ideal language for both rapid web development and the methodical construction of complex systems. PHP is the widely-used, free, and efficient alternative to competitors such as Microsoft's ASP.

It is intended primarily as a core text for use in course of BCA in Veer Narmad South Gujarat University and other tertiary-level educational institutions as well as beginners. It presents a systematic explanation of the designing Websites using PHP, which will serve as a foundation for students or anyone setting out on a designing a website. This book is not a collection of hints and tips, nor the presentation of a method to follow step-by-step, nor a tutorial introduction to any Web site creation program.

About the Book

This book is consisting of 8 chapters which are according to the syllabus of TYBCA affected from Jun 2011.

1st Chapter: Consists the basics of PHP and methods of installation of required softwares like Apache (web server), PHP (Scripting Language) and MySQL(Databse). It also consists of PHP features and importance of the configuration of PHP.ini file.

2nd Chapter: Consists the basic methodology for handling the PHP on server with html, various data types of PHP, available operators in PHP and various commenting styles in PHP.

3rd Chapter: Consists of control structures available in PHP. It is consisting of conditional statements and control loops as well as statements like exit, die, return and various array types we are able to use in PHP.

4th Chapter: Consists the revision concept of passing the data throughout the website by using form interface and various passing techniques (From Web Design Subject of 4th Semester of B.C.A.). It also consists of various validation techniques for handling the data on web.

5th Chapter: Consists the set of built in functions to manipulate the PHP data likewise: strings, array, mathematical, date & time, file system handling and miscellaneous functions as well as the creation and using of user defined functions in PHP.

6th Chapter: Consists the mostly used concept on web that is session and cookies for handling the data throughout the particular time moment the user is using the website.

7th Chapter: Consists the way of uploading various type of files to the web server by using PHP file uploading functions.

8th Chapter: Consists the MySQL explanation in required depth for handling the database for your website. It is having set of functions for manipulating the database and tables as well as necessary functions of PHP for establishing the connection to the database server and handling the data on server.

In the second part of the book you may find the available html tags, events and entities which may be useful for designing the website.

Veer Narmad South Gujarat University - Surat.
Bachelor of Computer Application(B.C.A.) 3rd Year
Syllabus
T.Y.B.C.A. – Semester 5
Effective From: Jun 2011

Paper No.: 501
Paper Title: PHP-MySQL

1. Introduction to PHP
   - 1.1 Installation of PHP and MySQL
   - 1.2 PHP configuration in IIS and Apache web server and Features of PHP
2. Writing PHP
   - 2.1 How PHP code is Parsed?
   - 2.2 Embedding PHP & HTML
   - 2.3 Executing PHP & Viewing in browsers
   - 2.4 Data types
   - 2.5 Operators
   - 2.6 PHP variables: Static and global Variable
   - 2.7 Comments in PHP
3. Control Structure
   - 3.1 Conditional Statement
     - 3.1.1 If…Else
     - 3.1.2 Switch
     - 3.1.3 ? Operator
   - 3.2 Loops
     - 3.2.1 While
     - 3.2.2 Break Statement
     - 3.2.3 Continue
     - 3.2.4 Do…While
     - 3.2.5 For
     - 3.2.6 For each
   - 3.3 Exit, Die and Return statement
   - 3.4 Arrays in PHP
4. Working with data
   - 4.1 Form elements
   - 4.2 Validating User Inputs
   - 4.3 Passing Variables between Pages
     - 4.3.1 Using GET method
     - 4.3.2 Using POST method
     - 4.3.3 Using REQUEST method
5. Functions
   - 5.1 Built-in Function
     - 5.1.1 String Function: chr, ord, strtolower, strtoupper, strlen, ltrim, rtrim, substr, strcmp, strcasecmp, strops, strrpos, strstr, stristr, str_replace,strrev, echo, print.
     - 5.1.2 Math. Function: abs, ceil, floor, round, fmod, min, max, pow, sqrt, rand.
     - 5.1.3 Date Function: date, getdate,checkdate, time, mktime.
     - 5.1.4 Array Function: count, list, in_array, current, next, previous, end, each, sort, rsort, assort, array_merge,array_reverse.

- 5.1.5 File Handling Function: fopen, fread, fwrite, fclose, file_exists, is_readable, is_writeable, fgets, file, file_get_contents, file_put_contents, ftell, fseek, rewind, copy, unlink, rename.
  - 5.1.6 Miscellaneous Function: define, constant, include, require, header, die.
- 5.2 User Defined Function

6. Handling Session & Cookie
   - 6.1 Concept of Session
   - 6.2 Starting Session
   - 6.3 Modifying Session Variables
   - 6.4 unregistering & Deleting Session Variables
   - 6.5 Concept of Cookies
   - 6.6 Handling Cookies
7. How to Upload Files
8. Introduction to MySQL
   - 8.1 Types of Table in MySQL
   - 8.2 Query in MySQL
     - 8.2.1 Select, insert, update, delete
   - 8.3 Truncate
   - 8.4 Alias
   - 8.5 Order By
   - 8.6 Database Connectivity of PHP and MySQL

Contents

(T.Y.B.C.A. – Semester – 5 – Paper No. - 501)

Effective from June - 2011

***

Part - 1

PHP – MySQL

Chapter 1 – 8

Chapter – 1

Introduction to PHP

In this Chapter

What is PHP?

Common use of PHP

Why choose PHP?

Characteristics / Features of PHP

What's new in PHP5?

Installing Apache, PHP & MySQL

What Is PHP?

PHP is a language that has outgrown its name. It was originally conceived as a set of macros to help coders maintain personal home pages, and its name grew from its purpose. Since then, PHP's capabilities have been extended, taking it beyond a set of utilities to a full-featured programming language, capable of managing huge database-driven online environments.

As PHP's capabilities have grown, so too has its popularity. According to NetCraft (http://www.netcraft.com), PHP was running on more than 1 million hosts in November 1999. As of September 2001, that figure had already risen to over 6 million hosts, and by October 2003 PHP was reportedly installed on almost 14 million hosts. According to SecuritySpace.com, PHP is the most popular Apache module available, beating mod_ssl, Perl, and FrontPage.

PHP is also installed as a command-line application, making it an excellent tool for scripting on a server. Many system administrators now use PHP for the sort of automation that has been traditionally handled by Perl or shell scripting.

The PHP Hypertext Preprocessor (PHP) is a programming language that allows web developers to create dynamic content that interacts with databases.

PHP is basically used for developing web based software applications.

PHP started out as a small open source project that evolved as more and more people found out how useful it was. Rasmus Lerdorf unleashed the first version of PHP way back in 1994.

- PHP is a recursive acronym for "PHP: Hypertext Preprocessor".
- PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.
- It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.
- PHP is pleasingly zippy in its execution, especially when compiled as an Apache module on the Unix side. The MySQL server, once started, executes even very complex queries with huge result sets in record-setting time.
- PHP supports a large number of major protocols such as POP3, IMAP, and LDAP. PHP4 added support for Java and distributed object architectures (COM and CORBA), making n-tier development a possibility for the first time.
- PHP is forgiving: PHP language tries to be as forgiving as possible.
- PHP Syntax is C-Like.

Common uses of PHP:

- PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them.
- PHP can handle forms, i.e. gather data from files, save data to a file, thru email you can send data, return data to the user.
- You add, delete and modify elements within your database through PHP.
- Access cookies variables and set cookies.
- Using PHP, you can restrict users to access some pages of your website.
- It can encrypt data.

Why Choose PHP?

There are some compelling reasons to work with PHP. For many projects, you will find that the production process is significantly faster than you might expect if you are used to working with other scripting languages. At Corrosive we work with both PHP and Java. We choose PHP when we want to see results quickly without sacrificing stability. As an open-source product, PHP is well supported by a talented production team and a committed user community. Furthermore, PHP can be run on all the major operating systems and with most servers.

Characteristics / Features of PHP

- ***Speed of Development***

Because PHP allows you to separate HTML code from scripted elements, you will notice a significant decrease in development time on many projects. In many instances, you will be able to separate the coding stage of a project from the design and build stages. Not only can this make life easier for you as a programmer, but it also can remove obstacles that stand in the way of effective and flexible design.

- ***PHP Is Open Source***

To many people, open source simply means free, which is, of course, a benefit in itself. Well-maintained open-source projects offer users additional benefits, though. You benefit from an accessible and committed community that offers a wealth of experience in the subject. Chances are that any problem you encounter in your coding can be answered swiftly and easily with a little research. If that fails, a question sent to a mailing list can yield an intelligent, authoritative response.

You also can be sure that bugs will be addressed as they are found, and that new features will be made available as the need is defined. You will not have to wait for the next commercial release before taking advantage of improvements.

There is no vested interest in a particular server product or operating system. You are free to make choices that suit your needs or those of your clients, secure that your code will run whatever you decide.

- ***Performance***

Because of the powerful Zend engine, PHP shows solid performance compared with other server scripting languages, such as ASP, Perl, and Java Servlets, in benchmark tests. To further improve performance, you can acquire a caching tool (Zend Accelerator) from http://www.zend.com/; it stores compiled code in memory, eliminating the overhead of parsing and interpreting source files for every request.

- ***Portability***

PHP is designed to run on many operating systems and to cooperate with many servers and databases. You can build for a Unix environment and shift your work to NT without a problem. You can test a project with Personal Web Server and install it on a Unix system running on PHP as an Apache module.

What's New in PHP 5

PHP 5 introduces numerous new features that will make the programmer's life more interesting. Let's take a quick look at some of them.

- PHP has new integrated for support for XML. The various functions and classes provided to handle XML in different ways all now use the same underlying library (libxml2). This should make XML features more stable and interoperable.
- The SQLite SQL library is now bundled with PHP, together with all the functions you need to work with it.
- PHP now supports private and protected methods and properties in classes.
- PHP supports class constants.
- Objects passed to functions and methods are now passed by reference. That is, a reference to an object is passed around your script rather than copies of objects. This significantly reduces the likelihood of bugs in object-oriented code.
- PHP supports static methods and properties, making more advanced object-oriented designs possible.

- Methods can now be declared to require particular object types.
- The comparison operator (===) now checks that two references point to the same object. Previously, it was hard to test objects in this way.
- PHP now supports abstract classes and interfaces.

Why Install PHP Locally?

Installing PHP on your development PC allows you to safely create and test a web application without affecting the data or systems on your live website. This article describes PHP installation as a module within the Windows version of Apache 2.2. Mac and Linux users will probably have it installed already.

All-in-One packages

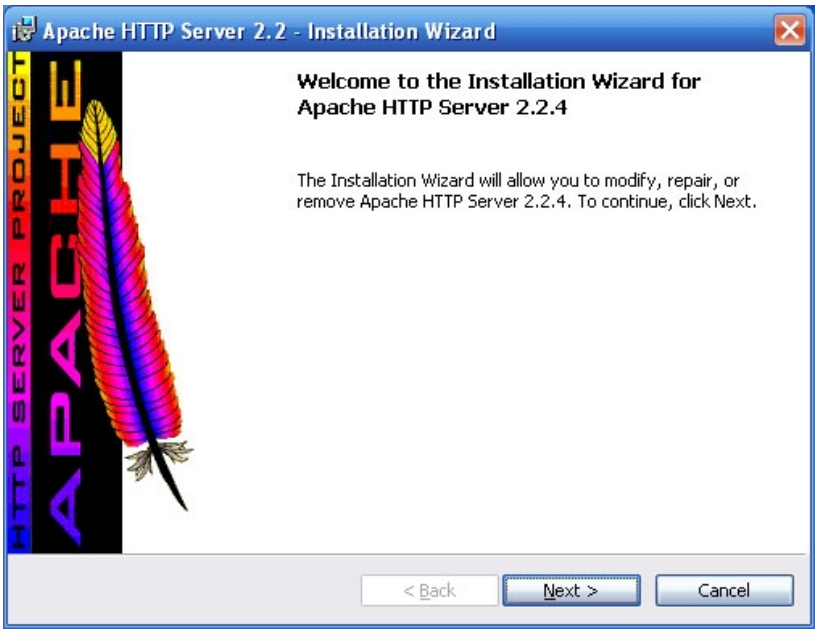
There are some excellent all-in-one Windows distributions that contain Apache, PHP, MySQL and other applications in a single installation file, e.g. XAMPP (including a Mac version), WampServer and WebDeveloper. There is nothing wrong with using these packages, although manually installing Apache and PHP will help you learn more about the system and its configuration options.

Before starting the installation of PHP first we see how to install Apache or IIS which will be treated as web server for PHP(server side scripting language).

Installing Apache:

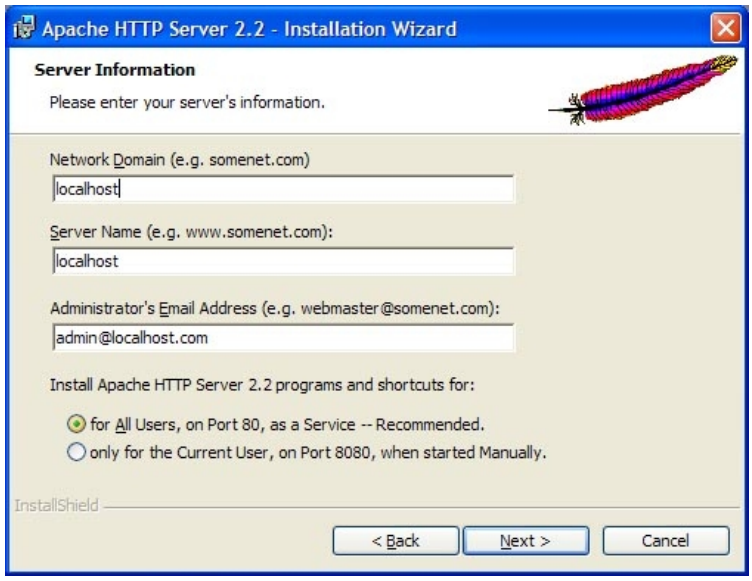
We will be installing **Apache version 2.2.4**. Follow the steps carefully.

1. Go to www.apache.org and download "Win32 Binary (MSI Installer): apache_2.2.4-win32-x86-no_ssl.msi" to your desktop.
   Note: Make sure that you download Apache version 2.2.4 (Win32 Binary MSI Installer)! The rest of the tutorial is written using this version.
2. Double click "apache_2.2.4-win32-x86-no_ssl.msi", and if prompted, click "run".
3. An installation wizard will appear:



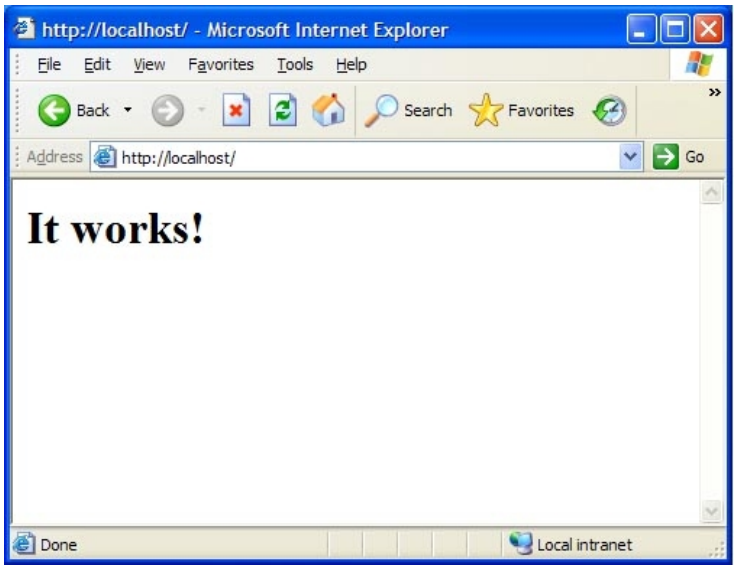
Click "Next".

4. The next page contains the terms of agreement. Select "I accept", and click "Next".
5. Read about the Apache Server, and click "Next"
6. The next screen will ask you for specific server information. Enter the values seen below:



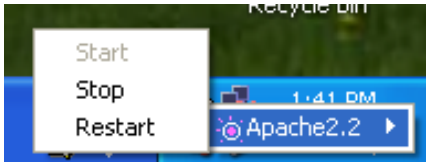
Click "Next".

7. On the next screen, select "Typical Installation" and click "Next".
8. Click "Next".
9. Click "Install".
10. Open up Internet Explorer and type in "http://localhost". If you see a page that says "It works!" then the Apache server has been installed successfully.



Now, if you look in your task bar, you should see an icon that looks like this:

That's a status indicator that says "Apache is running!" You can click it to stop Apache:



...and you should see the status change.

A few notes on your Apache Server Configuration:

- Apache is installed by default in your "C:\Program Files\Apache Software Foundation\Apache2.2" directory.
- Inside that directory there is a folder called "htdocs" (the equivilant of your \www\ or \public_html\ directory). You can develop your applications inside this folder and access them by going to http://localhost/your_file_name.php
- The Apache Configuration settings are defined in a file named "httpd.conf" located in the "conf" directory. **Do not attempt to change these settings unless you know what you're doing. An error in this file will result in the Apache Server not functioning correctly!**

To install IIS using the Configure Your Server Wizard

1.	From the **Start** menu, click **Manage Your Server**.
2.	Under **Managing Your Server Roles**, click **Add or remove a role**.
3.	Read the preliminary steps in the **Configure Your Server Wizard** and click **Next**.
4.	Under **Server Role**, click **Application server (IIS, ASP.NET)** and then click **Next**. By default, the wizard installs and enables IIS, COM+, and DTC.
5.	If you want to serve either of the optional technologies (FrontPage Server Extensions or ASP.NET), on the **Application Server Options** page, select the appropriate check boxes, and then click **Next**.
6.	Read the summary and click **Next**.
7.	Complete the wizard, and then click **Finish**.

Note: The **Configure Your Server Wizard** enables ASP.NET by default, unlike the **Add/Remove Windows components** install method below.

To install IIS, add components, or remove components using Control Panel

1.	From the **Start** menu, click **Control Panel**.
2.	Double-click **Add or Remove Programs**.
3.	Click **Add/Remove Windows Components**.
4.	In the **Components** list box, click **Application Server**.
5.	Click **Details**.
6.	Click **Internet Information Services Manager**.
7.	Click **Details** to view the list of IIS optional components. For a detailed description of IIS optional components, see "Optional Components" in this topic.
8.	Select all optional components you wish to install.

Now, we'll go for the PHP and MYSQL installation. We are able to install PHP in two different ways: **1) By PHP installer** and **2) Manual installation of PHP**

The PHP Installer

Although an installer is available from php.net, I would recommend the manual installation if you already have a web server configured and running.

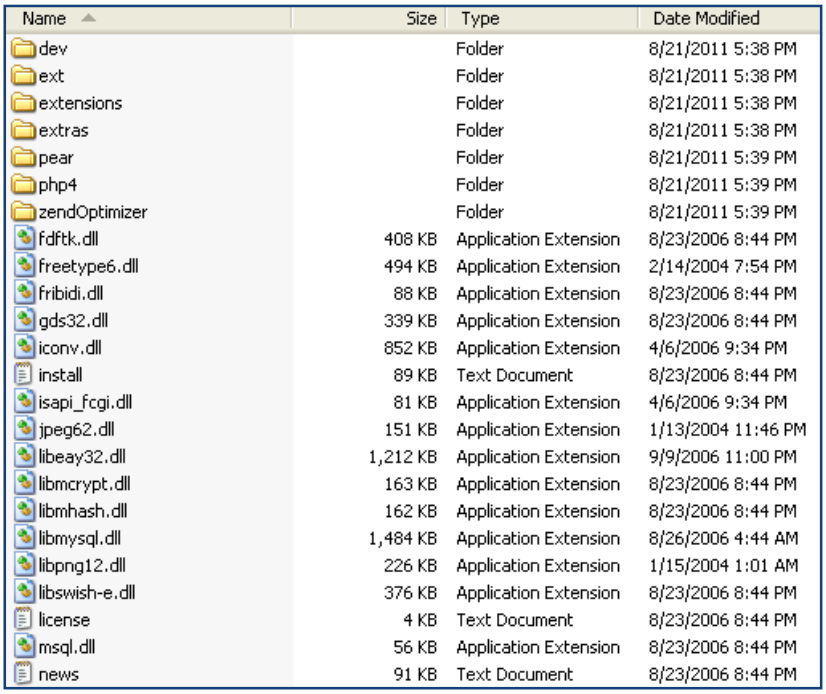
Manual Installation

Manual installation offers several benefits:

- ❖ Backing up, reinstalling, or moving the web server can be achieved in seconds
- ❖ You have more control over PHP and Apache configuration.

Next we will be installing PHP version 5. Follow the steps carefully.

1. Go to www.php.net and download the "PHP 5.2.0 zip package" to your desktop. (Be patient while it downloads, the ZIP file is over 9MB!)
   Note: Make sure that you download the PHP 5.2.0 zip package!
2. Create a new folder called "php" in your C Drive. Copy the "php-5.2.0-Win32.zip" file to there ("C:\php") and extract it using WinZIP or a similiar program like WinRar.
3. Your "C:\php" directory should now look like:



Name	Size	Type	Date Modified
dev		Folder	8/21/2011 5:38 PM
ext		Folder	8/21/2011 5:38 PM
extensions		Folder	8/21/2011 5:38 PM
extras		Folder	8/21/2011 5:38 PM
pear		Folder	8/21/2011 5:39 PM
php4		Folder	8/21/2011 5:39 PM
zendOptimizer		Folder	8/21/2011 5:39 PM
fdftk.dll	408 KB	Application Extension	8/23/2006 8:44 PM
freetype6.dll	494 KB	Application Extension	2/14/2004 7:54 PM
fribidi.dll	88 KB	Application Extension	8/23/2006 8:44 PM
gds32.dll	339 KB	Application Extension	8/23/2006 8:44 PM
iconv.dll	852 KB	Application Extension	4/6/2006 9:34 PM
install	89 KB	Text Document	8/23/2006 8:44 PM
isapi_fcgi.dll	81 KB	Application Extension	4/6/2006 9:34 PM
jpeg62.dll	151 KB	Application Extension	1/13/2004 11:46 PM
libeay32.dll	1,212 KB	Application Extension	9/9/2006 11:00 PM
libmcrypt.dll	163 KB	Application Extension	8/23/2006 8:44 PM
libmhash.dll	162 KB	Application Extension	8/23/2006 8:44 PM
libmysql.dll	1,484 KB	Application Extension	8/26/2006 4:44 AM
libpng12.dll	226 KB	Application Extension	1/15/2004 1:01 AM
libswish-e.dll	376 KB	Application Extension	8/23/2006 8:44 PM
license	4 KB	Text Document	8/23/2006 8:44 PM
msql.dll	56 KB	Application Extension	8/23/2006 8:44 PM
news	91 KB	Text Document	8/23/2006 8:44 PM

4. Next rename the "php.ini-dist" file to "php.ini" and copy it from "C:/php/" to "C:/WINDOWS". This is your PHP configuration file. Explanation on this file will be later.
5. Now we are require to configure the PHP according to the web server we have installed.

a) Configuration on Apache Web Server

- Using Notepad open *httpd.conf* (should be start-menu shortcut *"Apache HTTP Server 2.2 > Configure Apache Server > Edit the Apache httpd.conf Configuration File"*). Either at the very beginning or end of the file add the following lines:

(**NOTE:** be sure to change BOTH C:/php parts to the directory you installed your php to)

```
LoadModule php5_module "C:/php/php5apache2_2.dll"
AddType application/x-httpd-php .php
PHPIniDir "C:/php"
```

Note: If you installed the older Apache 2.0, instead of the above lines, you will need to use the lines listed on the bottom step of the Apache 2.0 tutorial.

- **[Optional]**To get Apache to automatically look for an index.php, search httpd.conf for **DirectoryIndex** (about line 212) and add the files you want apache to look for when a directory is loaded (if it doesn't find any of these files, it displays folder contents). Mine looks like:

```
<IfModule dir_module>
    DirectoryIndex index.php index.html default.html
</IfModule>
```

- Restart your Apache Server for the changes to take effect: Start > All Programs > Apache HTTP Server 4.2.4 > Control Apache Server > Restart

b) Configuration on IIS Web Server

- Copy some .dll files from your PHP directory to your systems directory (usually C:\Winnt\System32). You need php5ts.dll for every case. You will also probably need to copy the file corresponding to your Web server module - C:\PHP\Sapi\php5isapi.dll. It's possible you will also need others from the dlls subfolder - but start with the two mentioned above and add more if you need them.
- Copy either php.ini-dist or php.ini-recommended (preferably the latter) to your Windows directory (C:\Winnt or C:\Winnt40), and rename it php.ini. Open this file in a text editor (for example, Notepad). Edit this file to get configuration directives; We highly recommend new users set error reporting to E_ALL on their development machines at this point. For now, the most important thing is the doc_root directive under the Paths and Directories section.make sure this matches your IIS Inetpub folder (or wherever you plan to serve out of).
- Stop and restart the WWW service. Go to the **Start menu -> Settings -> Control Panel -> Services.** Scroll down the list to IIS Admin Service. Select it and click Stop. After it stops, select World Wide Web Publishing Service and click Start. Stopping and restarting the service from within Internet Service Manager will not suffice. Since this is Windows, you may also wish to reboot.

6. Now you have configured the PHP with the web server. In this step you have to just testing PHP is working properly. For that if you have installed apache web server then; In your "htdocs" directory (somewhere in **Program Files/ apache foundation...**), or if you have installed IIS web server then in your **www** directory (residing under **inetpub** directory) , create a file called "info.php". Open it in notepad and add this line of code to it:

<?php phpinfo(); ?>

7. Open up Internet Explorer or any other web browser you may have and type in: http://localhost/info.php. If your browser takes you to a page that looks like this, then PHP has been installed successfully!

System	Windows NT GRANITESTATE 5.1 build 2600
Build Date	Nov 2 2006 11:50:55
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--with-gd=shared"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\WINDOWS\php.ini
PHP API	20041225
PHP Extension	20060613
Zend Extension	220060519
Debug Build	no
Thread Safety	enabled
Zend Memory Manager	enabled
IPv6 Support	enabled
Registered PHP Streams	php, file, data, http, ftp, compress.zlib
Registered Stream Socket Transports	tcp, udp
Registered Stream Filters	convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, zlib.*

"Hello World" Script in PHP:

To get a feel for PHP, first start with simple PHP scripts. Since "Hello, World!" is an essential example, first we will create a friendly little "Hello, World!" script.

PHP is embedded in HTML. That means that in amongst your normal HTML (or XHTML if you're cutting-edge) you'll have PHP statements like this:

```
<html>
<head>
<title>Hello World</title>
<body>
   <?php echo "Hello, World!";?>
</body>
</html>
```

It will produce following result:

```
Hello, World!
```

If you examine the HTML output of the above example, you'll notice that the PHP code is not present in the file sent from the server to your Web browser. All of the PHP present in the Web page is processed and stripped from the page; the only thing returned to the client from the Web server is pure HTML output.

PHP Installation on Linux/Unix

If you plan to install PHP on Linux or any other variant of Unix, then here is the list of prerequisites:

- The PHP source distribution http://www.php.net/downloads.php
- The latest Apache source distribution http://httpd.apache.org/download.cgi
- A working PHP-supported database, if you plan to use one (For example MySQL, Oracle etc.)
- Any other supported software to which PHP must connect (mail server, BCMath package,JDK, and so forth)
- An ANSI C compiler
- Gnu make utility - you can freely download it at http://www.gnu.org/software/make

Now here are the steps to install Apache and PHP5 on your Linux or Unix machine. If your PHP or Apache versions are different then please take care accordingly.

Installation Steps

- If you haven't already done so, unzip and untar your Apache source distribution. Unless you have a reason to do otherwise, /usr/local is the standard place.

```
gunzip -c apache_1.3.x.tar.gz
tar -xvf apache_1.3.x.tar
```

- Build the apache Server as follows

```
cd apache_1.3.x
./configure --prefix=/usr/local/apache --enable-so
make
make install
```

- Unzip and untar your PHP source distribution. Unless you have a reason to do otherwise, /usr/local is the standard place.

```
gunzip -c php-5.x.tar.gz
tar -xvf php-5.x.tar
cd php-5.x
```

- Configure and Build your PHP, assuming you are using MySQL database.

```
./configure --with-apxs=/usr/sbin/apxs \
        --with-mysql=/usr/bin/mysql
make
make install
```

- Install the php.ini file. Edit this file to get configuration directives:

```
cd ../../php-5.x
cp php.ini-dist /usr/local/lib/php.ini
```

- Tell your Apache server where you want to serve files from, and what extension(s) you want to identify PHP files A .php is the standard, but you can use .html, .phtml, or whatever you want.
  - Go to your HTTP configuration files (/usr/local/apache/conf or whatever your path is)
  - Open httpd.conf with a text editor.
  - Search for the word DocumentRoot (which should appear twice), and change both paths to the directory you want to serve files out of (in our case, /home/httpd). We recommend a home directory rather than the default /usr/local/apache/htdocs because it is more secure, but it doesn.t have to be in a home directory. You will keep all your PHP files in this directory.
- Add at least one PHP extension directive, as shown in the first line of code that follows. In the second line, we.ve also added a second handler to have all HTML files parsed as PHP

```
AddType application/x-httpd-php .php
AddType application/x-httpd-php .html
```

- Restart your server. Every time you change your HTTP configuration or php.ini files, you must stop and start your server again.

```
cd ../bin
./apachectl start
```

- Set the document root directory permissions to world-executable. The actual PHP files in the directory need only be world-readable (644). If necessary, replace /home/httpd with your document root below:

```
chmod 755 /home/httpd/html/php
```

- Open a text editor. Type: <?php phpinfo(); ?>. Save this file in your Web server's document root as info.php.
- Start any Web browser and browse the file.you must always use an HTTP request (http://www.testdomain.com/info.php or http://localhost/info.php or http://127.0.0.1/info.php) rather than a filename (/home/httpd/info.php) for the file to be parsed correctly

You should see a long table of information about your new PHP installation message Congratulations!

PHP.INI file Configuration

The PHP configuration file, php.ini, is the final and most immediate way to affect PHP's functionality. The php.ini file is read each time PHP is initialized.in other words, whenever httpd is restarted for the module version or with each script execution for the CGI version. If your change isn.t showing up, remember to stop and restart httpd. If it still isn.t showing up, use phpinfo() to check the path to php.ini.

The configuration file is well commented and thorough. Keys are case sensitive, keyword values are not; whitespace, and lines beginning with semicolons are ignored. Booleans can be represented by 1/0, Yes/No, On/Off, or True/False. The default values in php.ini-dist will result in a reasonable PHP installation that can be tweaked later.

Here we are explaining the important settings in php.ini which you may need for your PHP Parser.

short_open_tag = Off

Short open tags look like this: <? ?>. This option must be set to Off if you want to use XML functions.

safe_mode = Off

If this is set to On, you probably compiled PHP with the --enable-safe-mode flag. Safe mode is most relevant to CGI use. See the explanation in the section "CGI compile-time options". earlier in this chapter.

safe_mode_exec_dir = [DIR]

This option is relevant only if safe mode is on; it can also be set with the --with-exec-dir flag during the Unix build process. PHP in safe mode only executes external binaries out of this directory. The default is /usr/local/bin. This has nothing to do with serving up a normal PHP/HTML Web page.

safe_mode_allowed_env_vars = [PHP_]

This option sets which environment variables users can change in safe mode. The default is only those variables prepended with "PHP_". If this directive is empty, most variables are alterable.

safe_mode_protected_env_vars = [LD_LIBRARY_PATH]

This option sets which environment variables users can't change in safe mode, even if safe_mode_allowed_env_vars is set permissively

disable_functions = [function1, function2...]

A welcome addition to PHP4 configuration and one perpetuated in PHP5 is the ability to disable selected functions for security reasons. Previously, this necessitated hand-editing the C code from which PHP was made. Filesystem, system, and network functions should probably be the first to go because allowing the capability to write files and alter the system over HTTP is never such a safe idea.

max_execution_time = 30

The function set_time_limit() won.t work in safe mode, so this is the main way to make a script time out in safe mode. In Windows, you have to abort based on maximum memory consumed rather than time. You can also use the Apache timeout setting to timeout if you use Apache, but that will apply to non-PHP files on the site too.

error_reporting = E_ALL & ~E_NOTICE

The default value is E_ALL & ~E_NOTICE, all errors except notices. Development servers should be set to at least the default; only production servers should even consider a lesser value

error_prepend_string = [""]

With its bookend, error_append_string, this setting allows you to make error messages a different color than other text, or what have you.

warn_plus_overloading = Off

This setting issues a warning if the + operator is used with strings, as in a form value.

variables_order = EGPCS

This configuration setting supersedes gpc_order. Both are now deprecated along with register_globals. It sets the order of the different variables: Environment, GET, POST, COOKIE, and SERVER (aka Built-in).You can change this order around. Variables will be overwritten successively in left-to-right order, with the rightmost one winning the hand every time. This means if you left the default setting and happened to use the same name for an environment variable, a POST variable, and a COOKIE variable, the COOKIE variable would own that name at the end of the process. In real life, this doesn't happen much.

register_globals = Off

This setting allows you to decide whether you wish to register EGPCS variables as global. This is now deprecated, and as of PHP4.2, this flag is set to Off by default. Use superglobal arrays instead. All the major code listings in this book use superglobal arrays.

gpc_order = GPC

This setting has been GPC Deprecated.

magic_quotes_gpc = On

This setting escapes quotes in incoming GET/POST/COOKIE data. If you use a lot of forms which possibly submit to themselves or other forms and display form values, you may need to set this directive to On or prepare to use addslashes() on string-type data.

magic_quotes_runtime = Off

This setting escapes quotes in incoming database and text strings. Remember that SQL adds slashes to single quotes and apostrophes when storing strings and does not strip them off when returning them. If this setting is Off, you will need to use stripslashes() when outputting any type of string data from a SQL database. If magic_quotes_sybase is set to On, this must be Off.

magic_quotes_sybase = Off

This setting escapes single quotes in incoming database and text strings with Sybase-style single quotes rather than backslashes. If magic_quotes_runtime is set to On, this must be Off.

auto-prepend-file = [path/to/file]

If a path is specified here, PHP must automatically include() it at the beginning of every PHP file. Include path restrictions do apply.

auto-append-file = [path/to/file]

If a path is specified here, PHP must automatically include() it at the end of every PHP file.unless you escape by using the exit() function. Include path restrictions do apply.

include_path = [DIR]

If you set this value, you will only be allowed to include or require files from these directories. The include directory is generally under your document root; this is mandatory if you.re running in safe mode. Set this to . in order to include files from the same directory your script is in. Multiple directories are separated by colons: .:/usr/local/apache/htdocs:/usr/local/lib.

doc_root = [DIR]

If you are using Apache, you.ve already set a document root for this server or virtual host in httpd.conf. Set this value here if you.re using safe mode or if you want to enable PHP only on a portion of your site (for example, only in one subdirectory of your Web root).

file_uploads = [on/off]

Turn on this flag if you will upload files using PHP script.

upload_tmp_dir = [DIR]

Do not uncomment this line unless you understand the implications of HTTP uploads!

session.save-handler = files

Except in rare circumstances, you will not want to change this setting. So don't touch it.

ignore_user_abort = [On/Off]

This setting controls what happens if a site visitor clicks the browser.s Stop button. The default is On, which means that the script continues to run to completion or timeout. If the setting is changed to Off, the script will abort. This setting only works in module mode, not CGI.

mysql.default_host = hostname

The default server host to use when connecting to the database server if no other host is specified.

mysql.default_user = username

The default user name to use when connecting to the database server if no other name is specified.

mysql.default_password = password

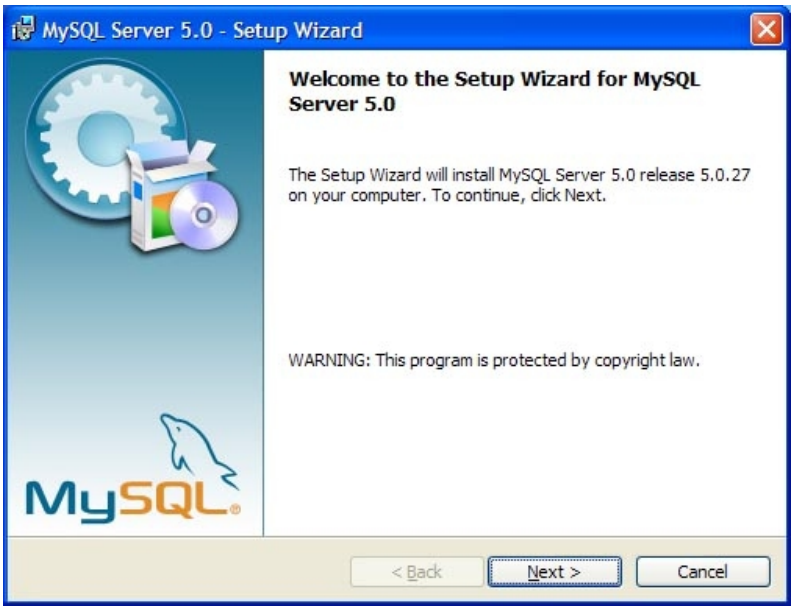
The default password to use when connecting to the database server if no other password is specified.

MySQL Installation

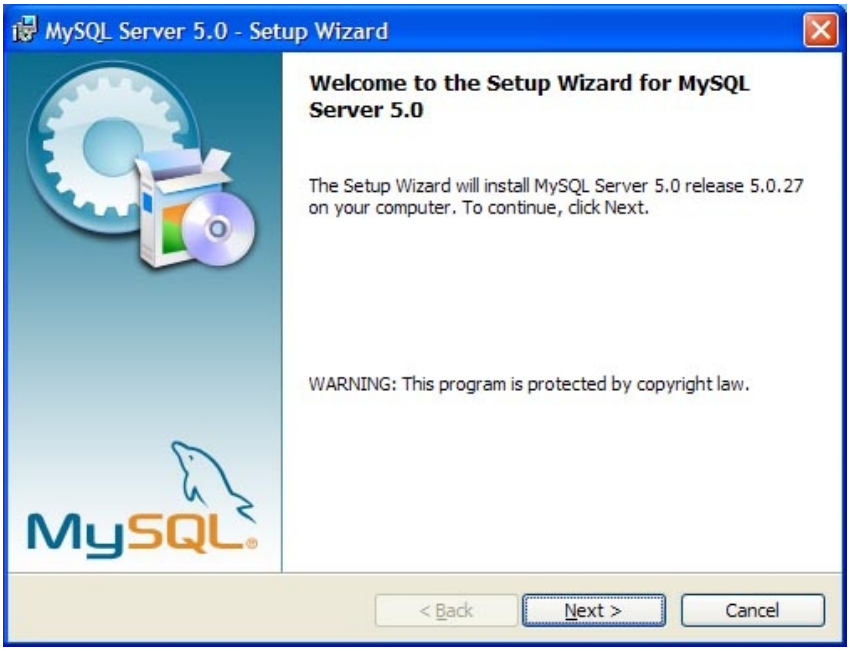
Following are the steps for installing the MySQL with the installer .exe file

1. Go to www.mysql.com and download the "Windows (x86) ZIP/Setup.EXE (version 5.0.27)" to your desktop. (To do this you'll need to register an account with MySQL.)

2. Once "mysql-5.0.27-win32.zip" has finished downloading, you can extract it using WinZIP or a similiar program.
3. Once extracted, double click on the "Setup.exe" file. An installation wizard will appear. Click "Next".

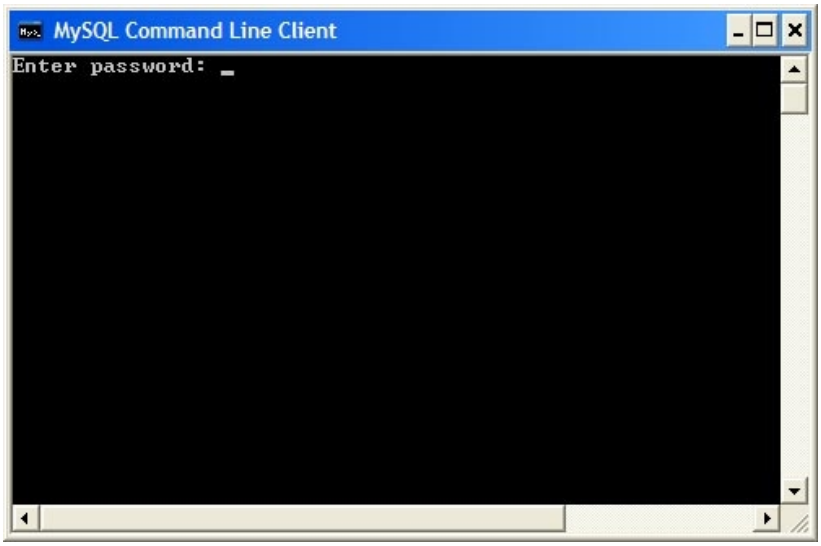


4. Select "Typical" Installation and click "Next".
5. Click "Install". (Be patient, this can take up to several minutes).
6. The next screen will ask you to "Sign Up". Select "Skip Sign-Up" for now.
7. The next screen will tell you that the installation wizard is complete. Make sure that the "Configure the MySQL Server now" field is checked before clicking "Finish".

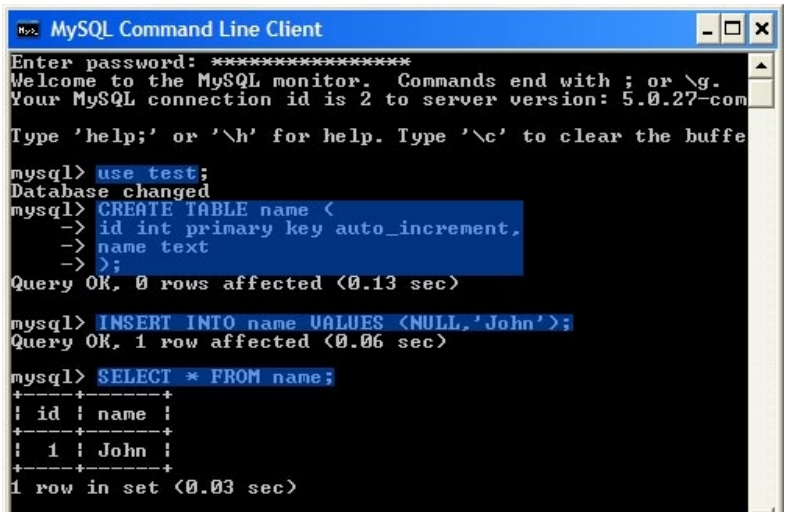


8. The MySQL Server Instance Configuration Wizard should appear. Click "Next".
9. Select "Detailed Configuration" and click "Next".

10. Select "Developer Machine" and click "Next".
11. Select "Multifunctional Database" and click "Next".
12. Click "Next".
13. Select "Decision Support (DSS)/OLAP" and click "Next".
14. Select "Multifunctional Database" and click "Next".
15. Make sure "Enable TCP/IP Networking" is checked, the Port Number is set to "3306", and "Enable Strict Mode" is checked. Click "Next".
16. Select "Standard Character Set" and click "Next".
17. Check "Install As Windows Service", set the Service Name to "MySQL", and check "Launch the MySQL Server automatically". Make sure that the "Include Bin Directory in Windows Path" is NOT checked. Click "Next".
18. On the next screen, check the box that says "Modify Security Settings". Enter a password for the default "root" account, and confirm the password in the box below. Do NOT check the boxes "Enable root access from remote machines" or "Create An Anonymous Account". Click "Next".
19. Click "Execute". (This may take a few minutes. Be patient).
20. Click "Finish".
21. To test if MySQL was installed correct, go to: Start > All Programs > MySQL > MySQL Server 5.0 > MySQL Command Line Client. The MySQL Command Line Client will appear:



22. It will ask you for a password. Enter the password you created in step 18. (If you enter an incorrect password MySQL will automatically close the command line)
23. Next, type in the commands shown below: (shown in blue)



If you don't get any errors, and it returns the information shown above, then MySQL has been successfully installed! Next we will need to configure PHP to work with MySQL.

Another way of installing MySQL will be exaplined in chapter 8 through the zipped folder.

Configuring PHP to work with MySQL:

Now that both PHP and MySQL are installed, we have to configure them to work together.

1. Open up your php.ini file (C:/WINDOWS/php.ini) and find the line:
   ;extension=php_mysql.dll
   To enable the MySQL extension, delete the semi-colon at the beginning of that line.
2. Next we must add the PHP directory to the Windows PATH. To do this, click: Start > My Computer > Properties > Advanced > Environment Variables. Under the second list (System Variables), there will be a variable called "Path". Select it and click "Edit". Add ";C:\php" to the very end of the string and click "OK".
3. Restart your computer for the changes to take effect.

Connection to the mysql from PHP will be explained in chapter 8

Exercise

1. What is PHP and state its features.
2. Write steps to manual installation PHP.
3. How to install MySQL in detail?
4. How to Configure IIS and Apache Web Server?
5. Explain php.ini configuration file.
6. How do I know if **magic quotes** is switched on?

Chapter – 2

Writing PHP

In this Chapter

How PHP code is parsed?

Embedding PHP and MySQL

Data types in PHP

PHP Variables

Global and Static Variables

Operators in PHP

Comments in PHP

This chapter will give you an idea of very basic syntax of PHP and very important to make your PHP foundation strong.

Writing PHP

Before we talk about PHP's syntax, let us first define what syntax is referring to.

Syntax - The rules that must be followed to write properly structured code.

PHP's syntax and semantics are similar to most other programming languages (C, Java, Perl) with the addition that all PHP code is contained with a tag, of sorts. All PHP code must be contained within the following

How PHP code is parsed?

The PHP parsing engine needs a way to differentiate PHP code from other elements in the page. The mechanism for doing so is known as 'escaping to PHP.' When PHP parses a file, it looks for opening and closing tags, which tell PHP to start and stop interpreting the code between them. Parsing in this manner allows PHP to be embedded in all sorts of different documents, as everything outside of a pair of opening and closing tags is ignored by the PHP parser. There are four ways to do this:

1) Canonical PHP tags:

The most universally effective PHP tag style is:

```
<?php...?>
```

If you use this style, you can be positive that your tags will always be correctly interpreted.

2) Short-open (SGML-style) tags:

Short or short-open tags look like this:

```
<?...?>
```

Short tags are, as one might expect, the shortest option You must do one of two things to enable PHP to recognize the tags:

- Choose the --enable-short-tags configuration option when you're building PHP.
- Set the short_open_tag setting in your php.ini file to on. This option must be disabled to parse XML with PHP because the same syntax is used for XML tags.

3) ASP-style tags:

ASP-style tags mimic the tags used by Active Server Pages to delineate code blocks. ASP-style tags look like this:

```
<%...%>
```

To use ASP-style tags, you will need to set the configuration option in your php.ini file.

4) HTML script tags:

HTML script tags look like this:

```
<script language="PHP">...</script>
```

While the tags seen in one and two are both always available, one is the most commonly used, and recommended, of the two.

Short tags are only available when they are enabled via the **short_open_tag** *php.ini* configuration file directive, or if PHP was configured with the**--enable-short-tags** option.

ASP style tags (example four) are only available when they are enabled via the asp_tags *php.ini* configuration file directive.

Note: Using short tags should be avoided when developing applications or libraries that are meant for redistribution, or deployment on PHP servers which are not under your control, because short tags may not be supported on the target server. For portable, redistributable code, be sure not to use short tags.

Note: In PHP 5.2 and earlier, the parser does not allow the *<?php* opening tag to be the only thing in a file. This is allowed as of PHP 5.3.

Embedding PHP and HTML

Most of the time you will see PHP embedded in HTML documents, as in this example.

```
<p>This is going to be ignored.</p>
<?php echo 'While this is going to be parsed.'; ?>
<p>This will also be ignored.</p>
```

You can also use more advanced structures:

Example #1 Advanced embedding

```
<?php
if ($expression)
        {
?>
<strong>This is true.</strong>
<?php
        }
else
        {
?>
<strong>This is false.</strong>
<?php
        }
?>
```

This works as expected, because when PHP hits the ?> closing tags, it simply starts outputting whatever it finds (except for an immediately following newline) until it hits another opening tag. The example given here is contrived, of course, but for outputting large blocks of text, dropping out of PHP parsing mode is generally more efficient than sending all of the text through echo() or print().

Note: Also note that if you are embedding PHP within XML or XHTML you will need to use the <?php ?> tags to remain compliant with standards.

PHP is whitespace insensitive:

Whitespace is the stuff you type that is typically invisible on the screen, including spaces, tabs, and carriage returns (end-of-line characters).

PHP whitespace insensitive means that it almost never matters how many whitespace characters you have in a row. One whitespace character is the same as many such characters

For example, each of the following PHP statements that assigns the sum of 2 + 2 to the variable $four is equivalent:

```
$four = 2 + 2; // single spaces
$four <tab>=<tab2<tab>+<tab>2 ; // spaces and tabs
```

```
$four =
2+
2; // multiple lines
```

PHP is case sensitive:

Yeah it is true that PHP is a case sensitive language. Try out following example:

```
<html>
<body>
<?
$capital = 67;
print("Variable capital is $capital<br>");
print("Variable CaPiTaL is $CaPiTaL<br>");
?>
</body>
</html>
```

This will produce following result:

```
Variable capital is 67
Variable CaPiTaL is
```

Statements are expressions terminated by semicolons:

A *statement* in PHP is any expression that is followed by a semicolon (;).Any sequence of valid PHP statements that is enclosed by the PHP tags is a valid PHP program. Here is a typical statement in PHP, which in this case assigns a string of characters to a variable called $greeting:

```
$greeting = "Welcome to PHP!";
```

Expressions are combinations of tokens:

The smallest building blocks of PHP are the indivisible tokens, such as numbers (3.14159), strings (.two.), variables ($two), constants (TRUE), and the special words that make up the syntax of PHP itself like if, else, while, for and so forth

Braces make blocks:

Although statements cannot be combined like expressions, you can always put a sequence of statements anywhere a statement can go by enclosing them in a set of curly braces.

Here both statements are equivalent:

```
if (3 == 2 + 1)
  print("Good - I haven't totally lost my mind.<br>");

if (3 == 2 + 1)
{
  print("Good - I haven't totally");
  print("lost my mind.<br>");
}
```

Running PHP Script from Command Prompt:

You can run your PHP script on your command prompt. Assuming you have following content in test.php file

```
<?php
  echo "Hello PHP!!!!!";
?>
```

Now run this script as command prompt as follows:

```
$ php test.php
```

It will produce following result:

```
Hello PHP!!!!!
```

Hope now you have basic knowledge of PHP Syntax.

DATA TYPES in PHP

PHP supports eight primitive types. Four scalar types:

- Boolean
- Integer
- float (floating-point number, aka double)
- string

Two compound types:

- array
- object

And finally two special types:

- resource
- NULL

This manual also introduces some pseudo-types for readability reasons:

- Mixed
- Number
- callback

And the pseudo-variable *$...*.

Some references to the type "double" may remain in the manual. Consider double the same as float; the two names exist only for historic reasons.

The type of a variable is not usually set by the programmer; rather, it is decided at runtime by PHP depending on the context in which that variable is used.

Note: To check the type and value of an expression, use the var_dump() function.

To get a human-readable representation of a type for debugging, use the gettype() function. To check for a certain type, do ***not*** use gettype(), but rather the *is_type* functions.

Some examples:

```
<?php
        $a_bool = TRUE;         // a boolean
        $a_str  = "foo";  // a string
        $a_str2 = 'foo';  // a string
        $an_int = 12;   // an integer
        echo gettype($a_bool); // prints out:  boolean
        echo gettype($a_str);  // prints out:  string
        // If this is an integer, increment it by four
        if (is_int($an_int))
                {
```

```
                $an_int += 4;
                }
        // If $a_bool is a string, print it out
        // (does not print out anything)
        if (is_string($a_bool))
                {
                echo "String: $a_bool";
                }
?>
```

To forcibly convert a variable to a certain type, either cast the variable or use the settype() function on it.

Note that a variable may be evaluated with different values in certain situations, depending on what type it is at the time. For more information, see the section on Type Juggling. The type comparison tables may also be useful, as they show examples of various type-related comparisons.

Booleans

This is the simplest type. A boolean expresses a truth value. It can be either **TRUE** or **FALSE**.

Note: The boolean type was introduced in PHP 4.

Syntax

To specify a boolean literal, use the keywords TRUE or FALSE. Both are case-insensitive.

Example

```
<?php
$foo = True; // assign the value TRUE to $foo
?>
```

Typically, some kind of operator which returns a boolean value, and the value is passed on to a control structure.

```
<?php
        // == is an operator which test
        // equality and returns a boolean
        if ($action == "show_version")
        {
        echo "The version is 1.23";
        }
        // this is not necessary...
        if ($show_separators == TRUE)
        {
        echo "<hr>\n";
        }
        // ...because instead, this can be used:
        if ($show_separators)
        {
        echo "<hr>\n";
        }
?>
```

Converting to boolean

To explicitly convert a value to Boolean, use the *(bool)* or *(boolean)* casts. However, in most cases the cast is unnecessary, since a value will be automatically converted if an operator, function or control structure requires a Boolean argument.

When converting to Boolean, the following values are considered **FALSE**:

- the boolean **FALSE**itself
- the integer 0 (zero)
- the float 0.0 (zero)
- the empty string, and the string "0"
- an array with zero elements
- an object with zero member variables (PHP 4 only)
- the special type NULL (including unset variables)
- SimpleXML objects created from empty tags

Integers

An integer is a number among the set {…,-2,-1,0,1,2…}

Syntax

Integers can be specified in decimal (base 10), hexadecimal (base 16), or octal (base 8) notation, optionally preceded by a sign (- or +).

To use octal notation, precede the number with a *0* (zero). To use hexadecimal notation precede the number with *0x*.

Example #1 Integer literals

```
<?php
$a = 1234; // decimal number
$a = -123; // a negative number
$a = 0123; // octal number (equivalent to 83 decimal)
$a = 0x1A; // hexadecimal number (equivalent to 26 decimal)
?>
```

Formally, the structure for integer literals is:

```
decimal     : [1-9][0-9]*| 0
hexadecimal : 0[xX][0-9a-fA-F]+
octal       : 0[0-7]+
integer     : [+-]?decimal| [+-]?hexadecimal| [+-]?octal
```

Floating point numbers

Floating point numbers (also known as "floats", "doubles", or "real numbers") can be specified using any of the following syntaxes:

```
<?php
$a = 1.234;
$b = 1.2e3;
$c = 7E-10;
?>
```

Formally:

```
LNUM  [0-9]+
DNUM ([0-9]*[\.]{LNUM}) | ({LNUM}[\.][0-
9]*) EXPONENT_DNUM [+-]?(({LNUM} |
{DNUM}) [eE][+-]? {LNUM})
```

The size of a float is platform-dependent, although a maximum of ~1.8e308 with a precision of roughly 14 decimal digits is a common value (the 64 bit IEEE format).

Strings

A string is series of characters, where a character is the same as a byte. This means that PHP only supports a 256-character set, and hence does not offer native Unicode support.

Note: There is no problem for a string to become very large. PHP imposes no boundary on the size of a string; the only limit is the available memory of the computer on which PHP is running.

Syntax

A string literal can be specified in four different ways:

- single quoted
- double quoted
- heredoc syntax
- nowdoc syntax (since PHP 5.3.0)

Single quoted

The simplest way to specify a string is to enclose it in single quotes (the character '). To specify a literal single quote, escape it with a backslash (\). To specify a literal backslash, double it (\\). All other instances of backslash will be treated as a literal backslash: this means that the other escape sequences you might be used to, such as *r* or *n*, will be output literally as specified rather than having any special meaning.

Note: Unlike the double-quoted and heredoc syntaxes, variables and escape sequences for special characters will ***not*** be expanded when they occur in single quoted strings.

```
<?php
echo 'this is a simple string';
echo 'You can also have embedded newlines in strings this way as it is okay to do';
// Outputs: Arnold once said: "I'll be back"
echo 'Arnold once said: "I\'ll be back"';
// Outputs: You deleted C:\*.*?
echo 'You deleted C:\\*.*?';
// Outputs: You deleted C:\*.*?
echo 'You deleted C:\*.*?';
// Outputs: This will not expand: \n a newline
echo 'This will not expand: \n a newline';
// Outputs: Variables do not $expand $either
echo 'Variables do not $expand $either';
?>
```

Double quoted

If the string is enclosed in double-quotes ("), PHP will interpret more escape sequences for special characters:

Escaped characters	
Sequence	**Meaning**
n	linefeed (LF or 0x0A (10) in ASCII)

\r	carriage return (CR or 0x0D (13) in ASCII)
\t	horizontal tab (HT or 0x09 (9) in ASCII)
\v	vertical tab (VT or 0x0B (11) in ASCII) (since PHP 5.2.5)
\f	form feed (FF or 0x0C (12) in ASCII) (since PHP 5.2.5)
**	backslash
\$	dollar sign
\"	double-quote
\[0-7]{1,3}	the sequence of characters matching the regular expression is a character in octal notation
\x[0-9A-Fa-	the sequence of characters matching the regular expression is a character in hexadecimal

As in single quoted strings, escaping any other character will result in the backslash being printed too. Before PHP 5.1.1, the backslash in *\{$var}* had not been printed.

PHP Variable

The main way to store information in the middle of a PHP program is by using a variable. Here are the most important things to know about variables in PHP.

- All variables in PHP are denoted with a leading dollar sign ($).
- The value of a variable is the value of its most recent assignment.
- Variables are assigned with the = operator, with the variable on the left-hand side and the expression to be evaluated on the right.
- Variables can, but do not need, to be declared before assignment.
- Variables in PHP do not have intrinsic types - a variable does not know in advance whether it will be used to store a number or a string of characters.
- Variables used before they are assigned have default values.
- PHP does a good job of automatically converting types from one to another when necessary.
- PHP variables are Perl-like.

Variable Naming:

Rules for naming a variable are:

- Variable names must begin with a letter or underscore character.
- A variable name can consist of numbers, letters, underscores but you cannot use characters like + , - , % , (,) . & , etc
- There is no size limit for variables.

PHP has a total of eight data types which we use to construct our variables:

- **Integers:** are whole numbers, without a decimal point, like 4195.
- **Doubles:** are floating-point numbers, like 3.14159 or 49.1.
- **Booleans:** have only two possible values either true or false.
- **NULL:** is a special type that only has one value: NULL.
- **Strings:** are sequences of characters, like 'PHP supports string operations.'
- **Arrays:** are named and indexed collections of other values.
- **Objects:** are instances of programmer-defined classes, which can package up both other kinds of values and functions that are specific to the class.
- **Resources:** are special variables that hold references to resources external to PHP (such as database connections).

The first five are *simple types*, and the next two (arrays and objects) are compound - the compound types can package up other arbitrary values of arbitrary type, whereas the simple types cannot.

We will explain only simple data type in this chapter.

Integers:

They are whole numbers, without a decimal point, like 4195. They are the simplest type .they correspond to simple whole numbers, both positive and negative. Integers can be assigned to variables, or they can be used in expressions, like so:

```
$int_var = 12345;
$another_int = -12345 + 12345;
```

Integer can be in decimal (base 10), octal (base 8), and hexadecimal (base 16) format. Decimal format is the default, octal integers are specified with a leading 0, and hexadecimals have a leading 0x.

For most common platforms, the largest integer is (2**31 . 1) (or 2,147,483,647), and the smallest (most negative) integer is . (2**31 . 1) (or .2,147,483,647).

Doubles:

They like 3.14159 or 49.1. By default, doubles print with the minimum number of decimal places needed. For example, the code:

```
$many = 2.2888800;
$many_2 = 2.2111200;
$few = $many + $many_2;
print(.$many + $many_2 = $few<br>.);
```

It produces the following browser output:

```
2.28888 + 2.21112 = 4.5
```

Boolean:

They have only two possible values either true or false. PHP provides a couple of constants especially for use as Booleans: TRUE and FALSE, which can be used like so:

```
if (TRUE)
   print("This will always print<br>");
else
   print("This will never print<br>");
```

Interpreting other types as Booleans:

Here are the rules for determine the "truth" of any value not already of the Boolean type:

- If the value is a number, it is false if exactly equal to zero and true otherwise.
- If the value is a string, it is false if the string is empty (has zero characters) or is the string "0", and is true otherwise.
- Values of type NULL are always false.
- If the value is an array, it is false if it contains no other values, and it is true otherwise. For an object, containing a value means having a member variable that has been assigned a value.

- Valid resources are true (although some functions that return resources when they are successful will return FALSE when unsuccessful).
- Don't use double as Booleans.

Each of the following variables has the truth value embedded in its name when it is used in a Boolean context.

```
$true_num = 3 + 0.14159;
$true_str = "Tried and true"
$true_array[49] = "An array element";
$false_array = array();
$false_null = NULL;
$false_num = 999 - 999;
$false_str = "";
```

NULL:

NULL is a special type that only has one value: NULL. To give a variable the NULL value, simply assign it like this:

```
$my_var = NULL;
```

The special constant NULL is capitalized by convention, but actually it is case insensitive; you could just as well have typed:

```
$my_var = null;
```

A variable that has been assigned NULL has the following properties:

- It evaluates to FALSE in a Boolean context.
- It returns FALSE when tested with IsSet() function.

Strings:

They are sequences of characters, like "PHP supports string operations". Following are valid examples of string

```
$string_1 = "This is a string in double quotes";
$string_2 = "This is a somewhat longer, singly quoted string";
$string_39 = "This string has thirty-nine characters";
$string_0 = ""; // a string with zero characters
```

Singly quoted strings are treated almost literally, whereas doubly quoted strings replace variables with their values as well as specially interpreting certain character sequences.

```
<?
$variable = "name";
$literally = 'My $variable will not print!\\n';
print($literally);
$literally = "My $variable will print!\\n";
print($literally);
?>
```

This will produce following result:

```
My $variable will not print!\n
My name will print
```

There are no artificial limits on string length - within the bounds of available memory, you ought to be able to make arbitrarily long strings.

Strings that are delimited by double quotes (as in "this") are preprocessed in both the following two ways by PHP:

- Certain character sequences beginning with backslash (\) are replaced with special characters
- Variable names (starting with $) are replaced with string representations of their values.

The escape-sequence replacements are:

- \n is replaced by the newline character
- \r is replaced by the carriage-return character
- \t is replaced by the tab character
- \$ is replaced by the dollar sign itself ($)
- \" is replaced by a single double-quote (")
- \\ is replaced by a single backslash (\)

Bit more about Variables

As a regular expression, it would be expressed thus: '*[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]**'

Note: For our purposes here, a letter is a-z, A-Z, and the bytes from 127 through 255 (*0x7f-0xff*).

Note: *$this* is a special variable that can't be assigned.

```
<?php
$var = 'Bob';
$Var = 'Joe';
echo "$var, $Var";      // outputs "Bob, Joe"
$4site = 'not yet';     // invalid; starts with a number
$_4site = 'not yet';    // valid; starts with an underscore
$täyte = 'mansikka';  // valid; 'ä' is (Extended) ASCII 228.
?>
```

By default, variables are always assigned by value. That is to say, when you assign an expression to a variable, the entire value of the original expression is copied into the destination variable. This means, for instance, that after assigning one variable's value to another, changing one of those variables will have no effect on the other. For more information on this kind of assignment, see the chapter on Expressions.

PHP also offers another way to assign values to variables: assign by reference.

This means that the new variable simply references (in other words, "becomes an alias for" or "points to") the original variable. Changes to the new variable affect the original, and vice versa.

To assign by reference, simply prepend an ampersand (&) to the beginning of the variable which is being assigned (the source variable). For instance, the following code snippet outputs '*My name is viral*' twice:

```
<?php
$foo = 'viral;              // Assign the value 'viral' to $foo
$bar = &$foo;               // Reference $foo via $bar.
$bar = "My name is $bar";  // Alter bar... echo $bar;
echo $foo;                  // $foo is altered too.?>
```

One important thing to note is that only named variables may be assigned by reference.

```
<?php
$foo = 25;
$bar = &$foo;    // This is a valid assignment.
$bar = &(24 * 7);  // Invalid; references an unnamed expression.
function test()
{
return 25;
}
$bar = &test();  // Invalid.
?>
```

It is not necessary to initialize variables in PHP however it is a very good practice. Uninitialized variables have a default value of their type depending on the context in which they are used - booleans default to **FALSE**, integers and floats default to zero, strings (e.g. used in echo()) are set as

an empty string and arrays become to an empty array.

Example

Default values of uninitialized variables

```
<?php
// Unset AND unreferenced (no use context) variable; outputs NULL
var_dump($unset_var);
// Boolean usage; outputs 'false' (See ternary operators for more on this syntax)
echo($unset_bool ? "true\n" : "false\n");
// String usage; outputs 'string(3) "abc"'
$unset_str .= 'abc';
var_dump($unset_str);
// Integer usage; outputs 'int(25)'
$unset_int += 25; // 0 + 25 => 25
var_dump($unset_int);
// Float/double usage; outputs 'float(1.25)'
$unset_float += 1.25;
var_dump($unset_float);
// Array usage; outputs array(1) {  [3]=>  string(3) "def" }
$unset_arr[3] = "def"; // array() + array(3 => "def") => array(3 => "def")
var_dump($unset_arr);
// Object usage; creates new stdClass object (see http://www.php.net/manual/en/reserved.classes.php)
// Outputs: object(stdClass)#1 (1) {  ["foo"]=>  string(3) "bar" }
$unset_obj->foo = 'bar';
var_dump($unset_obj);
?>
```

Relying on the default value of an uninitialized variable is problematic in the case of including one file into another which uses the same variable name. It is also a major security risk with register_globals turned on. E_NOTICE level error is issued in case of working with uninitialized variables, however not in the case of appending elements to the uninitialized array. isset() language construct can be used to detect if a variable has been already initialized.

Variable Scope:

The scope of a variable is the context within which it is defined. For the most part all PHP variables only have a single scope. This single scope spans included and required files as well. Scope can be defined as the range of availability a variable has to the program in which it is declared. PHP variables can be one of four scope types:

- Local variables
- Function parameters
- Global variables
- Static variables

- **Local Variables:**

A variable declared in a function is considered local; that is, it can be referenced solely in that function. Any assignment outside of that function will be considered to be an entirely different variable from the one contained in the function:

```
<?
$x = 4;
function assignx () {
$x = 0;
print "\$x inside function is $x.
";
}
assignx();
```

```
print "\$x outside of function is $x.
";
?>
```

This will produce following result.

```
$x inside function is 0.
$x outside of function is 4.
```

Function Parameters:

Function parameters are declared after the function name and inside parentheses. They are declared much like a typical variable would be:

```
<?
// multiply a value by 10 and return it to the caller
function multiply ($value)
{
   $value = $value * 10;
   return $value;
}
$retval = multiply (10);
Print "Return value is $retval\n";
?>
```

This will produce following result.

```
Return value is 100
```

Global Variables:

In contrast to local variables, a global variable can be accessed in any part of the program. However, in order to be modified, a global variable must be explicitly declared to be global in the function in which it is to be modified. This is accomplished, conveniently enough, by placing the keyword **GLOBAL** in front of the variable that should be recognized as global. Placing this keyword in front of an already existing variable tells PHP to use the variable having that name.

Consider an example:

```
<?
$somevar = 15;
function addit() {
GLOBAL $somevar;
$somevar++;
print "Somevar is $somevar";
}
addit();
?>
```

This will produce following result.

```
Somevar is 16
```

Example

Using *global*

```
<?php
$a = 1;
$b = 2;
function Sum()
```

```
{
    global $a, $b;
    $b = $a + $b;
}
Sum();
echo $b;
?>
```

The above script will output *3*. By declaring *$a* and *$b* global within the function, all references to either variable will refer to the global version. There is no limit to the number of global variables that can be manipulated by a function.

A second way to access variables from the global scope is to use the special PHP-defined *$GLOBALS* array. The previous example can be rewritten as:

Example

Using *$GLOBALS* instead of global

```
<?php
$a = 1;
$b = 2;
function Sum()
        {
        $GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b'];
        }
Sum();
echo $b;
?>
```

The *$GLOBALS* array is an associative array with the name of the global variable being the key and the contents of that variable being the value of the array element. Notice how *$GLOBALS* exists in any scope, this is because *$GLOBALS* is a superglobal.

Here's an example demonstrating the power of superglobals:

Example

Demonstrating superglobals and scope

```
<?php
function test_global()
{
// Most predefined variables aren't "super" and require ‘global’ to be available to the functions local scope.
global $HTTP_POST_VARS;
echo $HTTP_POST_VARS['name'];
// Superglobals are available in any scope and do not require 'global'. Superglobals are available as of PHP 4.1.0,
and HTTP_POST_VARS is now deemed deprecated.
echo $_POST['name'];
}
?>
```

Note:Using *global* keyword outside a function is not an error. It can be used if the file is included from inside a function.

Static Variables:

The final type of variable scoping that I discuss is known as static. In contrast to the variables declared as function parameters, which are destroyed on the function's exit, a static variable will not lose its value when the function exits and will still hold that value should the function be called again.

You can declare a variable to be static simply by placing the keyword STATIC in front of the variable name.

```
<?
function keep_track() {
   STATIC $count = 0;
   $count++;
   print $count;
   print "
";
}
keep_track();
keep_track();
keep_track();
?>
```

This will produce following result.

```
1
2
3
```

Example

Demonstrating need for static variables

```
<?php
function test()
{
$a = 0;
echo $a;
$a++;
}
?>
```

This function is quite useless since every time it is called it sets *$a* to *0* and prints *0*. The *$a*++ which increments the variable serves no purpose since as soon as the function exits the *$a* variable disappears.

To make a useful counting function which will not lose track of the current count, the *$a* variable is declared static:

Example

Use of static variables

```
<?php
Function Test()
        {
        static $a = 0;
        echo $a;
        $a++;
        }
?>
```

Now, *$a* is initialized only in first call of function and every time the *test()* function is called it will print the value of *$a* and increment it.

Static variables also provide one way to deal with recursive functions. A recursive function is one which calls itself. Care must be taken when writing a recursive function because it is possible to make it recurse indefinitely. You must make sure you have an adequate way of terminating the recursion.

The following simple function recursively counts to 10, using the static variable *$count* to know when to stop:

Example

Static variables with recursive functions

```
<?php
function test()
        {
        static $count = 0;
        $count++;
        echo $count;
        if ($count < 10) {
           test();
        }
        $count--;
}
?>
```

Note: Static variables may be declared as seen in the examples above. Trying to assign values to these variables which are the result of expressions will cause a parse error.

Example

Declaring static variables

```
<?php
function foo(){
static $int = 0;              // correct
static $int = 1+2;            // wrong  (as it is an expression)
static $int = sqrt(121);  // wrong  (as it is an expression too)
$int++;
echo $int;
}        ?>
```

Note: Static declarations are resolved in compile-time.

Note: Using *global* keyword outside a function is not an error. It can be used if the file is included from inside a function.

PHP Operator Types

What is Operator? Simple answer can be given using expression *4 + 5 is equal to 9*. Here 4 and 5 are called operands and + is called operator.

PHP language supports following type of operators.

- Arithmetic Operators
- Comparision Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Lets have a look on all operators one by one.

Arithmatic Operators:

There are following arithmatic operators supported by PHP language:

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description
+	Adds two operands
-	Subtracts second operand from the first
*	Multiply both operands
/	Divide numerator by denumerator

%	Modulus Operator and remainder of after an integer division
++	Increment operator, increases integer value by one
--	Decrement operator, decreases integer value by one

Example

```
<html>
<head><title>Arithmetical Operators</title><head>
<body>
<?php
   $a = 42;
   $b = 20;
   $c = $a + $b;
   echo "Addtion Operation Result: $c <br/>";
   $c = $a - $b;
   echo "Substraction Operation Result: $c <br/>";
   $c = $a * $b;
   echo "Multiplication Operation Result: $c <br/>";
   $c = $a / $b;
   echo "Division Operation Result: $c <br/>";
   $c = $a % $b;
   echo "Modulus Operation Result: $c <br/>";
   $c = $a++;
   echo "Increment Operation Result: $c <br/>";
   $c = $a--;
   echo "Decrement Operation Result: $c <br/>";
?>
</body>
</html>
```

This will produce following result

```
Addtion Operation Result: 62
Substraction Operation Result: 22
Multiplication Operation Result: 840
Division Operation Result: 2.1
Modulus Operation Result: 2
Increment Operation Result: 42
Decrement Operation Result: 43
```

Comparison Operators:

There are following comparison operators supported by PHP language
Assume variable A holds 10 and variable B holds 20 then:

Operator	Description
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.

>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.

Example

```
<html>
<head><title>Comparision Operators</title><head>
<body>
<?php
   $a = 42;
   $b = 20;

   if( $a == $b ){
      echo "TEST1 : a is equal to b<br/>";
   }else{
      echo "TEST1 : a is not equal to b<br/>";
   }

   if( $a > $b ){
      echo "TEST2 : a is greater than  b<br/>";
   }else{
      echo "TEST2 : a is not greater than b<br/>";
   }
   if( $a < $b ){
      echo "TEST3 : a is less than  b<br/>";
   }else{
      echo "TEST3 : a is not less than b<br/>";
   }
   if( $a != $b ){
      echo "TEST4 : a is not equal to b<br/>";
   }else{
      echo "TEST4 : a is equal to b<br/>";
   }
   if( $a >= $b ){
      echo "TEST5 : a is either grater than or equal to b<br/>";
   }else{
      echo "TEST5 : a is nieghter greater than nor equal to b<br/>";
   }
   if( $a <= $b ){
      echo "TEST6 : a is either less than or equal to b<br/>";
   }else{
      echo "TEST6 : a is nieghter less than nor equal to b<br/>";
   }
?>
</body>
</html>
```

This will produce following result

```
TEST1 : a is not equal to b
TEST2 : a is greater than b
TEST3 : a is not less than b
TEST4 : a is not equal to b
TEST5 : a is either grater than or equal to b
TEST6 : a is nieghter less than nor equal to b
```

Logical Operators:

There are following logical operators supported by PHP language
Assume variable A holds 10 and variable B holds 20 then:

Operator	Description
And	Called Logical AND operator. If both the operands are true then then condition becomes true.
Or	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.
&&	Called Logical AND operator. If both the operands are non zero then then condition becomes true.
||	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.

Example

```
<html>
<head><title>Logical Operators</title><head>
<body>
<?php
   $a = 42;
   $b = 0;

   if( $a && $b ){
      echo "TEST1 : Both a and b are true<br/>";
   }else{
      echo "TEST1 : Either a or b is false<br/>";
   }
   if( $a and $b ){
      echo "TEST2 : Both a and b are true<br/>";
   }else{
      echo "TEST2 : Either a or b is false<br/>";
   }
   if( $a || $b ){
      echo "TEST3 : Either a or b is true<br/>";
   }else{
      echo "TEST3 : Both a and b are false<br/>";
   }
   if( $a or $b ){
      echo "TEST4 : Either a or b is true<br/>";
   }else{
      echo "TEST4 : Both a and b are false<br/>";
   }
   $a = 10;
   $b = 20;
   if( $a ){
      echo "TEST5 : a is true <br/>";
   }else{
      echo "TEST5 : a  is false<br/>";
   }
   if( $b ){
```

```
      echo "TEST6 : b is true <br/>";
   }else{
      echo "TEST6 : b  is false<br/>";
   }
   if( !$a ){
      echo "TEST7 : a is true <br/>";
   }else{
      echo "TEST7 : a  is false<br/>";
   }
   if( !$b ){
      echo "TEST8 : b is true <br/>";
   }else{
      echo "TEST8 : b  is false<br/>";
   }
?>
</body>
</html>
```

This will produce following result

```
TEST1 : Either a or b is false
TEST2 : Either a or b is false
TEST3 : Either a or b is true
TEST4 : Either a or b is true
TEST5 : a is true
TEST6 : b is true
TEST7 : a is false
TEST8 : b is false
```

Assignment Operators:

There are following assignment operators supported by PHP language:

Operator	Description
=	Simple assignment operator, Assigns values from right side operands to left side operand
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand

Example

```
<html>
<head><title>Assignment Operators</title><head>
<body>
<?php
   $a = 42;
   $b = 20;
```

```
   $c = $a + $b;   /* Assignment operator */
   echo "Addtion Operation Result: $c <br/>";
   $c += $a;  /* c value was 42 + 20 = 62 */
   echo "Add AND Assigment Operation Result: $c <br/>";
   $c -= $a; /* c value was 42 + 20 + 42 = 104 */
   echo "Subtract AND Assignment Operation Result: $c <br/>";
   $c *= $a; /* c value was 104 - 42 = 62 */
   echo "Multiply AND Assignment Operation Result: $c <br/>";
   $c /= $a;  /* c value was 62 * 42 = 2604 */
   echo "Division AND Assignment Operation Result: $c <br/>";
   $c %= $a; /* c value was 2604/42 = 62*/
   echo "Modulus AND Assignment Operation Result: $c <br/>";
?>
</body>
</html>
```

This will produce following result

```
Addtion Operation Result: 62
Add AND Assigment Operation Result: 104
Subtract AND Assignment Operation Result: 62
Multiply AND Assignment Operation Result: 2604
Division AND Assignment Operation Result: 62
Modulus AND Assignment Operation Result: 20
```

Conditional Operator

There is one more oprator called conditional operator. This first evaluates an expression for a true or false value and then execute one of the two given statements depending upon the result of the evaluation.

The conditioanl operator has this syntax:

Operator	Description
? :	Conditional Expression

```
<html>
<head><title>Arithmetical Operators</title><head>
<body>
<?php
   $a = 10;
   $b = 20;

   /* If condition is true then assign a to result otheriwse b */
   $result = ($a > $b ) ? $a :$b;
   echo "TEST1 : Value of result is $result<br/>";
   /* If condition is true then assign a to result otheriwse b */
   $result = ($a < $b ) ? $a :$b;
   echo "TEST2 : Value of result is $result<br/>";
?>
</body>
</html>
```

This will produce following result

```
TEST1 : Value of result is 20
TEST2 : Value of result is 10
```

Operators Categories:

All the operators we have discussed above can be categorized into following categories:

- Unary prefix operators, which precede a single operand.
- Binary operators, which take two operands and perform a variety of arithmetic and logical operations.
- The conditional operator (a ternary operator), which takes three operands and evaluates either the second or third expression, depending on the evaluation of the first expression.
- Assignment operators, which assign a value to a variable.

Precedence of PHP Operators:

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator:

For example x = 7 + 3 * 2; Here x is assigned 13, not 20 because operator * has higher precedenace than + so it first get multiplied with 3*2 and then adds into 7.

Here operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedenace operators will be evaluated first.

Category	Operator	Associativity
Unary	! ++ --	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=	Right to left

Commenting PHP Code:

A *comment* is the portion of a program that exists only for the human reader and stripped out before displaying the programs result. There are two commenting formats in PHP:

Single-line comments: They are generally used for short explanations or notes relevant to the local code. Here are the examples of single line comments.

```
<?
# This is a comment, and
# This is the second line of the comment
// This is a comment too. Each style comments only
print "An example with single line comments";
?>
```

Multi-lines printing: Here are the examples to print multiple lines in a single print statement:

```
<?
# First Example
print <<<END
This uses the "here document" syntax to output
multiple lines with $variable interpolation. Note
that the here document terminator must appear on a
line with just a semicolon no extra whitespace!
END;
# Second Example
print "This spans
multiple lines. The newlines will be
output as well";
?>
```

Multi-lines comments: They are generally used to provide pseudocode algorithms and more detailed explanations when necessary. The multiline style of commenting is the same as in C. Here are the example of multi lines comments.

```
<?
/* This is a comment with multiline
   Author : Mohammad Mohtashim
   Purpose: Multiline Comments Demo
   Subject: PHP
*/
print "An example with multi line comments";
?>
```

Exercise

1. How to embed PHP with HTML?
2. What are the Global variables?
3. How to declare Static variables?
4. Is PHP case sensitive?
5. How to run PHP script from command prompt?
6. Difference between single and double quote?
7. What is the difference between $message and $$message?

Chapter – 3

Control Structures

In this Chapter

Conditional Statements

Control Loops

Exit, Die and Return Statements

PHP Array

Without Control Structures, PHP would be much like the first set of directions: linear, sequential, and not overly useful. Control Structures are at the core of programming logic. They allow a script to react differently depending on what has already occurred, or based on user input, and allow the graceful handling of repetitive tasks.

In PHP, there are two primary types of Control Structures: **Conditional Statements** and **Control Loops**.

Conditional Statements in PHP

Conditional Statements allow you to branch the path of execution in a script based on whether a single, or multiple conditions, evaluate to true or false. Put simply, they let you test things and perform various actions based on the results.

The if, elseif ...else and switch statements are used to take decision based on the different condition. You can use conditional statements in your code to make your decisions. PHP supports following threedecision making statements:

- **if...else statement** - use this statement if you want to execute a set of code when a condition is true and another if the condition is not true
- **switch statement** - is used if you want to select one of many blocks of code to be executed, use the Switch statement. The switch statement is used to avoid long blocks of if..elseif..else code.
- **? operator -** evaluates an expression for a true or false value and then execute one of the two given statements depending upon the result of the evaluation.

The If...Else Statement

If you want to execute some code if a condition is true and another code if a condition is false, use the if....else statement.

Syntax

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

Example

The following example will output "Have a nice weekend!" if the current day is Friday, otherwise it will output "Have a nice day!":

```
<html>
<body>
<?php
$d=date("D");
if ($d=="Fri")
  echo "Have a nice weekend!";
else
  echo "Have a nice day!";
?>
</body>

</html>
```

If more than one line should be executed if a condition is true/false, the lines should be enclosed within curly braces:

```
<html>
<body>
<?php
$d=date("D");
if ($d=="Fri")
  {
```

```
  echo "Hello!<br />";
  echo "Have a nice weekend!";
  echo "See you on Monday!";
  }
?>
</body>
</html>
```

The ElseIf Statement

If you want to execute some code if one of several conditions are true use the elseif statement

Syntax

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

Example

The following example will output "Have a nice weekend!" if the current day is Friday, and "Have a nice Sunday!" if the current day is Sunday. Otherwise it will output "Have a nice day!":

```
<html>
<body>
<?php
$d=date("D");
if ($d=="Fri")
  echo "Have a nice weekend!";
elseif ($d=="Sun")
  echo "Have a nice Sunday!";
else
  echo "Have a nice day!";
?>
</body>
</html>
```

The Switch Statement

If you want to select one of many blocks of code to be executed, use the Switch statement. The switch statement is used to avoid long blocks of if..elseif..else code.

Syntax

```
switch (expression)
{
case label1:
  code to be executed if expression = label1;
  break;
case label2:
  code to be executed if expression = label2;
  break;
default:
  code to be executed
  if expression is different
  from both label1 and label2;
}
```

Example

The *switch* statement works in an unusual way. First it evaluates given expression then seeks a lable to match the resulting value. If a matching value is found then the code associated with the matching label will be executed or if none of the lables match then statement will will execute any specified default code.

```
<html>
<body>
<?php
$d=date("D");
switch ($d)
{
case "Mon":
  echo "Today is Monday";
  break;
case "Tue":
  echo "Today is Tuesday";
  break;
case "Wed":
  echo "Today is Wednesday";
  break;
case "Thu":
  echo "Today is Thursday";
  break;
case "Fri":
  echo "Today is Friday";
  break;
case "Sat":
  echo "Today is Saturday";
  break;
case "Sun":
  echo "Today is Sunday";
  break;
default:
  echo "Wonder which day is this ?";
}
?>
</body>
</html>
```

The Ternary Operator

Though Technically an Operator, not a Control Structure, the Ternary Operator, represented by "?", can be used as shorthand for simple If/Else Statements. It can only be used in situations where you want execute a single expression based on whether a single condition is true or false:

Syntax

```
(condition) ? (executes if the condition is true) : (executes if the condition is false);
```

Example

```
<html>
<head><title>Arithmetical Operators</title><head>
<body>
<?php
   $a = 10;
   $b = 20;
      /* If condition is true then assign a to result otheriwse b */
   $result = ($a > $b ) ? $a :$b;
   echo "TEST1 : Value of result is $result<br/>";
   /* If condition is true then assign a to result otheriwse b */
   $result = ($a < $b ) ? $a :$b;
```

```
   echo "TEST2 : Value of result is $result<br/>";
?>
</body>
</html>
```

This will produce following result

```
TEST1 : Value of result is 20
TEST2 : Value of result is 10
```

The condition is contained within the first set of parentheses. If it evaluates to true, then the expression within the second set of parentheses will be performed. Otherwise, the expression in the third set will be performed:

PHP Loop Types

Loops in PHP are used to execute the same block of code a specified number of times. PHP supports following four loop types.

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

We will discuss about **continue** and **break** keywords used to control the loops execution.

The for loop statement

The for statement is used when you know how many times you want to execute a statement or a block of statements.

Syntax

```
for (initialization; condition; increment)
                    {
                code to be executed;
                    }
```

The initializer is used to set the start value for the counter of the num,ber of loop iterations. A variable may be declared here for this purpose and it is tradional to name it $i.

Example

The following example makes five iterations and changes the assigned value of two variables on each pass of the loop:

```
<html>
<body>
<?php
$a = 0;
$b = 0;
for( $i=0; $i<5; $i++ )
{
   $a += 10;
   $b += 5;
}
echo ("At the end of the loop a=$a and b=$b" );
?>
</body>
</html>
```

This will produce following result:

```
At the end of the loop a=50 and b=25
```

The while loop statement

The while statement will execute a block of code if and as long as a test expression is true. If the test expression is true then the code block will be executed. After the code has executed the test expression will again be evaluated and the loop will continue until the test expression is found to be false.

Syntax

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

Example

This example decrements a variable value on each iteration of the loop and the counter increments until it reaches 10 when the evaluation is false and the loop ends.

```
<html>
<body>
<?php
$i = 0;
$num = 50;

while( $i < 10)
{
  $num--;
  $i++;
}
echo ("Loop stopped at i = $i and num = $num" );
?>
</body>
</html>
```

This will produce following result:

```
Loop stopped at i = 1 and num = 40
```

The do...while loop statement

The do...while statement will execute a block of code at least once - it then will repeat the loop as long as a condition is true.

Syntax

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

Example

The following example will increment the value of i at least once, and it will continue incrementing the variable i as long as it has a value of less than 10:

```
<html>
<body>
<?php
$i = 0;
$num = 0;
do
{
  $i++;
}while( $i < 10 );
echo ("Loop stopped at i = $i" );
?>
</body>
```

```
</html>
```

This will produce following result:

```
Loop stopped at i = 10
```

The foreach loop statement

The foreach statement is used to loop through arrays. For each pass the the value of the current array element is assigned to $value and the array pointer is moved by one and in the next pass next element will be processed.

Syntax

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

Example

Try out following example to list out the values of an array.

```
<html>
<body>
<?php
$array = array( 1, 2, 3, 4, 5);
foreach( $array as $value )
{
  echo "Value is $value <br />";
}
?>
</body>
</html>
```

This will produce following result:

```
Value is 1
Value is 2
Value is 3
Value is 4
Value is 5
```

The break statement

The PHP **break** keyword is used to terminate the execution of a loop prematurely. The **break** statement is situated inside the statement block. If gives you full control and whenever you want to exit from the loop you can come out. After coming out of a loop immediate statement to the loop will be executed.

Example

In the following example condition test becomes true when the counter value reaches 3 and loop terminates.

```
<html>
<body>
<?php
$i = 0;
while( $i < 10)
{
   $i++;
   if( $i == 3 )break;
}
echo ("Loop stopped at i = $i" );
```

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

This will produce following result:

```
Loop stopped at i = 3
```

The continue statement

The PHP **continue** keyword is used to halt the current iteration of a loop but it does not terminate the loop. Just like the **break** statement the **continue** statement is situated inside the statement block containing the code that the loop executes, preceded by a conditional test. For the pass encountering **continue** statement, rest of the loop code is skipped and next pass starts.

Example

In the following example loop prints the value of array but for which condition bceoms true it just skip the code and next valuye is printed.

```
<html>
<body>
<?php
$array = array( 1, 2, 3, 4, 5);
foreach( $array as $value )
{
  if( $value == 3 )continue;
  echo "Value is $value <br />";
}
?>
</body>
</html>
```

This will produce following result

```
Value is 1
Value is 2
Value is 4
Value is 5
```

Exit , Die and Return statements

Exit / Die

exit -- Output a message and terminate the current script

Description

void **exit** ([string status])

void **exit** (int status)

Note: This is not a real function, but a language construct. PHP >= 4.2.0 does NOT print the *status* if it is an **integer**.

The **exit()** function terminates execution of the script. It prints *status* just before exiting. If *status* is an **integer**, that value will also be used as the exit status. Exit statuses should be in the range 0 to 254, the exit status 255 is reserved by PHP and shall not be used. The status 0 is used to Terminate the program successfully.

Example 1.

exit() example

```
<?php
$filename = '/path/to/data-file';
$file = fopen($filename, 'r')
exit("unable to open file ($filename)");
or die("unable to open file ($filename)");
```

```
?>
```

Example 2. exit() status example

```
<?php
//exit program normally
exit;
exit();
exit(0);
//exit with an error code
exit(1);
exit(0376); //octal
?>
```

Note: This language construct is equivalent to **die()**.

Return statement

If called from within a function, the **return()** statement immediately ends execution of the current function, and returns its argument as the value of the function call. **return()** will also end the execution of an eval() statement or script file. If called from the global scope, then execution of the current script file is ended. If the current script file was include()ed or require()ed, then control is passed back to the calling file.

Furthermore, if the current script file was include()ed, then the value given to **return()** will be returned as the value of the include() call. If **return()** is called from within the main script file, then script execution ends. If the current script file was named by the auto_prepend_file or auto_append_file configuration options in *php.ini*, then that script file's execution is ended.

For more information, see Returning values.

Note: Note that since **return()** is a language construct and not a function, the parentheses surrounding its arguments are not required. It is common to leave them out, and you actually should do so as PHP has less work to do in this case.

Note: If no parameter is supplied, then the parentheses must be omitted and *NULL* will be returned. Calling **return()** with parentheses but with no arguments will result in a parse error.

Note: You should ***never*** use parentheses around your return variable when returning by reference, as this will not work. You can only return variables by reference, not the result of a statement. If you use *return ($a);* then you're not returning a variable, but the result of the expression *($a)* (which is, of course, the value of *$a*).

PHP Arrays

An array is a data structure that stores one or more similar type of values in a single value. For example if you want to store 100 numbers then instead of defining 100 variables its easy to define an array of 100 length.

There are three different kind of arrays and each array value is accessed using an ID c which is called array index.

- **Numeric array** - An array with a numeric index. Values are stored and accessed in linear fashion
- **Associative array** - An array with strings as index. This stores element values in association with key values rather than in a strict linear index order.
- **Multidimensional array** - An array containing one or more arrays and values are accessed using multiple indices

NOTE: Built-in array functions is given in function reference PHP Array Functions

Numeric Array

These arrays can store numbers, strings and any object but their index will be prepresented by numbers. By defualt array index starts from zero.

Example

Following is the example showing how to create and access numeric arrays. Here we have used **array()** function to create array. This function is explained in function reference.

```
<html>
<body>
<?php
/* First method to create array. */
$numbers = array( 1, 2, 3, 4, 5);
foreach( $numbers as $value )
{
  echo "Value is $value <br />";
}
/* Second method to create array. */
$numbers[0] = "one";
$numbers[1] = "two";
$numbers[2] = "three";
$numbers[3] = "four";
$numbers[4] = "five";
foreach( $numbers as $value )
{
  echo "Value is $value <br />";
}
?>
</body>
</html>
```

This will produce following result:

```
Value is 1
Value is 2
Value is 3
Value is 4
Value is 5
Value is one
Value is two
Value is three
Value is four
Value is five
```

Associative Arrays

The associative arrays are very similar to numeric arrays in term of functionality but they are different in terms of their index. Associative array will have their index as string so that you can establish a strong association between key and values.

To store the salaries of employees in an array, a numerically indexed array would not be the best choice. Instead, we could use the employees names as the keys in our associative array, and the value would be their respective salary.

NOTE: Don't keep associative array inside double quote while printing otheriwse it would not return any value.

Example

```
<html>
<body>
<?php
/* First method to associate create array. */
$salaries = array(
                        "megha" => 15000,
                        "vrishti" => 10000,
                        "Kejal" => 20000
                       );
```

```
echo "Salary of megha is ". $salaries['megha'] . "<br />";
echo "Salary of vrishti is ".  $salaries['vrishti']. "<br />";
echo "Salary of kejal is ".  $salaries['kejal']. "<br />";

/* Second method to create array. */
$salaries['megha'] = "medium";
$salaries['vrishti'] = "low";
$salaries['kejal'] = "high";

echo "Salary of megha is ". $salaries['megha'] . "<br />";
echo "Salary of vrishti is ".  $salaries['vrishti']. "<br />";
echo "Salary of kejal is ".  $salaries['kejal']. "<br />";
?>
</body>
</html>
```

This will produce following result:

```
Salary of megha is 15000
Salary of vrishti is 10000
Salary of kejal is 20000
Salary of megha is medium
Salary of vrishti is low
Salary of kejal is high
```

Multidimensional Arrays

A multi-dimensional array each element in the main array can also be an array. And each element in the sub-array can be an array, and so on. Values in the multi-dimensional array are accessed using multiple index.

Example

In this example we create a two dimensional array to store marks of three students in three subjects:

This example is an associative array, you can create numeric array in the same fashion.

```
<html>
<body>
<?php
  $markes = array(
                    "megha" => array
                    ("physics" => 35,"maths" => 30, "chemistry" => 39),
                    "vrishti" => array
                    ("physics" => 30, "maths" => 32, "chemistry" => 29),
                    "kejal" => array
                    ("physics" => 31, "maths" => 22, "chemistry" => 39)
                  );
  /* Accessing multi-dimensional array values */
  echo "Markes for megha in physics : " ;
  echo $markes['megha']['physics'] . "<br />";
  echo "Markes for vrishti in maths : ";
  echo $markes['vrishti']['maths'] . "<br />";
  echo "Markes for kejal in chamistry : " ;
  echo $markes['kejal']['chemistry'] . "<br />";
?>
</body>
</html>
```

This will produce following result:

```
Markes for megha in physics : 35
```

Markes for vrishti in maths : 32
Markes for kejal in chamistry : 39

Exercise

1. Explain continue.
2. Can we differentiate exit and die?
3. What is the use of Return statement?
4. How to declare array in PHP ?
5. What is an associative array?
6. How to display 2 dimensional arrays?
7. Print_r and var_dump

Chapter – 4

Working with Data

In this Chapter

Form Element

Validating User Input

Client-Side and Server-side

Passing Variables between pages

FORM element

Forms are the most popular way to make web pages interactive. Like forms on paper, a form on a web page allows the user to enter requested information and submit it for processing. (Fortunately, forms on a web page are processed much faster.)

Syntax	**<FORM>**...**</FORM>**
Attribute Specifications	• **ACTION**=*URL* (form handler) • **METHOD**=[get | post] (HTTP method for submitting form) • **ENCTYPE**=*ContentType* (content type to submit form as) • **NAME**=*CDATA* (name for client-side scripting) • **ACCEPT-CHARSET**=*Charsets* (supported character encodings) • **ACCEPT**=*ContentTypes* (media types for file upload) • **ONSUBMIT**=*Script* (form was submitted) • **ONRESET**=*Script* (form was reset)
Contained in	BLOCKQUOTE, BODY, DD, DEL, DIV, FIELDSET, INS, LI, MAP, NOSCRIPT, OBJECT, TD, TH

The **FORM** element defines an *interactive form*. The element should contain form controls--input, select, textarea and button -- through which the user interacts.

When the user submits the form, through an **INPUT** or **BUTTON** element with **TYPE=submit**, the form values are submitted to the URL given in **FORM**'s required **ACTION** attribute. **ACTION** usually points to a server-side script that handles the form submission.

A mailto URL (*e.g.*, **mailto:abc@xyz.com**) is also allowed as an **ACTION**, but this is not supported by all browsers. Even on supporting browsers, mailto forms are troublesome in that they fail to provide feedback to the user after the form submission, and browsers may offer a privacy warning before submitting the form with the user's email address.

How the form input is sent to the server depends on the **METHOD** and **ENCTYPE** attributes. When the **METHOD** is **get** (the default), the form input is submitted as an HTTP GET request with **?***form_data* appended to the URI specified in the **ACTION** attribute.

Using the **get** method allows the form submission to be contained completely in a URL. This can be advantageous in that it permits bookmarking in current browsers. However, the amount of form data that can be handled by the **get** method is limited by the maximum length of the URL that the server and browser can process. To be safe, any form whose input might contain more than a few hundred characters should use **METHOD=post**.

With a **METHOD** value of **post**, the form input is submitted as an HTTP POST request with the form data sent in the body of the request. Most current browsers are unable to bookmark POST requests, but POST does not entail the length restrictions imposed by GET.

The **ENCTYPE** attribute specifies the content type used in submitting the form, and defaults to **application/x-www-form-urlencoded**. This content type results in name/value pairs sent to the server as *name1=value1*&*name2=value2...* with space characters replaced by "+" and reserved characters (like "#") replaced by "%*HH*" where *HH* is the ASCII code of the character in hexadecimal. Line breaks are encoded as "%0D%0A"--a carriage return followed by a line feed.

Authors should generally only use a different **ENCTYPE** when the form includes a **TYPE=file INPUT** element, in which case the **ENCTYPE** should be **multipart/form-data** and the **METHOD** must be **post**. The format of multipart/form-data requests is given in *RFC 2388*.

The **ACCEPT-CHARSET** attribute specifies a list of character encodings that are accepted by the form handler. The value consists of a list of "charsets" separated by commas and/or spaces. The default value is **UNKNOWN** and is usually considered to be the character encoding used to transmit the document containing the **FORM**.

The **ACCEPT** attribute gives a comma-separated list of media types accepted for file upload, allowing the browser to filter out inappropriate files. Current browsers generally ignore the **ACCEPT** attribute.

The **FORM** element also takes a number of attributes to specify client-side scripting actions for various events. In addition to the core events common to most elements,**FORM** accepts the following event attributes:

- **ONSUBMIT**, when the form is submitted;
- **ONRESET**, when the form is reset.

The **NAME** attribute, added to **FORM** in HTML 4.0**1**, specifies a name for referring to the form from a client-side script. The **ID** attribute provides the same functionality, but old browsers such as Netscape 4.*x* only support the **NAME** attribute.

<INPUT ...>

- **TYPE:** what type of field
- **NAME:** name of this form field
- **VALUE:** initial or only value of this field
- **SIZE:** how wide the text field should be
- **MAXLENGTH:** maximum number of characters
- **CHECKED:** check this checkbox or radio button
- **BORDER:** border around image
- **SRC:** URL of image
- **ALT:** text to show if you don't show the picture
- **LOWSRC:** a version of the picture that isn't such a big file
- **WIDTH:** width of image
- **HEIGHT:** height of image
- **ALIGN:** how text should flow around the picture
- **VSPACE:** vertical distance between the picture and the text

- **HSPACE:** horizontal distance between the picture and the text
- **READONLY:** the value of this field cannot be changed
- **DISABLED:** don't let the user do anything with this field
- **TABINDEX:** tab order
- **LANGUAGE:** scripting language to use
- **onClick:** when the user clicks here
- **onChange:** when this field is changed
- **onFocus:** when this field gets the focus
- **onBlur:** when this field loses the focus
- **onKeyPress:** script to run when a key is pressed
- **onKeyUp:** script for when a key goes up while the field has the focus
- **onKeyDown:** script for when a key goes down while the field has the focus
- **AUTOCOMPLETE:** If the browser should use autocompletion for the field

<INPUT ...> creates the data entry fields on an HTML form. (Well, it creates *most* types of fields, <TEXTAREA ...> and<SELECT ...> also create some, as does the new <BUTTON ...> tag.)
The TYPE attribute establishes what type of field the input is creating. The other <INPUT ...> attributes affect different types of inputs different ways (or not at all). So let's jump straight into the TYPE attribute and look at the different types of input fields.

Attributes for <INPUT ...>

TYPE = TEXT | CHECKBOX | RADIO | PASSWORD | HIDDEN | SUBMIT | RESET |BUTTON | FILE | IMAGE

TYPE establishes what type of data entry field this is. The <INPUT ...> tag has ten different types of fields (the<TEXTAREA ...>, <SELECT ...>, and <BUTTON ...> tags create the other types).

TYPE = TEXT

TEXT creates a text entry field (the most popular type of data entry field):

```
<FORM ACTION="testform.php">
Name: <INPUT TYPE=TEXT NAME="realname">
<P><INPUT TYPE=SUBMIT VALUE="submit">
</FORM>
```

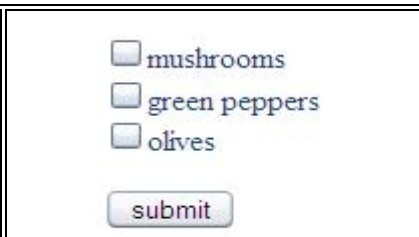
TEXT is the default input type (if you want a text field, you don't even need to use the TYPE attribute). The behavior of TEXT fields can be modified using these attributes:

- VALUE: set an initial value for the field
- SIZE: how wide the field should be
- MAXLENGTH: the maximum number of characters the user can enter

TYPE = CHECKBOX

CHECKBOX creates a checkbox which can be either on or off. CHECKBOX is often used in groups to indicate a series of choices any one of which can be on or off:

```
<FORM ACTION="testform.php">
<INPUT TYPE=CHECKBOX NAME="mushrooms">mushrooms<BR>
<INPUT TYPE=CHECKBOX NAME="greenpeppers">green peppers<BR>
<INPUT TYPE=CHECKBOX NAME="olives">olives<BR>
<P><INPUT TYPE=SUBMIT VALUE="submit">
</FORM>
```

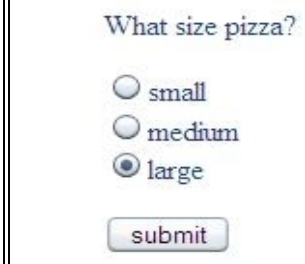


By default, the checkbox is initially off. If you want the checkbox init ially on, use the **CHECKED** attribute. CHECKBOXs are only sent to the CGI if they are on; if they are off, no name/value pair is sent

TYPE = RADIO

RADIO is used to create a series of choices of which only one can be selected. The term "radio button" comes from the buttons for the radio in an automobile, where selecting one radio station automatically de-selects all the others. HTML radio buttons are created by using several <INPUT TYPE=RADIO> buttons, all with the same name, but with different values. For example, this series of buttons allows you to choose one size for a pizza:

```
<FORM ACTION="testform.php">
What size pizza?<P>

<INPUT TYPE=RADIO NAME="pizzasize" VALUE="S" >small<BR>
<INPUT TYPE=RADIO NAME="pizzasize" VALUE="M" >medium<BR>
<INPUT TYPE=RADIO NAME="pizzasize" VALUE="L" checked="checked" >large

<P><P><INPUT TYPE=SUBMIT VALUE="submit">
</FORM>
```



If no CHECKED attribute is used, different browsers have different ways of displaying the initial state of a series of radio buttons. Netscape and MSIE have none of the buttons selected. Mosaic selects the first button.

TYPE = PASSWORD

PASSWORD indicates that the field is for typing in a password. PASSWORD works just like a TEXT type field, with the difference that whatever is typed is not displayed the screen (in case someone is watching over your shoulder or you have to leave the work station). Instead of showing

what you typed in, the browser displays a series of asterisks (*), bullets (·), or something to show *that* you are typing, but not *what* you are typing. So, for example, this code:

```
<FORM ACTION="testform.php" method="post">

name: <INPUT TYPE=TEXT NAME="realname"><BR>
password: <INPUT TYPE=PASSWORD NAME="mypassword">
<P><INPUT TYPE=SUBMIT VALUE="submit">
</FORM>
```

Note that PASSWORD fields are *not* sent encrypted, they are sent in the same manner as all the other elements on the form: in the clear. Note also that when you use PASSWORD you should also set the form METHOD to POST.

TYPE = HIDDEN

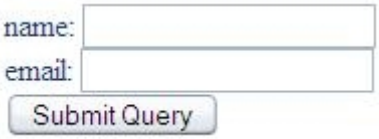
HIDDEN indicates that the field is invisible and the user never interacts with it. The field is still sent to the CGI, and scripts can also use the hidden field. HIDDEN is commonly used as output of a CGI which creates a new form for more input. For example, a web site which facilitates online discussions may use a hidden field to keep track of which message is being responded to:

```
<H2>Your Reply</H2>
<FORM METHOD=POST ACTION="testform.php">

<INPUT TYPE=HIDDEN NAME="postingID" value="98765">
name:    <INPUT NAME="realname" SIZE=20><BR>
email:   <INPUT NAME="email"><BR>
subject: <INPUT NAME="subject" VALUE="Re: important" SIZE=20><br>
comments:<BR>
<TEXTAREA NAME="comments" COLS=25 ROWS=3 WRAP=VIRTUAL>
Message Contents...
</TEXTAREA><br>
<INPUT TYPE=SUBMIT VALUE="Send It!">

</FORM>
```

TYPE = SUBMIT

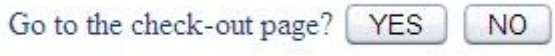
SUBMIT creates the "Submit" button which sends the form in to the CGI. In its simplest form, you can use SUBMIT and no other attributes for the <INPUT ...> tag:

```
<FORM METHOD=POST ACTION="testform.php">
name:    <INPUT NAME="realname"><BR>
email:   <INPUT NAME="contact"><BR>
<INPUT TYPE=SUBMIT>
</FORM>
```



You may sometimes find that you want to have more than one submit button on a form. If you give each button the same name, but different values, the browser will indicate which submit button was pressed:

```
<FORM METHOD=POST ACTION="testform.php">
Go to the check-out page?
<INPUT TYPE=SUBMIT NAME="checkout" VALUE="YES">
<INPUT TYPE=SUBMIT NAME="checkout" VALUE="NO">
</FORM>
```



TYPE = RESET

RESET resets the form so that it is the way it was before anything was typed in:

```
<FORM METHOD=POST ACTION="testform.php">
<INPUT TYPE=TEXT><INPUT TYPE=SUBMIT>
<INPUT TYPE=RESET>
</FORM>
```

For a while it was the perception that all forms "had" to have a reset button, but designers have found that resets are more likely to detract from the form than add to it. Users don't usually need to reset their forms, and they are more likely to*accidentally* hit the reset button than they are to actually want to wipe out their work. Unless you have a specific reason to expect that users will need a reset button it's probably best to leave it out.

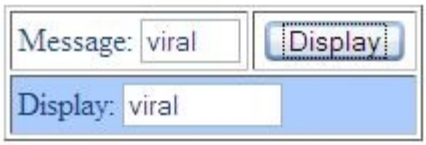
```
<FORM METHOD=POST ACTION="testform.php"
onReset="return confirm('Do you really want to reset the
form?')" >
<INPUT TYPE=TEXT NAME="query">
<INPUT TYPE=SUBMIT>
<INPUT TYPE=RESET></FORM>
```

If you do choose to use have a reset button in your form, consider adding a check if the user actually wants to reset. You can do this by adding an onReset event handler to the <FORM ...> tag:

TYPE = BUTTON

BUTTON defines a button which causes a script to run. Use the onClick attribute to give the script command(s). BUTTON is used *only* with scripting. Browsers that don't understand scripts don't understand this type of input and usually render it as a text input field.

```
<FORM>
<TABLE BORDER CELLPADDING=3>
<TR>
   <TD><NOBR>Message: <INPUT NAME="message" SIZE=4></TD>
   <TD><INPUT TYPE=BUTTON VALUE="Display"
OnClick="this.form.Display.value=this.form.message.value" ></TD>
</TR>
<TR>
   <TD BGCOLOR="#AACCFF" colspan=2>
   Display: <INPUT NAME="Display" SIZE=9><BR></TD>
   </TR>
</TABLE>
</FORM>
```



TYPE = FILE

FILE is used for doing file uploads in a form. File uploads are a relatively new and still not well-standardized type of form input, but they show great promise once the bugs are ironed out. File uploads allow you to send an entire file from your computer to the web server as part of your form input.

```
<FORM METHOD=POST ENCTYPE="multipart/form-data"
ACTION="test.php">
File to upload: <INPUT TYPE=FILE NAME="upfile"><BR>
<INPUT TYPE=SUBMIT VALUE="Submit">
</FORM>
```

Configuring a form for file uploads requires setting two attributes in the <FORM ...> tags in addition to using<INPUT TYPE=FILE>: POST and "multipart/form-data" (as in the example above). When the data is sent, the original file name (including the full path) of the file as it was on your computer is sent to the web server. The CGI, however, is free to save the file as anything it wants -- or to not save it at all.

For the time being, only use form-based file upload if you know that the users will have Netscape (for example in an intranet or if the form is just for one person that you know has Netscape) or MSIE 4.0 or later.

An often expressed wish with file uploads is to have a way of suggesting the file type being uploaded. Netscape, for example, when it gives you the file upload dialog box, inexplicably assumes you want to upload an HTML file. Unfortunately, there is no way to suggest a file type.

TYPE = IMAGE

IMAGE creates an image that is also a "submit" button. When the user clicks on the image, the form is submitted.

```
<FORM ACTION="../cgi-bin/mycgi.pl">
name: <INPUT NAME="realname">
<INPUT TYPE=IMAGE HEIGHT=110 WIDTH=160
    SRC="../../../Program Files/xampp/htdocs/Skype.png"
ALT="Send It In!" ALIGN=ABSMIDDLE>
</FORM>
```

Validating User Input

The validation of data that has been entered in a form is necessary in most cases. Why is important? For example, what good is holding a contest or sweepstakes if you can't notify the winner, because he or she entered an invalid telephone number or an incorrect address. What good is having a mailing list if the e-mail addresses on it aren't verified, and your mailing list just bounces back to you without reaching the subscribers and target audience.

Validating form entries saves you time and more importantly, it can save you money. And since somebody embossed the slogan "Time is money!", this should be very important for your web site!

Well when should we validate? There are two types of validation; **client side and server side.**

For reference, **client side** means that you are depending on what browser the user is currently using. On the client side, validation is performed using JavaScript. And that can be very tricky, because some users turn off JavaScript support in their browsers before they even come to your site. If you encounter of one those users, client side validation won't help you much if you try to verify data from a form because your JavaScript code will not be executed or interpreted by the browser, means you are back to square 1. Remember, the winner of your competition entered a wrong address.

This is where **server side** validation comes in handy. It will always work, no matter what. Of course assuming that you have access to the technology on your server. Server side validation can be done with Perl, PHP, ASP, ColdFusion, JSP and almost any other scripting language. For this tutorial, I'll use PHP. A quite popular and easy to master server side scripting language.

Now that you know the differences between client side and server side validation, you might ask, "**Why use client side validation at all**?" The reason is, that especially high traffic web sites, should seize the opportunity to take off the load of the server and distribute it to the client browser. This means that if you can verify the content of a field before it is submitted and processed by the server, it makes sense to do so. And there is a user friendly side of it as well. Since most people assume that once they have clicked the submit button on a form, the process is over. A nifty popup explaining what is missing or incorrect, improves their chance of entering correct data into the form. Who wants to miss out on that lottery jackpot just because he or she forgot to verify the data they entered on an online entry form.

Form Validation on the Client Side

Forms validation on the client-side is essential — it saves time and bandwidth, and gives you more options to point out to the user where they've gone wrong in filling out the form. Having said that, I don't mean that you don't need server-side validation. People who visit your site may use an old browser or have JavaScript disabled, which will break client-only validation. Client and server-side validation complement each other, and as such, they really shouldn't be used independently.

WHY IS CLIENT SIDE VALIDATION GOOD?

There are two good reasons to use client-side validation:

1. It's a fast form of validation: if something's wrong, the alarm is triggered upon submission of the form.
2. You can safely display only one error at a time and focus on the wrong field, to help ensure that the user correctly fills in all the details you need.

TWO MAJOR VALIDATION APPROACHES

The two key approaches to client-side form validation are:

- Display the errors one by one, focusing on the offending field
- Display all errors simultaneously, server-side validation style

While displaying all errors simultaneously is required for server-side validation, the better method for validation on the client-side is to show one error at a time. This makes it possible to highlight only the field that has been incorrectly completed, which in turn makes revising and successfully submitting the form much easier for the visitor. If you present users with all errors at the same time, most people will try to remember and correct them at once, instead of attempting to re-submit after each correction.

Given these advantages, we'll focus only on validation methods that display one error at a time.

NonEmpty Validation: Applied to TextBox, TextArea

```
<script type='text/javascript'>
        function notEmpty(elem, helperMsg)
                {
                if(elem.value.length == 0)
                        {
                        alert(helperMsg);
                        elem.focus();
                        return false;
                }
                return true;
                }
```

```
</script>
<form>
        Enter Your Name: <input type='text' id='req1'/><input type="text" name="msg" value="" />
        <input type='button' onclick="notEmpty(document.getElementById('req1'), 'Please Enter a Value')"
        value='Check Field' />
</form>
```

AlphabetOnly: Applied to TextBox, TextArea

```
<script type='text/javascript'>
function isAlphabet(elem, helperMsg){
        var alphaExp = /[a-zA-Z]/;
        if(elem.value.match(alphaExp)){
                return true;
        }else{
                alert(helperMsg);
                elem.focus();
                return false;
        }
}
</script>
<form>
Letters Only: <input type='text' id='letters'/>
<input type='button'
        onclick="isAlphabet(document.getElementById('letters'), 'Letters Only Please')"
        value='Check Field' />
</form>
```

Number Only: Applied to TextBox, TextArea

```
<script type='text/javascript'>
function isNumeric(elem, helperMsg){
        var numericExpression = /^[0-9]+$/;
        if(elem.value.match(numericExpression)){
                return true;
        }else{
                alert(helperMsg);
                elem.focus();
                return false;
        }
}
</script>
<form>
        Numbers Only: <input type='text' id='numbers'/>
        <input type='button' onclick="isNumeric(document.getElementById('numbers'), 'Numbers Only Please')"
        value='Check Field' />
</form>
```

AlphabNumericOnly: Applied to TextBox, TextArea

```
<script type='text/javascript'>
```

```
function isAlphanumeric(elem, helperMsg){
	var alphaExp = /^[0-9a-zA-Z]+$/;
	if(elem.value.match(alphaExp)){
		return true;
	}else{
		alert(helperMsg);
		elem.focus();
		return false;
	}
}</script>
<form>
AlphaNumberic Only: <input type='text' id='address'/>
<input type='button'
	onclick="isAlphanumeric(document.getElementById('address'), 'Aplha-Numeric Characters Only Please')"
	value='Check Field' />
</form>
```

Selection: Applied to ListBox

```
<script type='text/javascript'>
function madeSelection(elem, helperMsg){
	if(elem.value == "Please Choose"){
		alert(helperMsg);
		elem.focus();
		return false;
	}else{
		return true;
	}
}
</script>
<form>
Selection: <select id='selection'>
<option>Please Choose</option>
<option>CA</option>
<option>WI</option>
<option>XX</option>
</select>
<input type='button'
	onclick="madeSelection(document.getElementById('selection'), 'Please Choose Something')"
	value='Check Field' />
</form>
```

LengthRestriction: Applied to TextBox

```
<script type='text/javascript'>
function lengthRestriction(elem, min, max){
	var uInput = elem.value;
	if(uInput.length >= min && uInput.length <= max){
		return true;
	}else{
```

```
                alert("Please enter between " +min+ " and " +max+ " characters");
                elem.focus();
                return false;
        }
}
</script>
<form>
Username(6-8 characters): <input type='text' id='restrict'/>
<input type='button'
        onclick="lengthRestriction(document.getElementById('restrict'), 6, 8)"
        value='Check Field' />
</form>
```

EmailValidation: Applied to TextBox

```
<script type='text/javascript'>
function emailValidator(elem, helperMsg){
        var emailExp = /^[\w\-\.\+]+\@[a-zA-Z0-9\.\-]+\.[a-zA-z0-9]{2,4}$/;
        if(elem.value.match(emailExp)){
                return true;
        }else{
                alert(helperMsg);
                elem.focus();
                return false;
        }
}
</script>
<form>
Email: <input type='text' id='emailer'/>
<input type='button'
        onclick="emailValidator(document.getElementById('emailer'), 'Not a Valid Email')"
        value='Check Field' />
</form>
```

Server side validation with PHP

Server-side data validation means using PHP to verify that good values have been sent to the script. Using server-side validation has pretty much the exact opposite pros and cons of client-side development: it is more secure and works seamlessly with all browsers, but it does so at the cost of slightly higher server load and slower feedback for users.

One big advantage to server-side validation is that you can use PHP - a language by now you have attained some skill with. As you know, PHP has a wide variety of functions and language features to help you chop and change strings, check numbers are within ranges, and so on. Furthermore, you can use PHP to connect to a database to check whether a username exists, for example, which is simply impossible using client-side scripting.

We will verify a field for numbers only (e.g. a zip code), numbers and spaces (e.g. a telephone number), etc. Here's our setup; we have a form.php and a error.php.

form.php

```
<html>
<head> ...</head>
<body>
<form action="error.php" method="post">
        <table>
                <tr><td>Your name:</td><td><input type="text" name="your_name"></td></tr>
                <tr><td>Your phone:</td><td><input type="text" name="your_phone"></td></tr>
                <tr><td>Zip code:</td><td><input type="text" name="your_zip"></td></tr>
        </table> <br>
        <input type="submit">
</form>
</body>
</html>
```

Pretty easy, isn't it? The table is not necessary, but it helps to make the form look nice.

error.php

```
<?php
extract($_POST);
/* Validation */
function check_field1($field_name_1)
        {
          if(!preg_match("/[^a-zA-Z0-9\.\-\Ä\ä\Ö\ö\Ü\ü\  ]+$/s",$field_name_1))
                  return TRUE;
          else
                  return FALSE;
        }
function check_field2($field_name_2)
        {
          if(!preg_match("/[^0-9\ ]+$/",$field_name_2))
                  return TRUE;
          else
                  return FALSE;
        }
function check_field3($field_name_3)
        {
          if(!preg_match("/[^0-9]+$/ ",$field_name_3))
                  return TRUE;
          else
                  return FALSE;
        }
/* Validation */
$error=0; // check up variable
        /* get it checking */
if(!check_field1($your_name))
        {
          echo "Illegal input $your_name in 'your_name'";
          $error++; // $error=$error+1;
        }
if(!check_field2($your_phone))
        {
          echo "Illegal input $your_phone in 'your_phone'";
          $error++;
        }
if(!check_field3($your_zip))
        {
          echo "Illegal input $your_zip in 'your_zip'";
```

```
        $error++;
      }
if($error == 0)
      {
        echo  "  The data you entred was correct, thank you!<p>  Your data:<br>  Your name: $your_name<br>
        Your phone: $your_phone<br>  ZIP code: $your_zip  ";
      }
else  {
        echo "Number of errors: $error";
      }
?>
```

Now the above code explanation is here. First of all, we have three functions to do the error checking.

All three utilize a PHP function called preg_match (http://www.php.net/manual/en/function.preg-match.php). We call the function, tell it what field to check and when the entered data matches the string it looks by it returns true, or false if it doesn't.

If the function returns true it does nothing, if it returns false, it outputs the error message and increments the value of $error by 1.

Now what's that really do? /[^a-zA-Z0-9\.\-\Ä\ä\Ö\ö\Ü\ü\ ]+$/

The slashes "/" and "/" are delimiters, "^" marks the start of string or line and the Dollar sign "$" the end of the string, or line. The plus-symbol "+" means required.

Knowing what the special characters mean, it actually says the following: *A string, from start to finish, may contain this characters (a to z (lower case), A to Z (upper case), the numbers from 0 to 9, a dot (".”), a hyphen ("-”) and the special characters ä, ö ü (both upper and lower case) and space (” ")), and these characters only.*

preg_match() is a case sensitiv function, which means it treats "a" and "A" differently. I included upper ("A-Z") and lower case ("a-z"). So called "special characters" (Special, because they have another meaning in PHP as well. But that's another story.) have to be escaped, which means you write a backslash in front of it. For example: \- (the hiven) or \. (the dot). Other special characters are: "^[$()|*+?{\".

The other two functions are self explanatory, as they check only for numbers, and numbers and space ("\ ").

Hope you have learned the basics of server side scripting. Feel free to use the above code on your web site. Here, we will show you how to validate an email address using PHP. PHP is a server-side technology, which is not dependant on the user like client-side validation is.

First, let's begin with a very simple form where we ask the visitor to supply an email address. A real world example could be a form used to subscribe or unsubsribe from your newsletter and since newsletters are delivered to an email address we would not want to collect anything but a valid email address.

The only real disadvantage to the method I am about to describe is that we will not verify if the email address itself is valid and if it really exists but we will check its formatting, which works well in over 90% of all cases.

To check if an email address really exists, there are ways to query the mailserver – though those do not work in many cases because it also opens the door for spammers – or the more popular method called "double-opt-in", which involves sending an email to the subscriber with a mandatory action – for example, to click a link, or a reply – to confirm subscription or unsubscription from a service. Confirming a subscription is part of the CANSPAM act.

For purposes of this tutorial, newsletters and CANSPAM are not the objective, so let's get started with the form.

```
<html>
<head></head>
<body>
<form action="handler.php" method="post">
        <label for="the_email_id">Email</label>:<br />
        <input type="text" name="the_email" id="the_email_id" size="20" maxlength="60" /><br />
        <input type="submit" name="submit_btn" value="check email address" />
</form>
</body>
</html>
```

The form is fairly self-explanatory, a single text field and a submit button. The script to handle the validation process will be named "handler.php", as the form's "action" suggests.

Here is the script: **handler.php**

```
<?php
        if ($_SERVER['REQUEST_METHOD'] == 'GET')
                {
                  die('No post.');
                }
        $email = (string) $_POST['the_email'];
        if (empty($email))
                {
                  die('You did not enter anything. Please go<a href="javascript:history.back(-
1);">back</a>.');
                }
        if (!eregi("^[_a-z0-9-]+(.[_a-z0-9-]+)*@[a-z0-9-]+          (.[a-z0-9-]+)*(.[a-z]{2,4})$", $email))
                {
                  die('Your email address does not follow the        basic format. Sorry, please try again!');
                }
        echo sprintf('Your email address "%s"
        looks valid. Thank you!', $email);

/*  continue here with further processing, for example subscribe/unsubscribe process */
?>
```

Now let's walk through this piece of code step by step.

At first, we verify that the form was submitted via "post" (remember the HTML?). Why is this important? Well, we do not want people to tinker with our code. Tinkering leads to exploiting, and since we expect the email address to be in PHP's $_POST (which also hints on a required "post" method), this is a good way to start.

If we pass the "post"-check, we continue to check if anything was entered at all. This is not a necessary step as the following step will catch this as well, but a check performed on empty() is also a lot faster than a regular expression. Doing this gives us the possibility to exit early and actually save resources.

(On a sidenote: This is also a preferred measure when you deal with databases and maybe more critical data on other levels. You always want to verify what you got and if you got anything at all and prevent malicous code from entering further layers of your application.)

Last but not least we use a regular expression to test the format of the string/email supplied by the user.

"^[_a-z0-9-]+(.[_a-z0-9-]+) *@[a-z0-9-]+(.[a-z0-9-]+)*(.[a-z]{2,4})$"

A more closer look reveals that we allow characters from a to z, 0 to 9, a underscore, hyphen, and a dot to come in front of the "@"-symbol. Following the "@" we basically allow the same, but force an extension in the end. And the extension on email addresses are supposed to be characters only, with a minimum length of two characters and maxmimum length of (currently) four.

Passing variables between pages

There are two ways the browser client can send information to the web server.

- The GET Method
- The POST Method

The GET Method

Before the browser sends the information, it encodes it using a scheme called URL encoding. In this scheme, name/value pairs are joined with equal signs and different pairs are separated by the ampersand.

```
name1=value1&name2=value2&name3=value3
```

Spaces are removed and replaced with the + character and any other nonalphanumeric characters are replaced with a hexadecimal values. After the information is encoded it is sent to the server.

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the **?** character.

```
http://www.mysite.com/index.htm?name1=value1&name2=value2
```

- The GET method produces a long string that appears in your server logs, in the browser's Location: box.
- The GET method is restricted to send upto 1024 characters only.
- Never use GET method if you have password or other sensitive information to be sent to the server.
- GET can't be used to send binary data, like images or word documents, to the server.
- The data sent by GET method can be accessed using QUERY_STRING environment variable.
- The PHP provides **$_GET** associative array to access all the sent information using GET method.

Try out following example by putting the source code in test.php script.

```
<?php
            if( $_GET["name"] || $_GET["age"] )
          {
              echo "Welcome ". $_GET['name']. "<br />";
              echo "You are ". $_GET['age']. " years old.";
              exit();
            }
?>

<html>
<body>
          <form action="<?php $_PHP_SELF ?>" method="GET">
                      Name: <input type="text" name="name" />
                      Age: <input type="text" name="age" />
                      <input type="submit" />
```

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

The POST Method

The POST method transfers information via HTTP headers. The information is encoded as described in case of GET method and put into a header called QUERY_STRING.

- The POST method does not have any restriction on data size to be sent.
- The POST method can be used to send ASCII as well as binary data.
- The data sent by POST method goes through HTTP header so security depends on HTTP protocol. By using Secure HTTP you can make sure that your information is secure.
- The PHP provides **$_POST** associative array to access all the sent information using GET method.

Try out following example by putting the source code in **test.php** script.

```
<?php
        if( $_POST["name"] || $_POST["age"] )
                {
                    echo "Welcome ". $_POST['name']. "<br />";
                    echo "You are ". $_POST['age']. " years old.";
                    exit();
                }
?>
<html>
        <body>
                 <form action="<?php $_PHP_SELF ?>" method="POST">
                            Name: <input type="text" name="name" />
                            Age: <input type="text" name="age" />
                            <input type="submit" />
                 </form>
        </body>
</html>
```

The $_REQUEST variable

The PHP $_REQUEST variable contains the contents of both $_GET, $_POST, and $_COOKIE. We will discuss $_COOKIE variable when we will explain about cookies.

The PHP $_REQUEST variable can be used to get the result from form data sent with both the GET and POST methods.

Try out following example by putting the source code in test.php script.

```
<?php
         if( $_REQUEST["name"] || $_REQUEST["age"] )
                {
                  echo "Welcome ". $_REQUEST['name']. "<br />";
                  echo "You are ". $_REQUEST['age']. " years old.";
                  exit();
                }
?>
<html>
<body>
```

```
        <form action="<?php $_PHP_SELF ?>" method="POST">
                Name: <input type="text" name="name" />
                Age: <input type="text" name="age" />
                <input type="submit" />
        </form>
</body>
</html>
```

Here $_PHP_SELF variable contains the name of self script in which it is being called.

Chapter – 5

Function in PHP

In this Chapter

In-built function

Array functions, Date & Time functions, File system functions, Math functions, String functions

User Defined Function

Functions in PHP

Like other Programming Languages is PHP is also supporting two type of functions to users.

1) **In-built functions**
2) **User Defined Functions**

1) In-built functions

PHP is very rich in terms of Buil-in functions. Here is the list of various important function categories. There are various other function categories which are not covered here.

- PHP Array Functions
- PHP Calendar Functions
- PHP Class/Object Functions
- PHP Character Functions
- PHP Date & Time Functions
- PHP Directory Functions
- PHP Error Handling Functions
- PHP File System Functions
- PHP Math Functions
- PHP MySQL Functions
- PHP Network Functions
- PHP ODBC Functions
- PHP String Functions
- PHP SimpleXML Functions
- PHP XML Parsing Functions

We will study the required functions here.

1.1) PHP Array Functions

These functions allow you to interact with and manipulate arrays in various ways. Arrays are essential for storing, managing, and operating on sets of variables. PHP supports both simple and multi-dimensional arrays. There are also specific functions for populating arrays from database queries.

a) Count() function

Definition and Usage

Count elements in an array, or properties in an object. If the optional mode parameter is set to COUNT_RECURSIVE (or 1), count() will recursively count the array. This is particularly useful for counting all the elements of a multidimensional array. The default value for mode is 0. count() does not detect infinite recursion.

Syntax

```
count($array, $mode );
```

Paramters

Parameter	Description
Array	Required. Specifies an array
Mode	Optional. Specifies the mode of the function.

Return Value

Returns the number of elements in an array.

Example

```
<?php
$a[0] = 1;
$a[1] = 3;
$a[2] = 5;
$result = count($a);
print($result); ?>
```

This will produce following result:

```
3
```

b) List() function

Definition and Usage

Like array(), this is not really a function, but a language construct. list() is used to assign a list of variables in one operation.

Syntax

```
list ( $var1, $var2, $var3.. )
```

Paramters

Parameter	Description
var1	Required. The first variable to assign a value to
var2	Optional. The second variable to assign a value to
var3	Optional. The third variable to assign a value to

Return Value

This does not return anything.

Example

```
<?php
$my_array = array("mango","apple","banana");
list($a, $b, $c) = $my_array;
echo "I have several fruits, a $a, a $b and a $c.";
?>
```

This will produce following result:

```
I have several fruits, a mango, a apple, a banana
```

c) In_array() function

Definition and Usage

The in_array() function searches an array for a specific value. If the third parameter **strict** is set to TRUE then the in_array() function will also check the types of the $value.

Syntax

```
in_array ( $value, $array [,$strict ] );
```

Paramters

Parameter	Description
Value	Required. Value to be search in array.
Array	Required. Specifies an array
Strict	Optional. If this parameter is set, the in_array() function searches for the search-string and specific type in the array.

Return Value

This function returns TRUE if the value is found in the array, or FALSE otherwise.

Example

```
<?php
$os = array("Mac", "NT", "Irix", "Linux");
if (in_array("Irix", $os)) {
   echo "Got Irix";
}
if (in_array("mac", $os)) {
   echo "Got mac";
}
?>
```

This will produce following result:

```
Got Irix
```

d) Current() function

Definition and Usage

Every array has an internal pointer to its "current" element, which is initialized to the first element inserted into the array.

The current() function simply returns the value of the array element that's currently being pointed to by the internal pointer. It does not move the pointer in any way. If the internal pointer points beyond the end of the elements list, current() returns FALSE.

Syntax

```
current ( $array );
```

Paramters

Parameter	Description
Array	Required. Specifies an array

Return Value

Returns the current element in an array.

Example

```
<?php
$transport = array('foot', 'bike', 'car', 'plane');
$mode = current($transport);
print "$mode <br />";
$mode = next($transport);
print "$mode <br />";
$mode = current($transport)
print "$mode <br />";;
$mode = prev($transport);
print "$mode <br />";
$mode = end($transport);
print "$mode <br />";
$mode = current($transport);
print "$mode <br />";
?>
```

This will produce following result:

```
foot
bike
bike
foot
plane
plane
```

e) Next() function

Definition and Usage

The next() function returns the array value in the next place that's pointed to by the internal array pointer, or FALSE if there are no more elements.

Syntax

```
next ( $array );
```

Paramters

Parameter	Description
Array	Required. Specifies an array

Return Value

Returns the next element in an array.

Example

```
<?php
$transport = array('foot', 'bike', 'car', 'plane');
$mode = current($transport);
print "$mode <br />";
$mode = next($transport);
print "$mode <br />";
$mode = current($transport)
print "$mode <br />";;
$mode = prev($transport);
print "$mode <br />";
$mode = end($transport);
print "$mode <br />";
$mode = current($transport);
print "$mode <br />";
?>
```

This will produce following result:

```
foot
bike
bike
foot
plane
plane
```

f) Prev() function

Definition and Usage

The prev() function returns the array value in the previous place that's pointed to by the internal array pointer, or FALSE if there are no more elements.

Syntax

```
Prev ( $array );
```

Paramters

Parameter	Description
array	Required. Specifies an array

Return Value

Returns the previous element in an array.

Example

```
<?php
$transport = array('foot', 'bike', 'car', 'plane');
```

```
$mode = current($transport);
print "$mode <br />";
$mode = next($transport);
print "$mode <br />";
$mode = current($transport)
print "$mode <br />";;
$mode = prev($transport);
print "$mode <br />";
$mode = end($transport);
print "$mode <br />";
$mode = current($transport);
print "$mode <br />";
?>
```

This will produce following result:

```
Foot
bike
bike
foot
plane
plane
```

g) End() function

Definition and Usage

The end() function advances array's internal pointer to the last element, and returns its value.

Syntax

```
end ( $array );
```

Paramters

Parameter	Description
Array	Required. Specifies an array

Return Value

Returns the last element in an array.

Example

```
<?php
$transport = array('foot', 'bike', 'car', 'plane');
$mode = current($transport);
print "$mode <br />";
$mode = next($transport);
print "$mode <br />";
$mode = current($transport)
print "$mode <br />";;
$mode = prev($transport);
print "$mode <br />";
$mode = end($transport);
print "$mode <br />";
$mode = current($transport);
print "$mode <br />";
?>
```

This will produce following result:

```
foot
bike
bike
foot
plane
plane
```

h) Each() function

Definition and Usage

The each() function returns the current key and value pair from the array array and advances the array cursor. This pair is returned in a four-element array, with the keys 0, 1, key, and value. Elements 0 and key contain the key name of the array element, and 1 and value contain the data

Syntax

```
each ( $array );
```

Paramters

Parameter	Description
Array	Required. Specifies an array

Return Value

Returns the current key and value pair from an array.

Example

```
<?php
$transport = array('foot', 'bike', 'car', 'plane');
$key_value = each($transport);
print_r($key_value);
print "<br />";
$key_value = each($transport);
print_r($key_value);
print "<br />";
$key_value = each($transport);
print_r($key_value);
?>
```

This will produce following result:

```
Array ( [1] => foot [value] => foot [0] => 0 [key] => 0 )
Array ( [1] => bike [value] => bike [0] => 1 [key] => 1 )
Array ( [1] => car [value] => car [0] => 2 [key] => 2 )
```

i) Sort() function

Definition and Usage

This function sorts an array. Elements will be arranged from lowest to highest when this function has completed.

This function assigns new keys for the elements in array. It will remove any existing keys you may have assigned, rather than just reordering the keys.

Syntax

```
sort( $array [, $sort_flags] );
```

Paramters

Parameter	Description
array	Required. Specifies an array.
sort_flags	Optional. Specifies how to sort the array values. Possible values: • SORT_REGULAR - Default. Treat values as they are (don't change types) • SORT_NUMERIC - Treat values numerically • SORT_STRING - Treat values as strings • SORT_LOCALE_STRING - Treat values as strings, based on local settings

Return Value

Returns TRUE on success or FALSE on failure.

Example

```
<?php
$fruits = array("d"=>"lemon", "a"=>"orange", "b"=>"banana" );
sort($fruits);
print_r($fruits);
?>
```

This will produce following result:

```
Array ( [0] => banana [1] => lemon [2] => orange )
```

j) Rsort() function

Syntax

```
rsort( $array [, $sort_flags] );
```

Definition and Usage

This function sorts an array in reverse order (highest to lowest). This function assigns new keys for the elements in array. It will remove any existing keys you may have assigned, rather than just reordering the keys.

Paramters

Parameter	Description
Array	Required. Specifies an array.
sort_flags	Optional. Specifies how to sort the array values. Possible values: • SORT_REGULAR - Default. Treat values as they are (don't change types) • SORT_NUMERIC - Treat values numerically • SORT_STRING - Treat values as strings • SORT_LOCALE_STRING - Treat values as strings, based on local settings

Return Value

Returns TRUE on success or FALSE on failure.

Example

```
<?php
$fruits = array("d"=>"lemon", "a"=>"orange", "b"=>"banana" );
rsort($fruits);
print_r($fruits);
?>
```

This will produce following result:

```
Array ( [0] => orange [1] => lemon [2] => banana )
```

k) Assort() function

Syntax

```
asort( $array [, $sort_flags] );
```

Definition and Usage

This function sorts an array such that array indices maintain their correlation with the array elements they are associated with. This is used mainly when sorting associative arrays where the actual element order is significant.

Paramters

Parameter	Description
Array	Required. Specifies an array.
sort_flags	Optional. Specifies how to sort the array values. Possible values: • SORT_REGULAR - Default. Treat values as they are (don't change types) • SORT_NUMERIC - Treat values numerically • SORT_STRING - Treat values as strings • SORT_LOCALE_STRING - Treat values as strings, based on local settings

Return Value

Returns TRUE on success or FALSE on failure.

Example

```
<?php
$fruits = array("d"=>"lemon", "a"=>"orange", "b"=>"banana" );
asort($fruits);
print_r($fruits);
?>
```

This will produce following result:

```
Array ( [b] => banana [d] => lemon [a] => orange )
```

l) Array_merge() function

Definition and Usage

This function merges the elements of one or more arrays together so that the values of one are appended to the end of the previous one. If the input arrays have the same string keys, then the later value for that key will overwrite the previous one.

Syntax

```
array array_merge ( array $array1 [, array $array2 [, array $array3...]] );
```

Paramters

Parameter	Description
array1	Required.Specifies an array.
array2	Optional.Specifies an array.
array3	Optional.Specifies an array.

Return Values

It returns the resulting array.

Example

```
<?php
$array1=array("a"=>"Horse","b"=>"Cat","c"=>"Dog");
$array2=array("d"=>"Cow","a"=>"Cat","e"=>"elephant");
print_r(array_merge($array1,$array2));
?>
```

This will produce following result:

```
Array ( [a]=>Cat [b]=>Cat [c]=>Dog [d]=>Cow [e]=>elephant )
```

m) Array_reverse() function

Definition and Usage

This function reverses the order of all the elements of a padded array.

Syntax

```
array_reverse ( $array [, $preserve_keys] );
```

Paramters

Parameter	Description
Array	Required. Specifies an array.
preserve_keys	Optional. Specifies if order of keys also has to be changed ofr not. By default its FALSE.

Return Values

Return an array with elements in reverse order.

Example

```
<?php
$array=array("a"=>"banana","b"=>"apple","c"=>"orange");
print_r(array_reverse($array));
?>
```

This will produce following result:

```
Array ( [c] => orange [b] => apple [a] => banana )
```

1.2) PHP DATE / TIME function

These functions allow you to get the date and time from the server where your PHP scripts are running. You can use these functions to format the date and time in many different ways.

a) Date() Function

Definition and Usage

Returns a string formatted according to the given format string using the given integer timestamp or the current time if no timestamp is given. In other words, timestamp is optional and defaults to the value of time().

Syntax

```
string date ( string $format [, int $timestamp] );
```

Paramters

Parameter	Description
format	Required. Specifies how to return the result: • d - The day of the month (from 01 to 31) • D - A textual representation of a day (three letters) • j - The day of the month without leading zeros (1 to 31) • l (lowercase 'L') - A full textual representation of a day • N - The ISO-8601 numeric representation of a day (1 for Monday through 7 for Sunday) • S - The English ordinal suffix for the day of the month (2 characters st, nd, rd or th. Works well with j) • w - A numeric representation of the day (0 for Sunday through 6 for Saturday) • z - The day of the year (from 0 through 365) • W - The ISO-8601 week number of year (weeks starting on Monday) • F - A full textual representation of a month (January through December) • m - A numeric representation of a month (from 01 to 12) • M - A short textual representation of a month (three letters) • n - A numeric representation of a month, without leading zeros (1 to 12) • t - The number of days in the given month • L - Whether it's a leap year (1 if it is a leap year, 0 otherwise) • o - The ISO-8601 year number

	• Y - A four digit representation of a year • y - A two digit representation of a year • a - Lowercase am or pm • A - Uppercase AM or PM • B - Swatch Internet time (000 to 999) • g - 12-hour format of an hour (1 to 12) • G - 24-hour format of an hour (0 to 23) • h - 12-hour format of an hour (01 to 12) • H - 24-hour format of an hour (00 to 23) • i - Minutes with leading zeros (00 to 59) • s - Seconds, with leading zeros (00 to 59) • e - The timezone identifier (Examples: UTC, Atlantic/Azores) • I (capital i) - Whether the date is in daylights savings time (1 if Daylight Savings Time, 0 otherwise) • O - Difference to Greenwich time (GMT) in hours (Example: +0100) • T - Timezone setting of the PHP machine (Examples: EST, MDT) • Z - Timezone offset in seconds. The offset west of UTC is negative, and the offset east of UTC is positive (-43200 to 43200) • c - The ISO-8601 date (e.g. 2004-02-12T15:19:21+00:00) • r - The RFC 2822 formatted date (e.g. Thu, 21 Dec 2000 16:01:07 +0200) • U - The seconds since the Unix Epoch (January 1 1970 00:00:00 GMT)
timestamp	Optional. This is an integer Unix timestamp that defaults to the current local time if a timestamp is not given. In other words, it defaults to the value of time().

Return Value

Returns a formatted date string. If a non-numeric value is used for timestamp, FALSE is returned and an E_WARNING level error is emitted.

Example

```
<?php
// set the default timezone to use. Available since PHP 5.1
date_default_timezone_set('UTC');

// Prints something like: Monday
echo date("l");
echo "\n";

// Prints something like: Monday 15th of August 2011 03:12:46 PM
echo date('l dS \of F Y h:i:s A');
echo "\n";

// Prints: August 15, 2011 is on a Monday
echo "August 15, 2011 is on a " . date("l", mktime(0, 0, 0, 8, 15, 2011));
echo "\n";

// prints something like: 2011-08-15T00:00:00+00:00
echo date(DATE_ATOM, mktime(0, 0, 0, 8, 15, 2011));
?>
```

This will produce following result:

```
Sunday
Monday 15th of August 2011 08:17:47 AM
August 15, 2011 is on a Monday
2000-08-15T00:00:00+00:00
```

b) Getdate() function

Definition and Usage

Returns a string formatted according to the given format string using the given integer timestamp or the current time if no timestamp is given. In other words, timestamp is optional and defaults to the value of time().

Syntax

```
array getdate ( [int $timestamp] );
```

Paramters

Parameter	Description
timestamp	Optional. This is an integer Unix timestamp that defaults to the current local time if a timestamp is not given. In other words, it defaults to the value of time().

Return Value

Returns an associative array of information related to the typestamp. The returning array contains ten elements with relevant information needed when formatting a date string:

- [seconds] - seconds
- [minutes] - minutes
- [hours] - hours
- [mday] - day of the month
- [wday] - day of the week
- [year] - year
- [yday] - day of the year
- [weekday] - name of the weekday
- [month] - name of the month

Example

```
<?php
$today = getdate();
print_r($today);
?>
```

This will produce following result:

```
Array
(
   [seconds] => 40
   [minutes] => 58
   [hours]   => 21
   [mday]    => 17
   [wday]    => 2
   [mon]     => 6
   [year]    => 2003
   [yday]    => 167
   [weekday] => Tuesday
   [month]   => June
   [0]       => 1055901520
)
```

c) Checkdate() function

Definition and Usage

This function checks the validity of the date formed by the arguments. A date is considered valid if each parameter is properly defined.

Syntax

```
checkdate ( $month, $day, $year );
```

Paramters

Parameter	Description
Moth	Required. The month is between 1 and 12 inclusive.
Day	Required. The day is within the allowed number of days for the given month. Leap years are taken into consideration.
Year	Required. The year is between 1 and 32767 inclusive.

Return Value

Returns TRUE if the date given is valid; otherwise returns FALSE.

Example

```
<?php
var_dump(checkdate(12, 31, 2000));
var_dump(checkdate(2, 29, 2001));
?>
```

This will produce following result:

```
bool(true)
bool(false)
```

d) Time() function

Definition and Usage

The time() function returns the current time measured in the number of seconds since the Unix Epoch (January 1 1970 00:00:00 GMT).

Syntax

```
int time ( void );
```

Paramters

Parameter	Description
Void	NA

Return Value

Returns the current time measured in the number of seconds since the Unix Epoch (January 1 1970 00:00:00 GMT).

Example

```
<?php
        // 7 days; 24 hours; 60 mins; 60secs
$nextWeek = time() + (7 * 24 * 60 * 60);
echo 'Now:       '. date('Y-m-d') ."\n";
echo 'Next Week: '. date('Y-m-d', $nextWeek) ."\n";
?>
```

This will produce following result:

```
Now:       2005-03-30
Next Week: 2005-04-06
```

e) Mktime() function

Definition and Usage

This function returns the Unix timestamp corresponding to the arguments given. This timestamp

is a long integer containing the number of seconds between the Unix Epoch (January 1 1970 00:00:00 GMT) and the time specified.

Arguments may be left out in order from right to left; any arguments thus omitted will be set to the current value according to the local date and time.

Syntax

```
mktime(hour,minute,second,month,day,year,is_dst);
```

Paramters

Parameter	Description
Hour	Optional. Specifies the hour
Minute	Optional. Specifies the minute
Second	Optional. Specifies the second
Month	Optional. Specifies the numerical month
Day	Optional. Specifies the day
Year	Optional. Specifies the year.
is_dst	Optional. Parameters always represent a GMT date so is_dst doesn't influence the result.

Return Value

This function returns the Unix timestamp of the arguments given. If the arguments are invalid, the function returns FALSE.

Example

```
<?php
$lastday = mktime(0, 0, 0, 3, 0, 2000);
echo strftime("Last day in Feb 2000 is: %d\n", $lastday);
$lastday = mktime(0, 0, 0, 4, -31, 2000);
echo strftime("Last day in Feb 2000 is: %d", $lastday);
?>
```

This will produce following result:

```
Last day in Feb 2000 is: 29
Last day in Feb 2000 is: 29
```

1.3) PHP FILE Handling function

The filesystem functions are used to access and manipulate the filesystem PHP provides you all the possible functions you may need to manipulate a file.

a) Fopen() function

Definition and Usage

The fopen() function opens a file or URL.

Syntax

```
fopen(filename,mode,include_path,context)
```

Parameters

Parameter	Description
filename	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 beginning of the file) "r+" (Read/Write. Starts at the beginning of the file) "w" (Write only. Opens and clears the contents of file; or creates a new file if it doesn't exist) "w+" (Read/Write. Opens and clears the contents of file; or creates a new file if it doesn't exist) "a" (Write only. Opens and writes to the end of the file or creates a new file if it doesn't exist) "a+" (Read/Write. Preserves file content by writing to the end of the file) "x" (Write only. Creates a new file. Returns FALSE and an error if file already exists) "x+" (Read/Write. Creates a new file. Returns FALSE and an error if file already exists)
Include_path	Optional. Set this parameter to '1' if you want to search for the file in the include_path (in php.ini) as
Context	Optional. Specifies the context of the file handle. Context is a set of options that can modify the behavior of a stream

Return Value

If fopen() fails, it returns FALSE and an error on failure. You can hide the error output by adding an '@' in front of the function name.

Example

```
<?php
	$file = fopen("test.txt","r");
	$file = fopen("/home/test/test.txt","r");
	$file = fopen("/home/test/test.gif","wb");
	$file = fopen("http://www.example.com/","r");
	$file = fopen("ftp://user:password@example.com/test.txt","w");
?>
```

b) Fread() function

Description and Usage

The fread() reads from an open file. The function will stop at the end of the file or when it reaches the specified length, whichever comes first.

Syntax

```
fread(file,length)
```

Parameter

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

Return Value

This function returns the read string, or FALSE on failure.

Example 1

Read 10 bytes from file:

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

Example 2

Read entire file:

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

c) Fwrite() function

Description and Usage

The 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.

Syntax

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

Parameter

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

Return Value

This function returns the number of bytes written, or FALSE on failure.

d) Fclose() function

Definition and Usage

The fclose() function closes an open file.

Syntax

```
fclose(file)
```

Parameter

Parameter	Description
File	Required. Specifies the file to close

Return Value

This function returns TRUE on success or FALSE on failure.

Example

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

e) File_exists() function

Definition and Usage

The file_exists() function checks whether or not a file or directory exists.

Syntax

```
file_exists(path)
```

Parameter

Parameter	Description
Path	Required. Specifies the path to check

Return Value

This function returns TRUE if the file or directory exists, otherwise it returns FALSE.

Example

```
<?php
echo file_exists("test.txt");
?>
```

The output of the code above will be:

```
1
```

f) Is_readable() function

Definition and Usage

The is_readable() function checks whether the specified file is readable.

Syntax

```
is_readable(file)
```

Parameter

Parameter	Description
File	Required. Specifies the file to check

Return Value

This function returns TRUE if the file is readable.

Example

```
<?php
$file = "test.txt";
if(is_readable($file))  {
  echo ("$file is readable");
  }
else  {
  echo ("$file is not readable");
  }
?>
```

The output of the code above could be:

```
test.txt is readable
```

g) Is_writable function

Definition and Usage

The is_writable() function checks whether the specified file is writeable.

Syntax

```
is_writable(file)
```

Parameter

Parameter	Description
File	Required. Specifies the file to check

Return Value

This function returns TRUE if the file is writeable.

Example

```
<?php
$file = "test.txt";
if(is_writable($file))
  {
  echo ("$file is writeable");
  }
else
  {
  echo ("$file is not writeable");
  }
?>
```

The output of the code above could be:

```
test.txt is writeable
```

h) 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.

Syntax

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

Parameter

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

Return Value

This function returns FALSE on failure.

Example 1 Read the first line

```
<?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 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.
```

i) File() function

Definition and Usage

The file() reads a file into an array. Each array element contains a line from the file, with newline still attached.

Syntax

```
file(path,include_path,context)
```

Parameter

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 behavior of a stream. Can be skipped by using NULL.

Return Value

This function returns all the lines of file in array in key index form.

Example

```
<?php
print_r(file("test.txt"));
?>
```

The output of the code above will be:

```
Array
(
[0] => Hello World. Testing testing!
[1] => Another day, another line.
[2] => If the array picks up this line,
[3] => then is it a pickup line?
)
```

j) File_get_contents() function

Definition and Usage

The file_get_contents() reads a file into a string. 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 performance.

Syntax

file_get_contents(path,include_path,context,start,max_length)

Parameter

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 behavior 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. This parameter was added in PHP 5.1

Return Value

This function returns the file contents in string variable.

Example

```
<?php
echo file_get_contents("test.txt");
?>
```

The output of the code above will be:

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

k) File_put_contents() function

Definition and Usage

The file_put_contents() writes a string to a file. This function follows these rules when accessing a file:

1. If FILE_USE_INCLUDE_PATH is set, check the include path for a copy of *filename*
2. Create the file if it does not exist
3. Open the file
4. Lock the file if LOCK_EX is set
5. If FILE_APPEND is set, move to the end of the file. Otherwise, clear the file content
6. Write the data into the file
7. Close the file and release any locks

Syntax

file_put_contents(file,data,mode,context)

Parameter

Parameter	Description
File	Required. Specifies the file to write to. If the file does not exist, this function will create one
Data	Required. The data to write to the file. Can be a string, an array or a data stream
Mode	Optional. Specifies how to open/write to the file. Possible values: • FILE_USE_INCLUDE_PATH • FILE_APPEND • LOCK_EX
context	Optional. Specifies the context of the file handle. Context is a set of options that can modify the behavior of a stream.

Note: Use FILE_APPEND to avoid deleting the existing content of the file.

Return Value

This function returns the number of character written into the file on success, or FALSE on failure.

Example

```
<?php
echo file_put_contents("test.txt","Hello World. Testing!");
?>
```

The output of the code above will be:

```
21
```

1) Ftell() function

Definition and Usage

The ftell() function returns the current position in an open file.

Syntax

ftell(file)

Parameter

Parameter	Description
File	Required. Specifies the open file to check

Return Value

This function returns the current file pointer position, or FALSE on failure.

Example

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

// print current position
echo ftell($file);
```

```
// change current position
fseek($file,"15");
// print current position again
echo "<br />" . ftell($file);

fclose($file);
?>
```

The output of the code above will be:

```
0
15
```

m) Fseek() function

Definition and Usage

The fseek() 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.

Syntax

fseek(file,offset,whence)

Parameter

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 in PHP 4). Possible values: • SEEK_SET - Set position equal to offset. Default • SEEK_CUR - Set position to current location plus offset • SEEK_END - Set position to EOF plus offset (to move to a position before EOF, the offset must be a negative value)

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

Return Value

This function returns 0 on success, or -1 on failure. Seeking past EOF will not generate an error.

Example

```
<?php
$file = fopen("test.txt","r");
// read first line
fgets($file);
// move back to beginning of file
fseek($file,0);
?>
```

n) Rewind() function

Definition and Usage

The rewind() function "rewinds" the position of the file pointer to the beginning of the file.

Syntax

rewind(file)

Parameter

Parameter	Description
File	Required. Specifies the open file

Return Value

This function returns TRUE on success, or FALSE on failure.

Example

```
<?php
$file = fopen("test.txt","r");
//Change position of file pointer
fseek($file,"15");
//Set file pointer to 0
rewind($file);
fclose($file);
?>
```

o) Copy() function

Definition and Usage

The copy() function copies a file.

Syntax

```
copy(file,to_file)
```

Parameter

Parameter	Description
File	Required. Specifies the file to copy
to_file	Required. Specifies the file to copy to

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

Return Value

This function returns TRUE on success and FALSE on failure.

Example

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

The output of the code above will be:

```
1
```

p) Unlink() function

Definition and Usage

The unlink() function deletes a file.

Syntax

```
unlink(filename,context)
```

Parameter

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 behavior of a stream

Return Value

This function returns TRUE on success, or FALSE on failure.

Example

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

q) Rename() function

Definition and Usage

The rename() function renames a file or directory.

Syntax

```
rename(oldname,newname,context)
```

Parameter

Parameter	Description
Oldname	Required. Specifies the file or directory to be renamed
newname	Required. Specifies the new name of the file or directory
Context	Optional. Specifies the context of the file handle. Context is a set of options that can modify the behavior of a stream

Return Value

This function returns TRUE on success, or FALSE on failure.

Example

```
<?php
rename("images","pictures");
?>
```

1.4) PHP MATH function

The math functions can handle values within the range of integer and float types.

a) Abs() function

Definition and Usage

The abs() function returns the absolute value of a number.

Syntax

```
abs(x)
```

Parameter

Parameter	Description
X	Required. A number. If the number is of type float, the return type is also float, otherwise it is integer

Example

```
<?php
echo(abs(4.7) . "<br />");
echo(abs(-4) . "<br />");
echo(abs(8));
?>
```

The output of the code above will be:

```
4.7
4
8
```

b) Ceil() function

Definition and Usage

The ceil() function returns the value of a number rounded UPWARDS to the nearest integer.

Syntax

```
ceil(x)
```

Parameter

Parameter	Description
X	Required. A number

Example

```
<?php
echo(ceil(0.60) . "<br />");
echo(ceil(0.40) . "<br />");
echo(ceil(5) . "<br />");
echo(ceil(-5.1) . "<br />");
echo(ceil(-5.9))
?>
```

The output of the code above will be:

```
1
1
5
-5
-5
```

c) Floor() function

Definition and Usage

The floor() function returns the value of a number rounded DOWNWARDS to the nearest integer.

Syntax

```
floor(x)
```

Parameter

Parameter	Description
X	Required. A number

Example

```
<?php
echo(floor(0.60) . "<br />");
echo(floor(0.40) . "<br />");
echo(floor(5) . "<br />");
echo(floor(-5.1) . "<br />");
echo(floor(-5.9))
?>
```

The output of the code above will be:

```
0
0
5
-6
-6
```

d) Round() function

Definition and Usage

The round() function rounds a number to the nearest integer.

Syntax

```
round(x,prec)
```

Parameter

Parameter	Description
x	Required. The number to be round
prec	Optional. The number of digits after the decimal point

Example

```
<?php
echo(round(0.60) . "<br />");
echo(round(0.50) . "<br />");
echo(round(0.49) . "<br />");
echo(round(-4.40) . "<br />");
echo(round(-4.60))
?>
```

The output of the code above will be:

```
1
1
0
-4
-5
```

e) Fmod() function

Definition and Usage

The fmod() function divides x by y and returns the remainder (modulo) of the division.

Syntax

fmod(x,y)

Parameter

Parameter	Description
X	Required. A number
Y	Required.

Example

```
<?php
$r = fmod(5,2);
echo $r
?>
```

The output of the code above will be:

```
1
```

f) Min() function

Definition and Usage

The min() function returns the number with the lowest value of two specified numbers.

Syntax

min(x,y)

Parameter

Parameter	Description
X	Required. A number
Y	Required. A number

Example

```
<?php
echo(min(5,7) . "<br />");
echo(min(-3,5) . "<br />");
```

```
echo(min(-3,-5) . "<br />");
echo(min(7.25,7.30))
?>
```

The output of the code above will be:

```
5
-3
-5
7.25
```

g) Max() function

Definition and Usage

The max() function returns the number with the highest value of two specified numbers.

Syntax

```
max(x,y)
```

Parameter

Parameter	Description
x	Required. A number
y	Required. A number

Example

```
<?php
echo(max(5,7) . "<br />");
echo(max(-3,5) . "<br />");
echo(max(-3,-5) . "<br />");
echo(max(7.25,7.30))
?>
```

The output of the code above will be:

```
7
5
-3
7.3
```

h) Pow() function

Definition and Usage

The pow() function raises the first argument to the power of the second argument, and returns the result.

Syntax

```
pow(x,y)
```

Parameter

Parameter	Description
X	Required. Specifies the number to be raised
Y	Required. The power to which to raise the number

Example

```
<?php
echo pow(4,2) . "<br />";
echo pow(6,2) . "<br />";
echo pow(-6,2) . "<br />";
echo pow(-6,-2) . "<br />";
echo pow(-6,5.5);
?>
```

The output of the code above will be:

```
16
36
36
0.0277777777778
-1.#IND
```

i) Sqrt() function

Definition and Usage

The sqrt() function returns the square root of a number.

Syntax

```
sqrt(x)
```

Parameter

Parameter	Description
X	Required. A number

Note: The sqrt() function will return -1.#IND if the parameter x is a negative number.

Example

```
<?php
echo(sqrt(0) . "<br />");
echo(sqrt(1) . "<br />");
echo(sqrt(9) . "<br />");
echo(sqrt(0.64) . "<br />");
echo(sqrt(-9))
?>
```

The output of the code above will be:

```
0
1
3
0.8
-1.#IND
```

j) Rand() function

Definition and Usage

The rand() function generates a random integer. If this function is called without parameters, it returns a random integer between 0 and RAND_MAX. If you want a random number between 10 and 100 (inclusive), use rand (10,100).

Syntax

```
rand(min,max)
```

Parameter

Parameter	Description
min,max	Optional. Specifies the range the random number should lie within

Note: On some platforms (such as Windows) RAND_MAX is only 32768. So, if you require a range larger than 32768, you can specify min and max, or use the mt_rand() function instead.

Note: In PHP 4.2.0 and later, there is no need to seed the random generator with srand(). This is done automatically.

Example

```
<?php
echo(rand() . "<br />");
echo(rand() . "<br />");
echo(rand(10,100))
?>
```

The output of the code above could be:

```
17757
3794
97
```

1.5) PHP STRING function

The string functions allow you to manipulate strings.

a) Chr() function

Definition and Usage

The chr() function returns a character from the specified ASCII value.

Syntax

```
chr(ascii)
```

Parameter

Parameter	Description
Ascii	Required. An ASCII value

Note: The x parameter can be specified in decimal, octal, or hex values. Octal values are defined by a leading 0, while hex values are defined by a leading 0x.

Example

```
<?php
echo chr(52)."<br />";
echo chr(052)."<br />";
echo chr(0x52)."<br />";
?>
```

The output of the code above will be:

```
4
*
R
```

b) Ord() function

Definition and Usage

The ord() function returns the ASCII value of the first character of a string.

Syntax

```
ord(string)
```

Parameter

Parameter	Description
String	Required. The string to get an ASCII value from

Example

```
<?php
echo ord("h")."<br />";
echo ord("hello")."<br />";
?>
```

The output of the code above will be:

```
104
104
```

c) Strtolower() function

Definition and Usage

The strtolower() function converts a string to lowercase.

Syntax

```
strtolower(string)
```

Parameter

Parameter	Description
String	Required. Specifies the string to convert

Example

```
<?php
echo strtolower("Hello WORLD.");
?>
```

The output of the code above will be:

```
hello world.
```

d) Strtoupper() function

Definition and Usage

The strtoupper() function converts a string to uppercase.

Syntax

```
strtoupper(string)
```

Parameter

Parameter	Description
String	Required. Specifies the string to convert

Example

```
<?php
echo strtoupper("Hello WORLD!");
?>
```

The output of the code above will be:

```
HELLO WORLD!
```

e) Strlen() function

Definition and Usage

The strlen() function returns the length of a string.

Syntax

```
strlen(string)
```

Parameter

Parameter	Description
String	Required. Specifies the string to check

Example

```
<?php
echo strlen("Hello world!");
?>
```

The output of the code above will be:

```
12
```

f) Ltrim() function

Definition and Usage

The ltrim() function will remove whitespaces or other predefined character from the left side of a string.

Syntax

```
ltrim(string,charlist)
```

Parameter

Parameter	Description
String	Required. Specifies the string to check
Charlist	Optional. Specifies which characters to remove from the string. If omitted, all of the following characters are removed: • "\0" - NULL • "\t" - tab • "\n" - new line • "\x0B" - vertical tab • "\r" - carriage return • " " - ordinary white space

Example 1

```
<html>
<body>
<?php
$str = "    Hello World!";
echo "Without ltrim: " . $str;
echo "<br />";
echo "With ltrim: " . ltrim($str);
?>
<body>
<html>
```

The browser output of the code above will be:

```
Without ltrim: Hello World!
With ltrim: Hello World!
```

If you select "View source" in the browser window, you will see the following HTML:

```
<html>
<body>
Without ltrim:    Hello World!<br />With ltrim: Hello World!
</body>
</html>
```

Example 2

```
<?php
$str = "\r\nHello World!";
echo "Without ltrim: " . $str;
echo "<br />";
echo "With ltrim: " . ltrim($str);
?>
```

The browser output of the code above will be:

```
Without ltrim: Hello World!
With ltrim: Hello World!
```

If you select "View source" in the browser window, you will see the following HTML:

```
<html>
<body>
Without ltrim:
Hello World!<br />With ltrim: Hello World!
</body>
</html>
```

g) Rtrim() function

Definition and Usage

The rtrim() function will remove whitespaces or other predefined character from the right side of a string.

Syntax

```
rtrim(string,charlist)
```

Parameter

Parameter	Description
string	Required. Specifies the string to check
charlist	Optional. Specifies which characters to remove from the string. If omitted, all of the following characters are removed: • "\0" - NULL • "\t" - tab • "\n" - new line • "\x0B" - vertical tab • "\r" - carriage return • " " - ordinary white space

Example 1

```
<html>
<body>
<?php
$str = "Hello World!    ";
echo "Without rtrim: " . $str;
echo "<br />";
echo "With rtrim: " . rtrim($str);
?>
<body>
<html>
```

The browser output of the code above will be:

```
Without rtrim: Hello World!
With rtrim: Hello World!
```

If you select "View source" in the browser window, you will see the following HTML:

```
<html>
<body>
Without rtrim: Hello World!    <br />With rtrim: Hello World!
</body>
</html>
```

Example 2

```
<?php
$str = "Hello World!\r\n";
echo "Without rtrim: " . $str;
echo "<br />";
echo "With rtrim: " . rtrim($str);
?>
```

The browser output of the code above will be:

```
Without rtrim: Hello World!
With rtrim: Hello World!
```

If you select "View source" in the browser window, you will see the following HTML:

```
<html>
```

```
<body>
Without rtrim: Hello World!
<br />With rtrim: Hello World!
</body>
</html>
```

h) Substr() function

Definition and Usage

The substr() function returns a part of a string.

Syntax

```
substr(string,start,length)
```

Parameter

Parameter	Description
String	Required. Specifies the string to return a part of
start	Required. Specifies where to start in the string • A positive number - Start at a specified position in the string • A negative number - Start at a specified position from the end of the string • 0 - Start at the first character in string
length	Optional. Specifies the length of the returned string. Default is to the end of the string. • A positive number - The length to be returned from the start parameter • Negative number - The length to be returned from the end of the string

Note: If start is a negative number and length is less than or equal to start, length becomes 0.

Example 1

```
<?php
echo substr("Hello world!",6);
?>
```

The output of the code above will be:

```
world!
```

Example 2

```
<?php
echo substr("Hello world!",6,5);
?>
```

The output of the code above will be:

```
World
```

i) Strcmp() function

Definition and Usage

The strcmp() function compares two strings. This function returns:

- 0 - if the two strings are equal
- <0 - if string1 is less than string2
- >0 - if string1 is greater than string2

Syntax

```
strcmp(string1,string2)
```

Parameter

Parameter	Description
string1	Required. Specifies the first string to compare
string2	Required. Specifies the second string to compare

Note: The strcmp() function is binary safe and case-sensitive.

Example

```
<?php
echo strcmp("Hello world!","Hello world!");
?>
```

The output of the code above will be:

```
0
```

j) Strcasecmp() function

Definition and Usage

The strcasecmp() function compares two strings. This function returns:

- 0 - if the two strings are equal
- <0 - if string1 is less than string2
- >0 - if string1 is greater than string2

Syntax

```
strcasecmp(string1,string2)
```

Parameter

Parameter	Description
string1	Required. Specifies the first string to compare
string2	Required. Specifies the second string to compare

Example

```
<?php
echo strcasecmp("Hello world!","HELLO WORLD!");
?>
```

The output of the code above will be:

```
0
```

k) Strops() function

Definition and Usage

The strpos() function returns the position of the first occurrence of a string inside another string. If the string is not found, this function returns FALSE.

Syntax

```
strpos(string,find,start)
```

Parameter

Parameter	Description
String	Required. Specifies the string to search
Find	Required. Specifies the string to find
Start	Optional. Specifies where to begin the search

Note: The strpos() function is case-sensitive.

Example

```
<?php
echo strpos("Hello world!","wo");
?>
```

The output of the code above will be:

```
6
```

l) Strrpos() function

Definition and Usage

The strrpos() function finds the position of the last occurrence of a string inside another string. This function returns the position on success, otherwise it returns FALSE.

Syntax

```
strrpos(string,find,start)
```

Parameter

Parameter	Description
String	Required. Specifies the string to search
Find	Required. Specifies the string to find
Start	Optional. Specifies where to begin the search

Note: The strrpos() function is case-sensitive.

Example

```
<?php
echo strrpos("Hello world!","wo");
?>
```

The output of the code above will be:

```
6
```

m) Strstr() function

Definition and Usage

The strstr() function searches for the first occurrence of a string inside another string. This function returns the rest of the string (from the matching point), or FALSE, if the string to search for is not found.

Syntax

```
strstr(string,search)
```

Parameter

Parameter	Description
String	Required. Specifies the string to search
Search	Required. Specifies the string to search for. If this parameter is a number, it will search for the character matching the ASCII value of the number

Note: This function is case-sensitive. For a case-insensitive search, use stristr().

Example 1

```
<?php
echo strstr("Hello world!","world");
?>
```

The output of the code above will be:

```
world!
```

Example 2

```
<?php
echo strstr("Hello world!",111);
?>
```

The output of the code above will be:

```
o world!
```

n) Stristr() function

Definition and Usage

The stristr() function searches for the first occurrence of a string inside another string. This function returns the rest of the string (from the matching point), or FALSE, if the string to search for is not found.

Syntax

```
stristr(string,search)
```

Parameter

Parameter	Description
String	Required. Specifies the string to search
Search	Required. Specifies the string to search for. If this parameter is a number, it will search for the character matching the ASCII value of the number

Note: This function is case-insensitive. For a case-sensitive search, use strstr().

Example 1

```
<?php
echo stristr("Hello world!","WORLD");
?>
```

The output of the code above will be:

```
world!
```

Example 2

```
<?php
echo stristr("Hello world!",111);
?>
```

The output of the code above will be:

```
o world!
```

o) Str_replace() function

Definition and Usage

The str_replace() function replaces some characters with some other characters in a string. This function works by the following rules:

- If the string to be searched is an array, it returns an array
- If the string to be searched is an array, find and replace is performed with every array element
- If both find and replace are arrays, and replace has fewer elements than find, an empty string will be used as replace
- If find is an array and replace is a string, the replace string will be used for every find value

Syntax

```
str_replace(find,replace,string,count)
```

Parameter

Parameter	Description
find	Required. Specifies the value to find
replace	Required. Specifies the value to replace the value in *find*
string	Required. Specifies the string to be searched
count	Optional. A variable that counts the number of replacements

Note: This function is case-sensitive. Use str_ireplace() to perform a case-insensitive search.

Example 1

```
<?php
echo str_replace("world","Peter","Hello world!");
?>
```

The output of the code above will be:

```
Hello Peter!
```

Example 2

```
<?php
$arr = array("blue","red","green","yellow");
print_r(str_replace("red","pink",$arr,$i));
echo "Replacements: $i";
?>
```

The output of the code above will be:

```
Array
(
[0] => blue
[1] => pink
[2] => green
[3] => yellow
)
Replacements: 1
```

Example 3

```
<?php
$find = array("Hello","world");
$replace = array("B");
$arr = array("Hello","world","!");
print_r(str_replace($find,$replace,$arr));
?>
```

The output of the code above will be:

```
Array
(
[0] => B
[1] =>
[2] => !
)
```

p) Strrev() function

Definition and Usage

The strrev() function reverses a string.

Syntax

```
strrev(string)
```

Parameter

Parameter	Description
String	Required. Specifies the string to reverse

Example

```
<?php
echo strrev("Hello World!");
?>
```

The output of the code above will be:

```
!dlroW olleH
```

q) Echo() function

Definition and Usage

The echo() function outputs one or more strings.

Syntax

```
echo(strings)
```

Parameter

Parameter	Description
Strings	Required. One or more strings to be sent to the output

Note: The echo() function is not actually a function, so you are not required to use parentheses with it. However, if you want to pass more than one parameter to echo(), using parentheses will generate a parse error.

Tip: The echo() function is slightly faster than print().

Tip: The echo() function has the following shortcut syntax. See example 5.

Example 1

```
<?php
$str = "Who's Klenn?";
echo $str;
echo "<br />";
echo $str."<br />I don't know!";
?>
```

The output of the code above will be:

```
Who's Klenn?
Who's Klenn?
I don't know!
```

Example 2

```
<?php
echo "This text
spans multiple
lines.";
?>
```

The output of the code above will be:

```
This text spans multiple lines.
```

Example 3

```
<?php
echo 'This ','string ','was ','made ','with multiple parameters';
?>
```

The output of the code above will be:

```
This string was made with multiple parameters
```

Example 4

Difference of single and double quotes. Single quotes will print the variable name, not the value:

```
<?php
$color = "red";
echo "Roses are $color";
echo "<br />";
echo 'Roses are $color';
?>
```

The output of the code above will be:

```
Roses are red
Roses are $color
```

Example 5

Shortcut syntax:

```
<html>
<body>
<?php
$color = "red";
?>
```

```
<p>Roses are <?=$color?></p>
</body>
</html>
```

r) Print() function

Definition and Usage

The print() function outputs one or more strings.

Syntax

```
print(strings)
```

Parameter

Parameter	Description
Strings	Required. One or more strings to be sent to the output

Note: The print() function is not actually a function, so you are not required to use parentheses with it.
Tip: The print() function is slightly slower than echo().

Example 1

```
<?php
$str = "Who's Klenn?";
print $str;
print "<br />";
print $str."<br />I don't know!";
?>
```

The output of the code above will be:

```
Who's Klenn?
Who's Klenn?
I don't know!
```

Example 2

```
<?php
print "This text
spans multiple
lines.";
?>
```

The output of the code above will be:

```
This text spans multiple lines.
```

Example 3

Difference of single and double quotes. Single quotes will print the variable name, not the value:

```
<?php
$color = "red";
print "Roses are $color";
print "<br />";
print 'Roses are $color';
?>
```

The output of the code above will be:

```
Roses are red
Roses are $color
```

1.6) PHP Miscellaneous functions

These functions are used in PHP for performing several different task which are left in other function sets.

a) Define() function

Definition and Usage

The define() function defines a constant.

Constants are much like variables, except for the following differences:

- A constant's value cannot be changed after it is set
- Constant names do not need a leading dollar sign ($)
- Constants can be accessed regardless of scope
- Constant values can only be strings and numbers

Syntax

define(name,value,case_insensitive)

Parameter

Parameter	Description
Name	Required. Specifies the name of the constant
Value	Required. Specifies the value of the constant
case_insensitive	Optional. Specifies whether the constant name should be case-insensitive. If set to TRUE, the constant will be case-insensitive. Default is FALSE (case-sensitive)

Example 1

Define a case-sensitive constant:

```
<?php
define("GREETING","Hello you! How are you today?");
echo constant("GREETING");
?>
```

The output of the code above will be:

```
Hello you! How are you today?
```

Example 2

Define a case-insensitive constant:

```
<?php
define("GREETING","Hello you! How are you today?",TRUE);
echo constant("greeting");
?>
```

The output of the code above will be:

```
Hello you! How are you today?
```

b) Constant() function

Definition and Usage

The constant() function returns the value of a constant.

Syntax

constant(constant)

Parameter

Parameter	Description
Constant	Required. Specifies the name of the constant to check

Example

```
<?php
//define a constant
define("GREETING","Hello you! How are you today?");
echo constant("GREETING");
?>
```

The output of the code above will be:

```
Hello you! How are you today?
```

Server Side Includes (SSI)

You can insert the content of one PHP file into another PHP file before the server executes it, with the include() or require() function.

The two functions are identical in every way, except how they handle errors:

- include() generates a warning, but the script will continue execution
- require() generates a fatal error, and the script will stop

These two functions are used to create functions, headers, footers, or elements that will be reused on multiple pages.

Server side includes saves a lot of work. This means that you can create a standard header, footer, or menu file for all your web pages. When the header needs to be updated, you can only update the include file, or when you add a new page to your site, you can simply change the menu file (instead of updating the links on all your web pages).

c) Include() function

The include() function takes all the content in a specified file and includes it in the current file. If an error occurs, the include() function generates a warning, but the script will continue execution.

Example 1

Assume that you have a standard header file, called "header.php". To include the header file in a page, use the include() function:

```
<html>
<body>
<?php include("header.php"); ?>
<h1>Welcome to my home page!</h1>
<p>Some text.</p>
</body>
</html>
```

Example 2

Assume you have a standard menu file, called "menu.php", that should be used on all pages:

```
<a href="/default.php">Home</a>
<a href="/tutorials.php">Tutorials</a>
<a href="/references.php">References</a>
<a href="/examples.php">Examples</a>
<a href="/about.php">About Us</a>
<a href="/contact.php">Contact Us</a>
```

All pages in the Web site should include this menu file. Here is how it can be done:

```
<html>
<body>
<div class="leftmenu">
<?php include("menu.php"); ?>
</div>
<h1>Welcome to my home page.</h1>
<p>Some text.</p>
</body>
</html>
```

If you look at the source code of the page above (in a browser), it will look like this:

```
<html>
<body>
<div class="leftmenu">
<a href="/default.php">Home</a>
<a href="/tutorials.php">Tutorials</a>
<a href="/references.php">References</a>
<a href="/examples.php">Examples</a>
<a href="/about.php">About Us</a>
```

```
<a href="/contact.php">Contact Us</a>
</div>
<h1>Welcome to my home page!</h1>
<p>Some text.</p>
</body>
</html>
```

d) Require() function

The require() function is identical to include(), except that it handles errors differently. If an error occurs, the include() function generates a warning, but the script will continue execution. The require() generates a fatal error, and the script will stop.

Error Example include() Function

```
<html>
<body>
<?php
include("wrongFile.php");
echo "Hello World!";
?>
</body>
</html>
```

Error message:

```
Warning: include(wrongFile.php) [function.include]:
failed to open stream:
No such file or directory in C:\home\website\test.php on line 5
Warning: include() [function.include]:
Failed opening 'wrongFile.php' for inclusion
(include_path='.;C:\php5\pear')
in C:\home\website\test.php on line 5
Hello World!
```

Notice that the echo statement is executed! This is because a Warning does not stop the script execution.

Error Example require() Function

```
<html>
<body>
<?php
require("wrongFile.php");
echo "Hello World!";
?>
</body>
</html>
```

Error message:

```
Warning: require(wrongFile.php) [function.require]:
failed to open stream:
No such file or directory in C:\home\website\test.php on line 5
Fatal error: require() [function.require]:
Failed opening required 'wrongFile.php'
(include_path='.;C:\php5\pear')
in C:\home\website\test.php on line 5
```

The echo statement is not executed, because the script execution stopped after the fatal error.

It is recommended to use the require() function instead of include(), because scripts should not continue after an error.

e) Header() function

Definition and Usage

The header() function sends a raw HTTP header to a client. It is important to notice that header() must be called before any actual output is sent (In PHP 4 and later, you can use output buffering to solve this problem):

```
<html>
<?php
// This results in an error.
// The output above is before the header() call
header('Location: http://www.example.com/');
?>
```

Syntax

header(string,replace,http_response_code)

Parameter

Parameter	Description
String	Required. Specifies the header string to send
Replace	Optional. Indicates whether the header should replace previous or add a second header. Default is TRUE (will replace). FALSE (allows multiple headers of the same type)
http_response_code	Optional. Forces the HTTP response code to the specified value (available in PHP 4.3 and higher)

Note: Since PHP 4.4 this function prevents more than one header to be sent at once. This is a protection against header injection attacks.

Example 1

Prevent page caching:

```
<?php
// Date in the past
header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
header("Cache-Control: no-cache");
header("Pragma: no-cache");
?>
<html>
<body>
...
...
```

Note: There are options that users may set to change the browser's default caching settings. By sending the headers above, you should override any of those settings and force the browser to not cache!

Example 2

Let the user be prompted to save a generated PDF file (Content-Disposition header is used to supply a recommended filename and force the browser to display the save dialog box):

```
<?php
header("Content-type:application/pdf");
// It will be called downloaded.pdf
header("Content-Disposition:attachment;filename='downloaded.pdf'");
// The PDF source is in original.pdf
readfile("original.pdf");
?>
<html>
<body>
...
...
```

f) Die() function

Definition and Usage

The die() function prints a message and exits the current script. This function is an alias of the exit() function.

Syntax

die(message)

Parameter

Parameter	Description
Message	Required. Specifies the message or status number to write before exiting the script. The status number will not be written to the output.

Example

```
<?php
$site = "http://www.w3schools.com/";
fopen($site,"r")
or die("Unable to connect to $site");
?>
```

2) User Defined Function

PHP functions are similar to other programming languages. A function is a piece of code which takes one more input in the form of parameter and does some processing and returns a value.

You already have seen many functions like **fopen()** and **fread()** etc. They are built-in functions but PHP gives you option to create your own functions as well.

There are two parts which should be clear to you:

- Creating a PHP Function
- Calling a PHP Function

In fact you hardly need to create your own PHP function because there are already more than 1000 of built-in library functions created for different area and you just need to call them according to your requirement.

Creating PHP Function:

Its very easy to create your own PHP function. Suppose you want to create a PHP function which will simply write a simple message on your browser when you will call it. Following example creates a function called writeMessage() and then calls it just after creating it.

Note that while creating a function its name should start with keyword **function** and all the PHP code should be put inside { and } braces as shown in the following example below:

```
<html>
<head>
<title>Writing PHP Function</title>
</head>
<body>
<?php
/* Defining a PHP Function */
function writeMessage()
{
  echo "You are really a nice person, Have a nice time!";
}
/* Calling a PHP Function */
writeMessage();
?>
</body>
</html>
```

This will display following result:

```
You are really a nice person, Have a nice time!
```

PHP Functions with Paramters:

PHP gives you option to pass your parameters inside a function. You can pass as many as parameters your like. These parameters work like variables inside your function. Following example takes two integer parameters and add them together and then print them.

```
<html>
<head>
<title>Writing PHP Function with Parameters</title>
</head>
<body>
<?php
function addFunction($num1, $num2)
{
  $sum = $num1 + $num2;
  echo "Sum of the two numbers is : $sum";
}
addFunction(10, 20);
?>
</body>
</html>
```

This will display following result:

```
Sum of the two numbers is : 30
```

Passing Arguments by Reference:

It is possible to pass arguments to functions by reference. This means that a reference to the variable is manipulated by the function rather than a copy of the variable's value.

Any changes made to an argument in these cases will change the value of the original variable. You can pass an argument by reference by adding an ampersand to the variable name in either the function call or the function definition.

Following example depicts both the cases.

```
<html>
<head>
<title>Passing Argument by Reference</title>
</head>
<body>
<?php
function addFive($num)
{
   $num += 5;
}
function addSix(&$num)
{
   $num += 6;
}
$orignum = 10;
```

```
addFive( &$orignum );
echo "Original Value is $orignum<br />";
addSix( $orignum );
echo "Original Value is $orignum<br />";
?>
</body>
</html>
```

This will display following result:

```
Original Value is 15
Original Value is 21
```

PHP Functions returning value:

A function can return a value using the **return** statement in conjunction with a value or object. Return stops the execution of the function and sends the value back to the calling code.

You can return more than one value from a function using **return array(1,2,3,4)**.

Following example takes two integer parameters and add them together and then returns their sum to the calling program. Note that **return** keyword is used to return a value from a function.

```
<html>
<head>
<title>Writing PHP Function which returns value</title>
</head>
<body>

<?php
function addFunction($num1, $num2)
{
  $sum = $num1 + $num2;
  return $sum;
}
$return_value = addFunction(10, 20);
echo "Returned value from the function : $return_value
?>
</body>
</html>
```

This will display following result:

```
Returned value from the function : 30
```

Setting Default Values for Function Parameters:

You can set a parameter to have a default value if the function's caller doesn't pass it.

Following function prints NULL in case use does not pass any value to this function.

```
<html>
<head>
<title>Writing PHP Function which returns value</title>
</head>
<body>
<?php
function printMe($param = NULL)
{
```

```
   print $param;
}
printMe("This is test");
printMe();
?>
</body>
</html>
```

This will produce following result:

```
This is test
```

Dynamic Function Calls:

It is possible to assign function names as strings to variables and then treat these variables exactly as you would the function name itself. Follwoing example depicts this behaviour.

```
<html>
<head>
<title>Dynamic Function Calls</title>
</head>
<body>

<?php
function sayHello()
{
   echo "Hello<br />";
}
$function_holder = "sayHello";
$function_holder();
?>
</body>
</html>
```

This will display following result:

```
Hello
```

Other Functions

Addslashes() function

Definition and Usage

The addslashes() function returns a string with backslashes in front of predefined characters.
The predefined characters are:

- single quote (')
- double quote (")
- backslash (\)
- NULL

Syntax

addslashes(string)

Parameter	Description
String	Required. Specifies the string to check

Tip: This function can be used to prepare a string for storage in a database and database queries.
Note: PHP runs addslashes() on all GET, POST, and COOKIE data by default. Therefore you should not use addslashes() on strings that have already been escaped, this will cause double escaping. The function get_magic_quotes_gpc() can be used to check this.

Example

In this example we will add backslashes to the predefined characters in a string:

```
<?php
$str = "Who's Kai Jim?";
echo $str . " This is not safe in a database query.<br />";
echo addslashes($str) . " This is safe in a database query.";
?>
```

The output of the code above will be:

```
Who's Kai Jim? This is not safe in a database query.
Who\'s Kai Jim? This is safe in a database query.
```

PHP include_once()

Definition and Usage

The include_once() statement can be used to include a php file in another one, when you may need to include the called file more than once. If it is found that the file has already been included, calling script is going to ignore further inclusions.

If a.php is a php script calling b.php with include_once() statement, and does not find b.php, a.php executes with a warning, excluding the part of the code written within b.php.

Syntax

```
include_once('name of the called file with path');
```

Example

```
<?php
echo "today is:".date("Y-m-d");
?>
```

The above file is x.php

The above file x.php, is included twice with include_once() statement in the following file y.php. But from the output you will get that the second instance of inclusion is ignored, since include_once() statement ignores all the the similar inclusions after the first one.

```
<?php
include_once('x.php');
include_once('x.php');
?>
```

PHP require_once()

Definition and Usage

require_once() statement can be used to include a php file in another one, when you may need to include the called file more than once. If it is found that the file has already been included, calling script is going to ignore further inclusions.

If a.php is a php script calling b.php with require_once() statement, and does not find b.php, a.php stops executes causing a fatal error.

Syntax

```
require_once('name of the calling file with path');
```

Example

```
<?php
echo "today is:".date("Y-m-d");
?>
```

The above file is x.php

The above file x.php, is included twice with require_once() statement in the following file y.php. But from the output you will get that the second instance of inclusion is ignored, since require_once() statement ignores all the similar inclusions after the first one.

```
<?php
require_once('x.php');
require_once('x.php');
?>
```

Exercise

1. How many types of function available with programming language?
2. Explain any 5 math functions.
3. Explain any 5 char and string functions.
4. Explain any 5 date functions.
5. Explain any 5 file functions.
6. Explain any 5 miscellaneous functions.
7. Is date function supports time stamp?\
8. Explain difference between include () and require ()?
9. What is functionality of header?
10. Justify: Cannot modify header information - headers already sent by ...
11. Explain addslashes function.
12. What is the functionality of Include_once(), require_once()?
13. How do I get a variable in a function to retain its value between calls?
14. Difference between echo and print function?
15. How do I generate a random number from php?

Chapter – 6

Session and Cookie

In this Chapter

Why to use sessions?

PHP sessions

Starting Sessions

Session Variables

Creating, modifying, unregistering

isset() Function

Destroying session

What is Cookie?

Securities with Cookie

Cookies in PHP

Creating and retrieving

Why to Use Sessions?

As HTTP is stateless protocol, it becomes very difficult to maintain the state [like when you login, the site remembers you until you logout]. So there are two ways to do it. Either store cookies at client side which stores the user info and other state related info, which the PHP will use. Session serves the same purpose but it better than cookies as user can disable cookies. You can get lot of info related to this on the internet.

A normal HTML website will not pass data from one page to another. In other words, all information is forgotten when a new page is loaded. This makes it quite a problem for tasks like a shopping cart, which requires data (the user's selected product) to be remembered from one page to the next.

As a website becomes more sophisticated, so must the code that backs it. When you get to a stage where your website needs to pass along user data from one page to another, it might be time to start thinking about using PHP sessions.

PHP Sessions

A PHP session solves this problem by allowing you to store user information on the server for later use (i.e. username, shopping cart items, etc). However, this session information is temporary and is usually deleted very quickly after the user has left the website that uses sessions.
It is important to ponder if the sessions' temporary storage is applicable to your website. If you require a more permanent storage you will need to find another solution, like a MySQL database.

Sessions work by creating a unique identification (UID) number for each visitor and storing variables based on this ID. This helps to prevent two users' data from getting confused with one another when visiting the same webpage.

Note:If you are not experienced with session programming it is not recommended that you use sessions on a website that requires high-security, as there are security holes that take some advanced techniques to plug.

PHP Sessions allow web pages to be treated as a group, allowing variables to be shared between different pages. PHP sessions also work when the user has disabled the browser's cookie support. In this situation it includes the session ID information in the web page URLs.

PHP Session Variables

When you are working with an application, you open it, do some changes and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are and what you do because the HTTP address doesn't maintain state.

A PHP session solves this problem by allowing you to store user information on the server for later use (i.e. username, shopping items, etc). However, session information is temporary and will be deleted after the user has left the website. If you need a permanent storage you may want to store the data in a database.

Sessions work by creating a unique id (UID) for each visitor and store variables based on this UID. The UID is either stored in a cookie or is propagated in the URL.

Starting a Session

Before you can store user information in your PHP session, you must first start up the session. When you start a session, it must be at the very beginning of your code, before any HTML or text is sent.

Below is a simple script that you should place at the beginning of your PHP code to start up a PHP session.

Note: The session_start() function must appear BEFORE the <html> tag:

```
<?PHP
        session_start();
?>
        <html>
                <body>……..</body>
        </html>
```

This tiny piece of code will register the user's session with the server, allow you to start saving user information and assign a UID (unique identification number) for that user's session.

Storing a Session Variable

When you want to store user data in a session use the $_SESSION associative array. This is where you both store and retrieve session data. In previous versions of PHP there were other ways to perform this store operation, but it has been updated and this is the correct way to do it.

```
<?php
        session_start();
        $_session[‘views’]=1;        //creating and storing session variable
?>
<html>
        <body>
                <?php
                        echo “Page Views:- ”.$_SESSION[‘views’];        //retrieving session variable
                ?>
        </body>
</html>
```

Output

```
Page Views: 1
```

Modifying Session Variable

For modification of the session variable the user just require to replace the old value of the session variable with new one which can be done by assigning the new value only.

```
<?php
        session_start();
        $_session['views']=4;        //modifying session variable
?>
<html>
        <body>
                <?php
                        echo "Page Views:- ".$_SESSION['views'];        //retrieving session variable
                ?>
        </body>
</html>
```

Output

```
Page Views: 4
```

PHP – isset () function

Now that you are able to store and retrieve data from the $_SESSION array, we can explore some of the real functionality of sessions. When you create a variable and store it in a session, you probably want to use it in the future. However, before you use a session variable it is necessary that you check to see if it exists already!

This is where PHP's *isset* function comes in handy. *isset* is a function that takes any variable you want to use and checks to see if it has been **set**. That is, it has already been assigned a value.

With our previous example, we can create a very simple pageview counter by using *isset* to check if the pageview variable has already been created. If it has we can increment our counter. If it doesn't exist we can create a pageview counter and set it to one. Here is the code to get this job done:

```
<?php
        session_start();
        if (isset($_SESSION['views']))
                $_SESSION['views'] = $_SESSION['views']+1;
        else
                $_SESSION['views'] = 1;
?>
<html>
        <body>
                <?php
                        echo "Page Views: ".$_SESSION['views'] ;
                ?>
        </body>
</html>
```

Output

```
Page Views: 5
```

The first time you run this script on a **freshly opened browser** the *if statement* will fail because no session variable *views* would have been stored yet. However, if you were to refresh the page the *if statement* would be true and the counter would increment by one. Each time you reran this script you would see an increase in *view* by one.

Unregistering and Deleting Session Variables

Although a session's data is temporary and does not require that you explicitly clean after yourself, you may wish to delete some data for your various tasks.

Imagine that you were running an online business and a user used your website to buy your goods. The user has just completed a transaction on your website and you now want to remove everything from their shopping cart.

The unset() function is used to free the specified session variable:

```
<?php
        unset($_SESSION['views']);
?>
```

You can also completely destroy the session by calling the session_destroy() function:

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

Note: session_destroy() will reset your session and you will lose all your stored session data.

What is Cookie?

A "cookie" is a small piece of information sent by a web server to store on a web browser so it can later be read back from that browser. This is useful for having the browser remember some specific information.

Why is Cookie used?

An example is when a browser stores your passwords and user ID's. They are also used to store preferences of start pages, both Microsoft and Netscape use cookies to create personal start pages. Common cookies which companies use are find info are listed below:

Online ordering system

An online ordering system could be developed using cookies that would remember what a person

wants to buy, this way if a person spends three hours ordering CDs at your site and suddenly has to get off the net they could quit the browser and return weeks or even years later and still have those items in their shopping basket.

Site Personalization

This is one of the most beneficial uses, let's say a person comes to the MSNBC site but doesn't want to see any sports news. They allow people to select this as an option, from then on (until the cookie expires) they wouldn't see sports news. This is also usefull for start pages.

Website Tracking

A lot of people think it is an invasion of privacy, if a web site designer wanted to see what interests them. Site tracking can show you "Dead End Paths", places in your website that people go to and then wander off because they don't have any more interesting links to hit. It can also give you more accurate counts of how many people have been to pages on your site. You could differentiate 50 unique people seeing your site from one person hitting the reload button 50 times.

Targeted Marketing

This is probably one of the main uses of cookies, they can be used to build up a profile of where you go what adverts you click on, this information is then used to target adverts at you, which they think are of interest, companies also use cookies to store which adverts have been displayed so the same advert does not get displayed twice. Doubleclick's use of cookies.

How do Cookie worked?

A command line in the HTML of a document tells the browser to set a cookie of a certain name or value. Here is an example of some script used to set a cookie. Set-Cookie: NAME=VALUE; expir es=DATE; path=PATH; domain=DOMAIN_NAME; secure Cookies are usually run from CGI scripts, but they can also be set or read by Javascript.

Security with Cookies

An HTTP Cookie cannot be used to get data from your hard drive, get your email address or steal sensitive information about your person. Early implementations of Java and JavaScript could allow people to do this but for the most part these security leaks have been plugged. But HTTP Cookie can be used to track where you travel over a particular site, This site tracking can be easily done without using cookies as well, using cookies just makes the tracking data a little more consistent. If you want to disallow cookies you can do so with version 3.0 or greater of Netscape. Go to the Options Menu Select the Network Preferences Menu Item From the window that appears Select Protocols Locate the Section Show an Alert Before Check the box labeled Accepting a Cookie From now on you will get an Alert box telling you that a server is trying to set a cookie at your browser. It will tell you what the cookie value is and how long it will last before it is deleted.

The WWW is built on a very simple, but powerful premise. All material on the Web is formatted in a general, uniform format called HTML (Hypertext Markup Language), and all information requests and responses conform to a similarly standard protocol. When someone accesses a server on the Web, such as the Library of Congress, the user's Web browser will send an information request to the

Library of Congress' computer. This computer is called a Web server. The Web server will respond to the request by transmitting the desired information to the user's computer. There, the user's browser will display the received information on the user's screen.

Cookies are pieces of information generated by a web server or web hosting sites and stored in the user's computer, ready for future access. Cookies are embedded in the HTML information flowing back and forth between the user's computer and the servers. Cookies were implemented to allow user-side customization of Web information. For example, cookies are used to personalize Web search engines, to allow users to participate in WWW-wide contests (but only once!), and to store shopping lists of items a user has selected while browsing through a virtual shopping mall.

Essentially, cookies make use of user-specific information transmitted by the Web server onto the user's computer so that the information might be available for later access by itself or other servers. In most cases, not only does the storage of personal information into a cookie go unnoticed, so does access to it. Web servers automatically gain access to relevant cookies whenever the user establishes a connection to them, usually in the form of Web requests.

Cookies are based on a two-stage process. First the cookie is stored in the user's computer without their consent or knowledge. For example, with customizable Web search engines like My Yahoo!, a user selects categories of interest from the Web page. The Web server then creates a specific cookie, which is essentially a tagged string of text containing the user's preferences, and it transmits this cookie to the user's computer. The user's Web browser, if cookie-savvy, receives the cookie and stores it in a special file called a cookie list. This happens without any notification or user consent. As a result, personal information (in this case the user's category preferences) is formatted by the Web server, transmitted, and saved by the user's computer.

During the second stage, the cookie is clandestinely and automatically transferred from the user's machine to a Web server. Whenever a user directs her Web browser to display a certain Web page from the server, the browser will, without the user's knowledge, transmit the cookie containing personal information to the Web server.

Cookie in PHP (Creating Cookies)

The setcookie() function is used to set a cookie.

Note: The setcookie() function must appear BEFORE the <html> tag.

When you create a cookie, using the function *setcookie*, you must specify three arguments. These arguments are ***setcookie(name, value, expire)***;

1. **name**: The name of your cookie. You will use this name to later retrieve your cookie, so don't forget it!
2. **value**: The value that is stored in your cookie. Common values are username (string) and last visit(date).
3. **expire**: The date when the cookie will expire and be deleted. If you do not set this expiration date, then it will be treated as a session cookie and be removed when the browser is restarted.

In the following example we will be creating a cookie that stores the user's last visit to measure how often people return to visit our webpage. We want to ignore people that take longer than two months to return to the site, so we will set the cookie's expiration date to two months in

the future!

```
<?php
        // Calculate 60 days in the future
        // Seconds*minutes*hours*days + current time
        $intwomonths = 60*60*24*60 + time();
        Setcookie(‘lastVisit’, date(“G:i –m/d/y”), $intwomonths);
?>
```

Don't worry if you can't follow the somewhat involved date calculations in this example. The important part is that you know how to set a cookie, by specifying the three important arguments: name, value and expiration date.

In the example below, we will create a cookie named "user" and assign the value "Alex Porter" to it. We also specify that the cookie should expire after one hour:

```
<?php
        $expire = time()+60*60*24*30;
        Setcookie(“user”,”Viral”,$expire);
?>
```

Retrieving Cookies in PHP

If your cookie hasn't expired yet, let's retrieve it from the user's PC using the aptly named *$_COOKIE* associative array. The name of your stored cookie is the key and will let you retrieve your stored cookie value!

```
<?php
        if (isset($_COOKIE[‘lastvisit’]))
                $visit = $_COOKIE[‘lastvisit’];
        else
                echo “You have got some stale cookies!”;
        echo “Your Last Visit was -”.$visit;
?
```

Exercise

1. What is SESSION?
2. When Session expire?
3. What is the difference between session_register and $_session?
4. What is cookie?
5. When cookie expire?
6. What Is a Persistent Cookie?
7. How to Unregistering and Deleting Session Variables?
8. Is cookie better than session?
9. Where does the PHP session stored, either client side or server side?

Chapter – 7

How to Upload Files?

In this Chapter

Uploading File in PHP

Input form, Action Page

Uploading Multiple Files

PHP Upload File

A very useful aspect of PHP is its ability to manage file uploads to your server. Allowing users to upload a file to your server opens a whole can of worms, so please be careful when enabling file uploads.

To allow users to upload a file to the server, you first need to provide a form for them to specify which file they want to upload. Once they click the submit button of the form, the action page is called. This is the page that needs to contain the PHP code to process the uploaded file. See the following HTML Form for a more in-depth look at forms.

The Input Form

Before a user can upload a file, you need to provide them with an interface that allows them to select a file and initiate the upload.

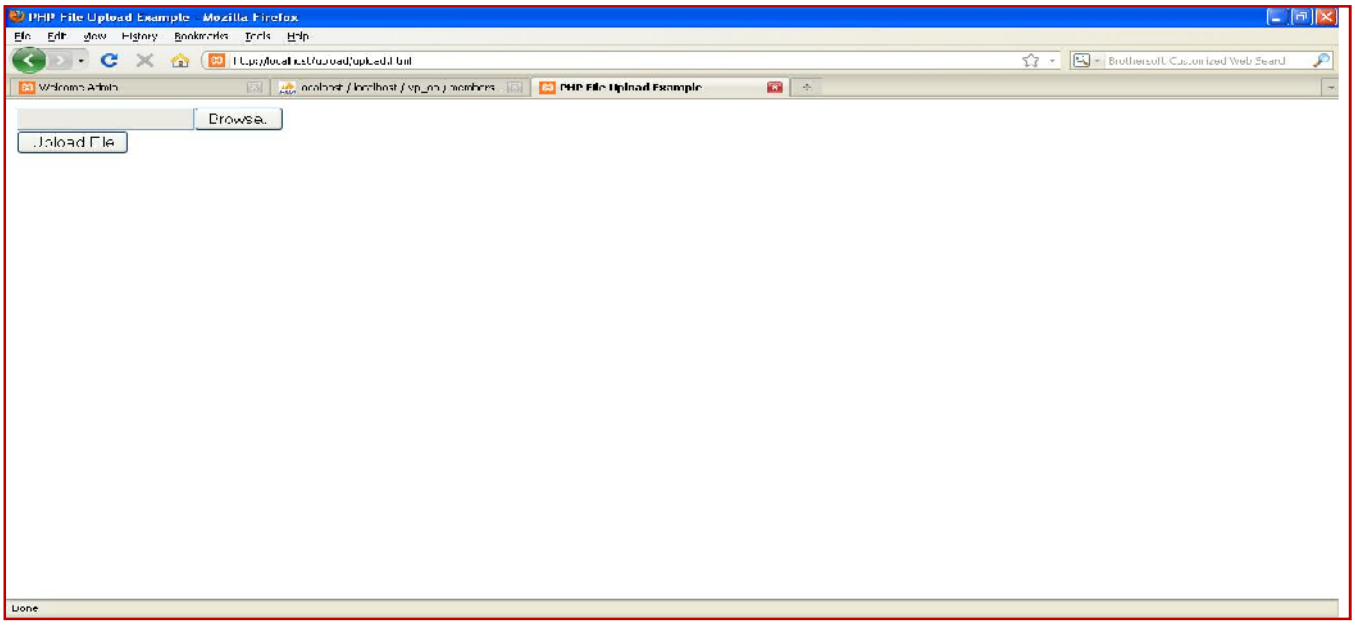
upload.html

```
<html>
        <head>
                <title>PHP File Upload Example</title>
        </head>
        <body>
        <form enctype="multipart/form-data" method="post" action="uploadFile.php">
                <input type="file" name="fileToUpload" /><br />
                <input type="submit" value="Upload File" />
        </form>
        </body>
</html>
```

Here is a brief description of the important parts of the above code:

- **enctype="multipart/form-data"** - Necessary for our to-be-created PHP file to function properly.
- **action="uploadFile.php"** - The name of our PHP page that will be created, shortly.
- **method="POST"** - Informs the browser that we want to send information to the server using POST method.
- **<input type=”file” name="fileToUpload" />** - is used for displaying the file upload control on the web browser. It will provide one textbox with the browse button besides of it which enable the user the browse the client PC for uploading the file.
- **<input type="submit" value="Upload File" />** - is the submit button which is required for submitting the form after user browses the file to be upload on the server.

Save this form code into a file and call it *upload.html*. If you view it in a browser it should look like this:



The Action Page

Once the user uploads a file, the file is uploaded into a temporary directory on the server. If you don't move the file it will disappear. Therefore, your action page needs to move the file to another location where it can stay as long as you want it to.

Whenever a file is uploaded, you can find out certain information about the file including its name, type, size, as well as the name of the temporary file on the server. These details are made available to you via a PHP array called *$_FILES*.

Displaying Details of the Uploaded File

This code simply displays the details of the uploaded file. It doesn't move the file to another location - we'll get to that next. For now, you can use this code in conjunction with the above input form to demonstrate what happens when you upload a file to the server.

Notice the PHP *$_FILES* array which contains info about the file. Note that we also divide the file size by 1024 in order to convert it into kb.

(Ignore any carriage returns in this example - each table row should be on one line).

uploadFile.php

```
<?php
        if ($_FILES["fileToUpload"]["error"] > 0)
                {
                echo "Error: " . $_FILES["fileToUpload"]["error"] . "<br />";
                }
        else
```

```
        {
        echo "<table border=\"0\">";

        echo "<tr><td>Client Filename: </td> <td>". FILES["fileToUpload"]["name"].
"</td></tr>";

        echo "<tr><td>File Type: </td><td>" . $_FILES["fileToUpload"]["type"].
"</td></tr>";

        echo "<tr><td>File Size: </td><td>" . ($_FILES["fileToUpload"]["size"] /
1024) . " Kb</td></tr>";

        echo "<tr><td>Name of Temporary File: </td><td>".
$_FILES["fileToUpload"]["tmp_name"] . "</td></tr>";

        echo "</table>";

        }

        ?>
```

The above code results in something like this:

Client Filename:	Water lilies.jpg
File Type:	image/jpeg
File Size:	81.830078125 Kb
Name of Temporary File:	C:\WINDOWS\TEMP\php48B2.tmp

Description : uploadFile.php

When the *uploadFile.php* file is executed, the uploaded file exists in a temporary storage area on the server. If the file is not moved to a different location it will be **destroyed**! To save our precious file we are going to need to make use of the *$_FILES* associative array.

The *$_FILES* array is where PHP stores all the information about files. There are two elements of this array that we will need to understand for this example.

- **fileToUpload** - is the reference we assigned in our HTML form. We will need this to tell the $_FILES array which file we want to play around with.
- **$_FILES['fileToUpload']['error]** – contains errors which occurs in case of file to be upload is not uploaded properly.
- **$_FILES['fileToUpload']['name']** - *name* contains the original path of the user uploaded file.
- **$_FILES['fileToUpload']['type']** – *type* contains the type of contents stored in file.
- **$_FILES['fileToUpload']['size']** - *size* contains the size of the user uploaded file in bytes.
- **$_FILES['fileToUpload ']['tmp_name']** - *tmp_name* contains the path to the temporary file that resides on the server. The file should exist on the server in a temporary directory with a temporary name.

Now we can finally start to write a basic PHP upload manager script! Here is how we would get the temporary file name, choose a permanent name, and choose a place to store the file.

Moving the Temp File

As mentioned, if we want to keep the file on the server, we need to move it to another location (of our choice). The following code demonstrates how to move the file from the temporary location.

move_uploaded_file($_FILES["fileToUpload"]["tmp_name"],"C:/upload/".$_FILES["fileToUpload"]["name"]);

Checking for Errors

The $_FILES array includes an item for any errors that may result from the upload. This contains an error code. If there are no errors, the value is zero (0).

You check this value within an "If" statement. If the value is greater than zero, you know an error has occurred and you can present a user friendly message to the user. Otherwise you can processing the file.

```
<?php

if ($_FILES["fileToUpload"]["error"] > 0)
        {
        echo "Apologies, an error has occurred.";
        echo "Error Code: " . $_FILES["fileToUpload"]["error"];
        }
else
        {
        move_uploaded_file($_FILES["fileToUpload"]["tmp_name"],"C:/upload/".$_FILES["fileToUpload"]["name"]);
        }
?>
```

Alternative code for move the uploaded file to different location:

```
<?php
        $target_path = "uploads/";
        $target_path = $target_path.basename($_FILES['fileToUpload']['name']);

        if(move_uploaded_file($_FILES[''fileToUpload]['tmp_name'], $target_path))
                {
                echo "The file ".  basename( $_FILES[''fileToUpload]['name'])." has been uploaded";
                }
        else
                {
                echo "There was an error uploading the file, please try again!";
                }
?>
```

If the upload is successful, then you will see the text "The file *filename* has been uploaded". This is because *move_uploaded_file* returns *true* if the file was moved, and *false* if it had a problem.

If there was a problem then the error message "There was an error uploading the file, please try again!" would be displayed.

Restricting File Type/Size

Letting your users upload files to your server can be very risky. If you're not careful, you could get users uploading all sorts of files - perhaps including harmful executables etc. You could also find one day that you've run out of disk space because some users have been uploading enormous files.

You can restrict the file types and file sizes by using an "if" statement. If the file type and size are acceptable, processing can continue, otherwise, display a message to the user.

Important Note: This doesn't prevent the temp file from being created. The file needs uploaded to the server before PHP can find out the file size and type. This simply prevents the file from being moved to your "permanent" location - hence the file should disappear and (hopefully) not become a problem. In any case, I recommend that you install good anti-virus software before allowing users to upload files to your server.

```
if (($_FILES["fileToUpload"]["type"] == "image/gif") ||
($_FILES["fileToUpload"]["type"] == "image/jpeg") ||
($_FILES["fileToUpload"]["type"] == "image/png" ) &&
($_FILES["fileToUpload"]["size"] < 10000))
        {
        move_uploaded_file($_FILES["fileToUpload"]["tmp_name"],"C:/upload/".$_FILES["fileToU
pload"]["name"]);
        }
else
        {
        echo "Files must be either JPEG, GIF, or PNG and less than 10,000 kb";
        }
```

PHP - File Upload: Safe Practices!

Note: This script is for education purposes only. We do not recommend placing this on a web page viewable to the public.

These few lines of code we have given you will allow anyone to upload data to your server. Because of this, we recommend that you do not have such a simple file uploader available to the general public. Otherwise, you might find that your server is filled with junk or that your server's security has been compromised.

We hope you enjoyed learning about how to work with uploading files with PHP. In the near future we will be adding an advanced lesson that will include more security and additional features!

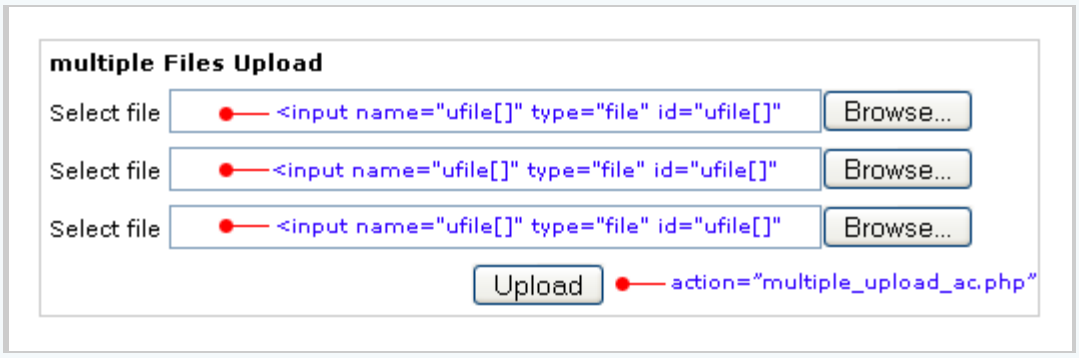
Note: For IE to recognize jpg files the type must be pjpeg, for FireFox it must be jpeg.

Display Multiple Upload boxes on the same page

Step
1. Create file multiple_upload.php
2. Create file multiple_upload_ac.php
3. Create folder "upload" for store uploaded files.
4. CHMOD your upload folder to "777" by using your ftp software(change permission).

1 Create file multiple_upload.php

View in browser



################ Code

```
<table width="500" border="0" align="center" cellpadding="0" cellspacing="1"
bgcolor="#CCCCCC">
<tr>
<form action="multiple_upload_ac.php" method="post" enctype="multipart/form-data"
name="form1" id="form1">
<td>
<table width="100%" border="0" cellpadding="3" cellspacing="1" bgcolor="#FFFFFF">
<tr>
<td><strong>multiple Files Upload </strong></td>
</tr>
<tr>
<td>Select file
<input name="ufile[]" type="file" id="ufile[]" size="50" /></td>
</tr>
<tr>
<td>Select file
<input name="ufile[]" type="file" id="ufile[]" size="50" /></td>
</tr>
<tr>
<td>Select file
<input name="ufile[]" type="file" id="ufile[]" size="50" /></td>
</tr>
<tr>
<td align="center"><input type="submit" name="Submit" value="Upload" /></td>
```

```
</tr>
</table>
</td>
</form>
</tr>
</table>
```

Exercise

1. What is $_FILES?
2. How to move temporary file?
3. How to restrict file type or size?
4. How to upload multiple files?
5. Explain in detail how to upload file?
6. How To Write the FORM Tag Correctly for Uploading Files?

Chapter – 8

Introduction to MySQL

In this Chapter

Installing MySQL

Managing MySQL Database

MySQL table types

MySQL Data types

Queries in MySQL

Integration of PHP and MySQL

Installation of MySQL on Windows

The installation of MySQL is break down into **3 steps**. They are:

Step 1: Getting the software

Download the latest version of MySQL from the http://dev.mysql.com/downloads/ . Unpack it into your desired folder. It is always a good practice to download the **zipped archive** of MySQL binary distribution since using the one click installer, you might sometimes lose the power of configuration manually.

Its also a better way to understand configuring manually since if you switch to a Unix based environment later on, you will have to carry out a similar procedure and there is usually no one click installer in such environments.

I unpacked MySQL in ***C:\Program Files\mysql-5.0.45-win32***

Step 2 : Configuring the environment

While configuring in the Windows environment you have to do two things. One, putting a file having a name **my.ini** in the Windows folder of the operating system. Two, setting the path in your system environment variable. Here's what you have to do:

Creating Option File (my.ini)

Go to your installation path of MySQL (In my case it is: *C:\Program Files\mysql-5.0.45-win32*) and rename the file **my-medium.ini** to **my.ini**. Open the renamed file in notepad and add the following lines, just beneath the line which says, **# The MySQL server** and above the line **port = 3306**:

```
[mysqld]
# set basedir to your installation path
basedir=C:/Program Files/mysql-5.0.45-win32
# set datadir to the location of your data directory
datadir=D:/workspace/data
```

The second line as shown above is, configuring your database directory. This means, the path which you assign to **datadir** will be the path where your databases will be created. You have to make sure that the path exists before you assign it. There will a data directory in the MySQL installed path. If you provide a **different path** to the data directory than this default one, you **must copy** the **entire contents** of the data directory in that path.

Finally, drop the my.ini file into your Windows directory. In my case, I copied the file into the path **C:/Windows**

Setting the environment variable

After completing the above step, goto

My Computer > Properties > Advanced >Environment Variables > System variables > Path > Edit

In the **Value** box, goto the end of the line and put a **semi colon** and the path of the MySQL **bin** folder. In my case I have put it as:

```
;C:\Program Files\mysql-5.0.45-win32\bin
```

Click **OK** 3 times and that's it with the configuration of MySQL. You are all set to test the installation.

Step 3: Final Configurations and Running MySQL

You are actually good to go for running mysql. For this, open up your command prompt and type in the command **mysqld –console.** This will start the mysql database server. Whenever you run this command henceforth you will see these lines at the end:

```
[Note] mysqld: ready for connections.
Version: '5.0.45-community-nt-log' socket: '' port: 3306 MySQL Community Edition (GPL)
```

When you see these lines, it means that mysql has started successfully.

Leave the command prompt open and start up another command prompt. For the first time you should do this:

```
C:\>mysql -u root -p
Enter password:
```

When it will ask for a password, just hit Enter. It will allow access to you without a password and will show up a prompt as:

```
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 12
Server version: 5.0.45-community-nt-log MySQL Community Edition (GPL)
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql>
```

Note that your database has **anonymous** access. It is not a secure thing to let it like that. You should always access the database with a password. So for setting up the password you will have to do the below steps. Put in the commands which are highlighted as shown:

```
mysql> update user set password=PASSWORD("admin") where User='root';
Query OK, 0 rows affected (0.00 sec)
Rows matched: 3 Changed: 0 Warnings: 0
mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)
mysql> quit
Bye
```

Once you have done this, try logging into the server again without a password. It must show the below error:

```
C:\Documents and Settings\xpuser>mysql -u root -p
Enter password:
ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: NO)
```

This means, now your database is secure.You should be relieved that your mysql setup has got completed. Now every time you want to start in your database start the server in one command prompt and in the second command prompt login to the database server by doing:

```
C:\>mysql -u root -p
Enter password: admin
```

Managing Databases in MySQL

Let's start creating a new database in MySQL.

Creating Database

To create a database in MySQL, you use the CREATE DATABASE statement as follows:

```
CREATE DATABASE [IF NOT EXISTS] database_name;
```

CREATE DATABASE statement will create the database with the given name you specified. IF NOT EXISTS is an optional part of the statement. The IF NOT EXISTS part prevents you from error if there is a database with the given name exists in the database catalog.

For example, to create *classicmodels* database, you just need to execute the CREATE DATABASE statement above as follows:

```
CREATE DATABASE classicmodels;
```

After executing the statement, the MySQL will returns you a message to indicate whether the data created successfully or not.

Showing Databases Statement

SHOW DATABASE statement shows all databases in your database server. You can use SHOW DATABASE statement to check the database you've created or to see all the databases' name on the database server before you create a new database, for example:

```
SHOW DATABASES;
```

In my database server, the output is :

```
+----------------------+
|  Database            |
```

```
+----------------------+
| information_schema |
| classicmodels          |
| mysql                   |
+----------------------+
8 rows in set (0.00 sec)
```

Selecting database to work with

To select a database which you plan to work with, you can use USE statement as follows:

```
USE database_name;
```

You can select our sample database by using the USE statement as follows:

```
USE classicmodels;
```

From now you can query the tables' data and operate data inside the selected database.

Removing Database

Removing database means you delete the database physically. All the data and related objects inside the database are permanently deleted and cannot be undone. So it is very important to execute this query with cares. MySQL provides a standard SQL statement DROP DATABASE to allow you to delete a database:

```
DROP DATABASE [IF EXISTS] database_name;
```

Like CREATE DATABASE statement, IF EXIST part is an optional part to prevent you from removing database which is not existed. In order to practice with DROP DATABASE statement, it is recommended to create a temporary database, show the database on the database server, and drop it. The sequence of SQL queries is as follows:

```
CREATE DATABASE IF NOT EXISTS temp_database;
SHOW DATABASES;
DROP DATABASE IF EXISTS temp_database;
```

MySQL Table Types

MySQL supports various types of tables or storage engines to allow you to optimize your database. The table types are available in MySQL are:

- ISAM
- MyISAM
- InnoDB
- BerkeleyDB (BDB)
- MERGE
- HEAP

The most important feature to make all the table types above distinction is transaction-safe or not. Only InnoDB and BDB tables are transaction safe and only MyISAM tables support full-text indexing and searching feature. MyISAM is also the default table type when you create table without declaring which storage engine to use. Here are some major features of each table types:

ISAM

ISAM had been deprecated and removed from version 5.x. All of it functionality entire replace by MyISAM. ISAM table has a hard size 4GB and is not portable.

MyISAM

MyISAM table type is default when you create table. MyISAM table work very fast but not transaction-safe. The size of MyISAM table depends on the operating system and the data file is portable from system to system. With MyISAM table type, you can have 64 keys per table and maximum key length of 1024 bytes.

InnoDB

Different from MyISAM table type, InnoDB table are transaction safe and supports row-level locking. Foreign keys are supported in InnoDB tables. The data file of InnoDB table can be stored in more than one file so the size of table depends on the disk space. Like the MyISAM table type, data file of InnoDB is portable from system to system. The disadvantage of InnoDB in comparison with MyISAM is it take more disk space.

BDB

BDB is similar to InnoDB in transaction safe. It supports page level locking but data file are not portable.

MERGE

Merge table type is added to treat multiple MyISAM tables as a single table so it remove the size limitation from MyISAM tables.

HEAP

Heap table is stored in memory so it is the fastest one. Because of storage mechanism, the data will be lost when the power failure and sometime it can cause the server run out of memory. Heap tables do not support columns with AUTO_INCREMENT, BLOB and TEXT characteristics.

MySQL Data Types

Database table contains multiple columns with specific data types such as numeric or string. MySQL provides you many more specific data types than just "numeric" or "string". Each data type in MySQL can be determined by the following characteristics:

- What kind of value it can represent.
- The space values take up and whether the values are fixed-length or variable-length.
- The values of a data type can be indexed.
- How MySQL compare values of that data types.

Numeric Data Types

You can find all SQL standard numeric types in MySQL including exact number data type and approximate numeric data types including integer, fixed-point and floating point. In addtion, MySQL also supports BIT data type for storing bit field values. Numeric types can be signed or unsigned except BIT type. The following table shows you the summary of numeric types in MySQL:

Numeric Types	Description
TINYINT	A very small integer
SMALLINT	A small integer
MEDIUMINT	A medium-sized integer
INT	A standard integer
BIGINT	A large integer
DECIMAL	A fixed-point number
FLOAT	A single-precision floating-point number
DOUBLE	A double-precision floating-point number
BIT	A bit field

String Data Types

In MySQL, string can hold anything from plain text to binary data such as images and files. String can be compared and searched based on pattern matching by using LIKE clause or regular expression. The table below shows you the string data types in MySQL:

String Types	Description
CHAR	A fixed-length non-binary (character) string
VARCHAR	A variable-length non-binary string
BINARY	A fixed-length binary string
VARBINARY	A variable-length binary string
TINYBLOB	A very small BLOB (binary large object)
BLOB	A small BLOB
MEDIUMBLOB	A medium-sized BLOB
LONGBLOB	A large BLOB

String Types	Description
TINYTEXT	A very small non-binary string
TEXT	A small non-binary string
MEDIUMTEXT	A medium-sized non-binary string
LONGTEXT	A large non-binary string
ENUM	An enumeration; each column value may be assigned one enumeration member
SET	A set; each column value may be assigned zero or more set members

Date and Time Data Types

MySQL provides types for date and time and combination of date and time. In addition, MySQL also provide timestamp data type for tracking last change on a record. If you just want to store the year without date and month, you can use YEAR data type. Here is the table which showing MySQL date and type data types:

Date and Time Types	Description
DATE	A date value in 'CCYY-MM-DD' format
TIME	A time value in 'hh:mm:ss' format
DATETIME	A date and time value in 'CCYY-MM-DD hh:mm:ss' format
TIMESTAMP	A timestamp value in 'CCYY-MM-DD hh:mm:ss' format
YEAR	A year value in CCYY or YY format

Spatial Data Types

MySQL support many spatial data types as below table which contains various kind of geometrical and geographical values.

Spatial Data Types	Description
GEOMETRY	A spatial value of any type
POINT	A point (a pair of X Y coordinates)
LINESTRING	A curve (one or more POINT values)
POLYGON	A polygon
GEOMETRYCOLLECTION	A collection of GEOMETRY values
MULTILINESTRING	A collection of LINESTRING values
MULTIPOINT	A collection of POINT values
MULTIPOLYGON	A collection of POLYGON values

Understanding MySQL TIMESTAMP

*The **MySQL TIMESTAMP** is a temporal data type that store combination of date and time values.*

How MySQL TIMESTAMP stored in the database

The format of MySQL TIMESTAMP column is 'YYYY-MM-DD HH:MM:SS' which is fixed at 19 characters. The MySQL TIMESTAMP are stored as four bytes number of second since the first day of UNIX time which is '1970-01-01 00:00:01'. The upper end of range correspond to maximum four bytes value of UNIX time which is '2038-01-19 03:14:07'. Therefore the TIMESTAMP column has a range of values from '1970-01-01 00:00:01' to '2038-01-19 03:14:07'.

The values of the MySQL TIMESTAMP columns depend on connection's time zone. When insert values for MySQL TIMESTAMP columns, they are converted to Universal Coordinated Time (UTC) from connection's time zone. When you select the value, the server converts it back from UTC to the connection's time zone so you have the same value that you inserted. However if another client with different time zone connects to the server to select value from MySQL TIMESTAMP column, it will see the value adjusted to its time zone. MySQL allows you to change your own time zone when you connect to it so you can see this effect by using a single connection.

Let's try in our sample database to see this effect.

```
CREATE TABLE test_timestamp('t1'  TIMESTAMP);
SET time_zone='+00:00';
INSERT INTO test_timestamp VALUES('2008-01-01 00:00:01');
SELECT t1
FROM test_timestamp;
```

Output

```
+---------------------+
| t1                  |
+---------------------+
| 2008-01-01 07:00:01 |
+---------------------+
1 row in set (0.00 sec)
```

The SQL commands can be explained as follows:

- First we created a table called test_TIMESTAMP.
- Next we set our time zone to UTC.
- Then we insert a TIMESTAMP value into the table test_timestamp.
- Finally we select it to see the value we inserted.

Now can set our time zone to a different time zone and see what value we get from database server:

```
SET time_zone ='+03:00';
SELECT t1
```

```
FROM test_timestamp;
```

Output:

```
+---------------------+
| t1                  |
+---------------------+
| 2008-01-01 03:00:01 |
+---------------------+
1 row in set (0.00 sec)
```

As you see, we get adjusted value to our new time zone.

INSERT and UPDATE TIMESTAMP column

If you omit the MySQL TIMESTAMP column's value in the INSERT statement or you set it to NULL, it will be automatically set to current TIMESTAMP. This characteristic of MySQL TIMESTAMP is known as automatic initialization.

In the table which has a MySQL TIMESTAMP column, if you change other columns' value, the MySQL TIMESTAMP column will be automatic updated to the current TIMESTAMP. The change only accepted if you change its current value to different values in order to have TIMESTAMP column updated. This characteristic of TIMESTAMP called automatic update. Note that only one column can be designated as a TIMESTAMP column which has automatic update.

The MySQL TIMESTAMP columns are design in a table that you need to save the created date and last change date for the records. By applying automatic initialization and automatic update of MySQL TIMESTAMP column you can design the table as follows:

```
CREATE TABLE tbl_name(        …..)
created_on  TIMESTAMP DEFAULT 0
changed_on TIMESTAMP DEFAULT CURRENT_TIMESTAMP
;
```

So when you insert a new record, just omit two TIMESTAMP columns or set them to NULL. The both *created_on* and *changed_on* column will be inserted as the current TIMESTAMP.

When you update, omit both TIMESTAMP columns, the *changed_on* column's value will be automatic updated if there is any change in other columns' values.

In this tutorial, you've learned how MySQLTIMESTAMP data stored in the MySQL database table and how to use its characteristics to design the *created on* and *last changed on* columns.

Query in MySQL

Database table or table in short is one of the most important objects of relational database. How to create databases tables correctly is crucial when you work with any database system especially MySQL.

Creating Tables

To create table we use the **CREATE TABLE** statement. The typical form of SQL CREATE TABLE statement is as follows:

```
CREATE TABLE [IF NOT EXISTS] table_name
(                Column list                )
type= table_type;
```

- MySQL supports IF NOT EXISTS after CREATE TABLE statement to prevent you from error of creating table which already exists on the database server.
- *table_name* is the name of table you would like to create. After that, you can define a set of columns which is usually in this form: column_name data_type(size) [NOT] NULL.
- You can specify the storage engine type you prefer to use for the table. MySQL supports various storage engines such as InnoDB, MyISAM... If you don't explicit declare storage engine type, MySQL will use MyISAM by default.

In our **classicmodels** sample database, to create ***employees*** table, we can use the CREATE TABLE statement as follows:

```
CREATE TABLE employees (
                        lastName varchar(50) NOT NULL,
                        firstName varchar(50) NOT NULL,
                        extension varchar(10) NOT NULL,
                        email varchar(100) NOT NULL,
                        officeCode varchar(10) NOT NULL,
                        reportsTo int(11) default NULL,
                        jobTitle varchar(50) NOT NULL,
                    PRIMARY KEY  (employeeNumber));
```

First, you specify table name *employees* after CREATE TABLE statement. Then you list the columns of the table with their attributes such as data type, size, NOT NULL. And finally you specify the primary key of the table, in this case the primary key is *employeeNumber*.

If the table has more than one primary key, you can seperate them by a comma. For example, the *payments* table has two primary keys *customerNumber* and *checkNumber*. You can create ***payments*** table by executing following query:

```
CREATE TABLE payments (
        customerNumber int(11) NOT NULL,
        checkNumber varchar(50) NOT NULL,
        paymentDate datetime NOT NULL,
        amount double NOT NULL,
        PRIMARY KEY  (customerNumber,checkNumber) );
```

Note that by default MySQL uses MyISAM storage engine for the table it created.

Showing and Describing Tables in a Database

In order to show all tables in a database, you use SHOW TABLES statment. By executing the SHOW TABLES statement, MySQL will returns all tables' name of the current selected database you're working with.

```
SHOW TABLES;
```

Here is the output of ***classicmodels*** database:

```
+-------------------------+
| Tables_in_classicmodels |
+-------------------------+
| customers               |
| employees               |
| offices                 |
| orderdetails            |
| orders                  |
| payments                |
| productlines            |
| products                |
+-------------------------+
8 rows in set (0.00 sec)
```

In some cases, you need to see the table's metadata, you can use DESCRIBE statement as follows:

```
DESCRIBE table_name;
```

For instance, we can describe employees table like below query:

```
DESCRIBE employees;
```

The output return from the database server:

```
+----------------+--------------+------+-----+---------+-------+
| Field          | Type         | Null | Key | Default | Extra |
+----------------+--------------+------+-----+---------+-------+
| employeeNumber | int(11)      | NO   | PRI | NULL    |       |
| lastName       | varchar(50)  | NO   |     | NULL    |       |
| firstName      | varchar(50)  | NO   |     | NULL    |       |
| extension      | varchar(10)  | NO   |     | NULL    |       |
| email          | varchar(100) | NO   |     | NULL    |       |
| officeCode     | varchar(10)  | NO   |     | NULL    |       |
| reportsTo      | int(11)      | YES  |     | NULL    |       |
| jobTitle       | varchar(50)  | NO   |     | NULL    |       |
+----------------+--------------+------+-----+---------+-------+
8 rows in set (0.02 sec)
```

Altering Table Structures

Besides creating table, MySQL allows you to alter existing table structures with a lot of options. To modify existing database table structure you use the ALTER TABLE statement. The following illustrates the ALTER TABLE statement syntax:

```
ALTER [IGNORE] TABLE table_name options[, options...]
options:
        ADD [COLUMN] create_definition [FIRST | AFTER col_name ]
    or  ADD [COLUMN] (create_definition, create_definition,...)
    or  ADD INDEX [index_name] (index_col_name,...)
    or  ADD PRIMARY KEY (index_col_name,...)
    or  ADD UNIQUE [index_name] (index_col_name,...)
    or  ADD FULLTEXT [index_name] (index_col_name,...)
    or  ADD [CONSTRAINT symbol] FOREIGN KEY [index_name] (index_col_name,...)  [reference Definition ]
    or  ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
    or  CHANGE [COLUMN] old_col_name create_definition [FIRST | AFTER column_name]
    or  MODIFY [COLUMN] create_definition [FIRST | AFTER col_name]
    or  DROP [COLUMN] col_name
    or  DROP PRIMARY KEY
    or  DROP INDEX index_name
    or  DISABLE KEYS
    or  ENABLE KEYS
    or  RENAME [TO] new_table_name
    or  ORDER BY col_name
    or  table_options
```

Most of these options are obvious. We will explain some here:

- The CHANGE and MODIFY are the same, they allow you to change the definition of the column or its position in the table.
- The DROP COLUMN will drop the column of the table permanently, if the table contain data all the data of the column will be lost.
- The DROP PRIMARY KEY and DROP INDEX only remove the primary key or index of the column.
- The DISABLE and ENABLE KEYS turn off and on updating indexes for MyISAM table only.
- The RENAME Clause allows you the change the table name to the new one.

Deleting Tables

To delete table from the database, you can use DROP TABLE statement:

```
DROP [TEMPORARY] TABLE [IF EXISTS] table_name [, table_name,...]
```

TEMPORARY keyword is used for deleting temporary tables. MySQL allows you to drop multiple tables at once by listing them and separated each by a comma. IF EXISTS is used to prevent you from deleting table which does not exist in the database.

Empty Table's Data

In some cases, you want to delete all table data in a fast way and reset all auto increment columns. MySQL also provides you SQL TRUNCATE table statement to allow you to do so. The SQL TRUNCATE statement is as follows:

```
TRUNCATE TABLE table_name
```

There are some points you should remember before using TRUNCATE TABLE statement:

- TRUNCATE TABLE statement drop table and recreate it therefore it is much faster than DELETE TABLE statement. However it is not transaction-safe.
- The number of deleted rows is not returned like SQL DELETE TABLE statement.
- ON DELETE triggers are not invoked because TRUNCATE does not use DELETE statement.

MySQL ALTER TABLE syntax

MySQL ALTER TABLE statement is used to change the structure of existing tables. You can use MySQL ALTER TABLE to add or drop column, change column data type, add primary key, rename table and a lot more. The following illustrates the MySQL ALTER TABLE syntax:

```
ALTER TABLE table_name action1[,action2,...]
```

Followed by the keyword ALTER TABLE is name of table that you want to make the changes. After the table name is the action you want apply to the table. An action can be anything from add a new column, add primary key…. to rename table. MySQL allows you to do multiple actions at a time, separated by a comma.

Let's create a new table for practicing MySQL ALTER TABLE statement. We're going to create a new table called *tasks* in our sample database *classicmodels* as follows*:*

```
CREATE  TABLE 'tasks' ('task_id' INT NOT NULL ,
                'subject' VARCHAR(45) NULL ,
                'start_date' DATETIME NULL ,
                'end_date' DATETIME NULL ,
                'description' VARCHAR(200) NULL ,
                PRIMARY KEY ('task_id') ,
                UNIQUE INDEX 'task_id_UNIQUE' ('task_id' ASC) );
```

Changing columns using MySQL ALTER TABLE statement

Using MySQL ALTER TABLE to add auto-increment for a column

Suppose we want the task id is increased by one automatically whenever we insert a new task. In order to accomplish this, we need to use the MySQL ALTER TABLE statement to change the column task id to make it auto increment as follows:

```
ALTER TABLE tasks
CHANGE COLUMN task_id task_id INT(11) NOT NULL AUTO_INCREMENT;
```

Using MySQL ALTER TABLE to add a new column into a table

Because of the new business requirement, we need to add a new column called *complete* to store completion percentage for each task in the *tasks* table. In this case, we can use MySQL ALTER TABLE to add a new column as follows:

```
ALTER TABLE tasks ADD COLUMN 'complete' DECIMAL(2,1) NULL
AFTER 'description' ;
```

Using MySQL ALTER TABLE to drop a column from a table

Let's say we don't want to store task description in the task table anymore so we have to remove that column. Here is the SQL command to drop a column from a table:

```
ALTER TABLE tasks
DROP COLUMN description;
```

Renaming table using MySQL ALTER TABLE statement

We can use MySQL ALTER table statement to rename a table. Note that before renaming a table you should take a serious consideration to see its dependencies from database to application level. We can rename our *tasks* table to *work_items* as follows:

```
ALTER TABLE 'tasks'
RENAME TO  'work_items' ;
```

MySQL SELECT Statement

In order to retrieve data from MySQL database table you need to use ***MySQL SELECT*** statement. The following illustrates MySQL SELECT statment syntax:

```
SELECT column_name1,column_name2...
FROM tables
[WHERE conditions]
[GROUP BY group]
[HAVING group_conditions]]
[ORDER BY sort_columns]
[LIMIT limits];
```

The MySQL SELECT statement has many optional elements that you can use. The order of FROM, WHERE, GROUP BY, HAVING, ORDER BY and LIMIT has to be in the sequence above.

To select all columns in a table you can use asterisk (*) notation instead of listing all column names in the MySQL SELECT statement. For example, if you need to query all the columns in *offices* table, you can use the following query:

```
SELECT * FROM employees;
```

employeeNumber	lastName	firstName	extension	email	officeCode	reportsTo	jobTitle
1002	Murphy	Diane	x5800	dmurphy@classicmodelcars.com	1	NULL	President
1056	Patterson	Mary	x4611	mpatterso@classicmodelcars.com	1	1002	VP Sales
1076	Firrelli	Jeff	x9273	jfirrelli@classicmodelcars.com	1	1002	VP Marketing
1088	Patterson	William	x4871	wpatterson@classicmodelcars.com	6	1056	Sales Manager (APAC)
1102	Bondur	Gerard	x5408	gbondur@classicmodelcars.com	4	1056	Sale Manager (EMEA)
1143	Bow	Anthony	x5428	abow@classicmodelcars.com	1	1056	Sales Manager (NA)
1165	Jennings	Leslie	x3291	ljennings@classicmodelcars.com	1	1143	Sales Rep
1166	Thompson	Leslie	x4065	lthompson@classicmodelcars.com	1	1143	Sales Rep
1188	Firrelli	Julie	x2173	jfirrelli@classicmodelcars.com	2	1143	Sales Rep
1216	Patterson	Steve	x4334	spatterson@classicmodelcars.com	2	1143	Sales Rep
1286	Tseng	Foon Yue	x2248	ftseng@classicmodelcars.com	3	1143	Sales Rep
1323	Vanauf	George	x4102	gvanauf@classicmodelcars.com	3	1143	Sales Rep

The *MySQL SELECT* statement also allows you to to view partial data of a table by listing columns' name after the SELECT keyword. This is called *projection*. For example if you need to view only *first name*, *last name* and *job title* of employee in the *employees* table, you can use the following query:

```
SELECT lastname,firstname,jobtitle
FROM employees;
```

lastname	firstname	jobtitle
Murphy	Diane	President
Patterson	Mary	VP Sales
Firrelli	Jeff	VP Marketing
Patterson	William	Sales Manager (APAC)
Bondur	Gerard	Sale Manager (EMEA)
Bow	Anthony	Sales Manager (NA)
Jennings	Leslie	Sales Rep
Thompson	Leslie	Sales Rep
Firrelli	Julie	Sales Rep
Patterson	Steve	Sales Rep
Tseng	Foon Yue	Sales Rep
Vanauf	George	Sales Rep

WHERE Clause

WHERE clause of the MySQL SELECT statement enables you to select particular rows which match its conditions or search criteria. You use WHERE clause to filter the records based on a certain conditions. For example, you can find the president of company by using the following query:

```
SELECT firstname,lastname,email
FROM employees
WHERE jobtitle="president"
```

firstname	lastname	email
Diane	Murphy	dmurphy@classicmodelcars.com

DISTINCT

With DISTINCT keyword, you can eliminate the duplicate records from the SELECT statement. For example, to find how many *job title* of all employees in the *employees* table, you use DISTINCT keyword in SELECT statement as follows:

```
SELECT DISTINCT jobTitle FROM employees;
```

jobTitle
President
VP Sales
VP Marketing
Sales Manager (APAC)
Sale Manager (EMEA)
Sales Manager (NA)
Sales Rep

Sorting result with ORDER BY

The ORDER BY clause allows you to sort the result set on one or more columns in ascending or descending order. To sort the result set in ascending order you use ASC and in descending order you use DESC keywords. By default, the ORDER BY will sort the result set in ascending order. For example, to sort the name of employees by ***first name*** and ***job title***, you can execute the following query:

```
SELECT firstname,lastname, jobtitle
FROM employees
ORDER BY firstname ASC,jobtitle DESC;
```

firstname	lastname	jobtitle
Andy	Fixter	Sales Rep
Anthony	Bow	Sales Manager (NA)
Barry	Jones	Sales Rep
Diane	Murphy	President
Foon Yue	Tseng	Sales Rep
George	Vanauf	Sales Rep
Gerard	Hernandez	Sales Rep
Gerard	Bondur	Sale Manager (EMEA)
Jeff	Firrelli	VP Marketing
Julie	Firrelli	Sales Rep
Larry	Bott	Sales Rep
Leslie	Jennings	Sales Rep
Leslie	Thompson	Sales Rep

How to Use MySQL Limit to Constrain Number of Returned Records

Most of the times, when you work with master data tables which contain thousand to millions of records and you don't want to write a query to get all the data from those tables because of application's performance and high traffic between database server and application server. MySQL supports a cool feature called LIMIT to allow you to constrain the returned records with SELECT statement. Let take a look at the MySQL LIMIT syntax:
Let's say you have a database table with 10000 records and you want to get just first N records, you can use the following query:

```
SELECT * FROM table
LIMIT N
```

The MySQL LIMIT also allows you to get a range of records where you decide starting record number and how many records you want to retrieve. Here is the syntax of MySQL LIMIT to select a range of records:

```
SELECT columns
From table
Limit S, N;
```

In the query above, S is the starting record index. MySQL specifies that the first record starts with 0. N is the number of records you want to select.

If you want to get the first five employees in the table *employees*, you can use the following query:

```
SELECT firstname,lastname
From employees
Limit 5
```

```
-----------+-----------+
| firstname | lastname  |
+-----------+-----------+
| Diane     | Murphy    |
| Mary      | Patterson |
| Jeff      | Firrelli  |
| William   | Patterson |
| Gerard    | Bondur    |
+-----------+-----------+
5 rows in set (0.00 sec)
```

Now if you want to get five employees from employee number 10 you can use MySQL LIMIT with offset as follows:

```
SELECT firstname,lastname
FROM employees
LIMIT 10,5
```

```
+-----------+-----------+
| firstname | lastname  |
+-----------+-----------+
| Foon Yue  | Tseng     |
| George    | Vanauf    |
| Loui      | Bondur    |
| Gerard    | Hernandez |
| Pamela    | Castillo  |
+-----------+-----------+
5 rows in set (0.00 sec)
```

Retrieving Data in a Range Using SQL BETWEEN

The SQL BETWEEN operator allows you to retrieve values within a specific range. the SQL between must be used in the WHERE clause of the SQL SELECT statement. The following illustrates the SQL BETWEEN syntax:

```
SELECT column_list
FROM table_name
WHERE column_1 BETWEEN lower_range AND upper_range
```

MySQL returns all records in which the column_1 value is in the range of *lower_rage* and *upper_range* as well as the values lower_rage and upper_range. The query which is equivalent to SQL BETWEEN to get the same result is

```
SELECT column_list
FROM table_name
WHERE column_1 >= lower_range AND column_1 <= upper_range
```

Suppose we want to find all products which buy price is in a range of 90$ and 100$, we can perform the following query to do so:

```
SELECT productCode,ProductName,buyPrice
FROM products
WHERE buyPrice BETWEEN 90 AND 100
ORDER BY buyPrice DESC;
```

Here is the output

```
+-------------+---------------------------------------+----------+
| productCode | ProductName                           | buyPrice |
+-------------+---------------------------------------+----------+
| S10_1949    | 1952 Alpine Renault 1300              |    98.58 |
| S24_3856    | 1956 Porsche 356A Coupe               |     98.3 |
| S12_1108    | 2001 Ferrari Enzo                     |    95.59 |
| S12_1099    | 1968 Ford Mustang                     |    95.34 |
+-------------+---------------------------------------+----------+
```

The output contains all products in the range of 90$ and 100$, and if there is a product with buy price 90$ or 100$, it will be included in the output too.

In order to find all records which are not in a range we use NOT BETWEEN. To find all products that buy price outside the range of 20 and 100, we can operate following query:

```
SELECT productCode,ProductName,buyPrice
FROM products
WHERE buyPrice NOT BETWEEN 20 AND 100
```

```
+-------------+---------------------------------------+----------+
| productCode | ProductName                           | buyPrice |
+-------------+---------------------------------------+----------+
| S10_4962    | 1962 LanciaA Delta 16V                |   103.42 |
| S24_2840    | 1958 Chevy Corvette Limited Edition |    15.91 |
```

```
+-------------+-----------------------------------+----------+
```

The query above is equivalent to the following query

```
SELECT productCode,ProductName,buyPrice
FROM products
WHERE buyPrice < 20 OR buyPrice > 100
ORDER BY buyPrice DESC
```

How to Use MySQL LIKE to Select Data Based on Patterns Matching

*MySQL provides LIKE operator in SQL standard. The **MySQL LIKE** operator is commonly used to select data based on patterns matching. Using **MySQL LIKE** in appropriate way is essential to increase application's performance.*

MySQL LIKE allows you to perform pattern matching in your characters column in a database table. MySQL LIKE is often used with SELECT statement in WHERE clause. MySQL provides you two wildcard characters for using with LIKE, the percentage % and underscore _.

- Percentage (%) wildcard allows you to match any string of zero or more characters
- Underscore (_) allows you to match any single character.

Let's practice with couples of examples which use MySQL Like with different wildcard characters.

Suppose we want to search for employee in *employees* table who has first name starting with character 'a', we can do it as follows:

```
SELECT employeeNumber, lastName, firstName
FROM employees
WHERE firName LIKE 'a%';
```

```
+----------------+----------+-----------+
| employeeNumber | lastName | firstName |
+----------------+----------+-----------+
|         1611 | Fixter   | Andy      |
+----------------+----------+-----------+
1 row in set (0.00 sec)
```

MySQL scans the whole *employees* table to find all employees which have first name starting with character 'a' and followed by any number of characters.

To search all employees which have last name ended with 'on' string you can perform the query as follows:

```
SELECT employeeNumber, lastName, firstName
FROM employees
WHERE lastName LIKE '%on';
```

```
+----------------+-----------+-----------+
| employeeNumber | lastName  | firstName |
+----------------+-----------+-----------+
|         1088 | Patterson | William   |
|         1216 | Patterson | Steve     |
+----------------+-----------+-----------+
2 rows in set (0.00 sec)
```

If you know a searched string is embedded somewhere in a column, you can put the percentage wild card at the beginning and the end of it to find all possibilities. For example, if you want to find all employees which have last name containing 'on' string you can execute following query:

```
SELECT employeeNumber, lastName, firstName
FROM employees
WHERE lastName LIKE '%on%';
```

```
+----------------+-----------+-----------+
| employeeNumber | lastName  | firstName |
+----------------+-----------+-----------+
|         1088 | Patterson | William   |
|         1102 | Bondur    | Gerard    |
|         1504 | Jones     | Barry     |
+----------------+-----------+-----------+
3 rows in set (0.00 sec)
```

To search all employees whose name are such as Tom, Tim… You can use underscore wildcard

```
SELECT employeeNumber, lastName, firstName
FROM employees
WHERE firstName LIKE 'T__';
```

```
+----------------+----------+-----------+
| employeeNumber | lastName | firstName |
+----------------+----------+-----------+
|         1619 | King     | Tom       |
+----------------+----------+-----------+
1 row in set (0.00 sec)
```

The MySQL LIKE allows you to put the NOT keyword to find all strings which are unmatched with a specific pattern. Suppose you want to search for all employees whose last name are not starting with 'B', you can use the following query

```
SELECT employeeNumber, lastName, firstName
FROM employees
WHERE lastName NOT LIKE 'B%';
```

```
+----------------+-----------+-----------+
| employeeNumber | lastName  | firstName |
+----------------+-----------+-----------+
|         1088 | Patterson | William   |
|         1188 | Firrelli  | Julie     |
|         1216 | Patterson | Steve     |
+----------------+-----------+-----------+
3 rows in set (0.00 sec)
```

Be noted that SQL LIKE is not case sensitive so 'b%' and 'B%' are the same.

What if you want to search for records which have a field starting with a wildcard character? In this case, you can use ESCAPE to shows that the wildcard characters followed it has literal meaning not wildcard meaning. If ESCAPE does not specify explicitly, the escape character in MySQL by default is '\'. For example, if you want to find all products which as product code which has _20 embedded on it, you can perform following query

```
SELECT productCode, productName
FROM products
WHERE productCode LIKE '%\_20%';
```

```
+-------------+-----------------------------------------+
| productCode | productName                             |
+-------------+-----------------------------------------+
| S10_2016    | 1996 Moto Guzzi 1100i                   |
| S24_2000    | 1960 BSA Gold Star DBD34                |
+-------------+-----------------------------------------+
2 rows in set (0.00 sec)
```

MySQL LIKE gives you a convenient way to find records which have character columns match specified patterns. Because MySQL LIKE scans the whole table to find all the matching records therefore it does not allow database engine to use the index for fast searching. When the data in the table is big enough, the performance of MySQL LIKE will degrade. In some cases you can avoid this problem by using other techniques to achieve the same result as MySQL LIKE. For example, if you want to find all employees which have first name starting with a specified string you can use LEFT function in where clause like the following query:

```
SET @str = 'b';
SELECT employeeNumber, lastName, firstName
FROM employees
WHERE LEFT(lastname,length(@str)) =@str;
```

It returns the same result as the query below but it faster because we can leverage the index on the column *lastname*.

```
SELECT employeeNumber, lastName, firstName
FROM employees
WHERE lastname LIKE 'b%'
```

And another technique to achieve all string which end with a specified string by using RIGHT function. Suppose we want to retrieve all employees which have last name ended with 'on' string, we can use RIGHT function instead of MySQL LIKE as follows:

```
SET @str = 'on';
SELECT employeeNumber, lastName, firstName
FROM employees
WHERE RIGHT(lastname,length(@str)) = @str;
```

```
+----------------+-----------+-----------+
| employeeNumber | lastName  | firstName |
+----------------+-----------+-----------+
|         1088 | Patterson | William   |
|         1216 | Patterson | Steve     |
+----------------+-----------+-----------+
2 rows in set (0.00 sec)
```

It returns the same result as the following query

```
SELECT employeeNumber, lastName, firstName
FROM employees
WHERE lastname LIKE '%on'
```

Combining Result Sets with MySQL UNION

Like SQL standard, MySQL UNION allows you to combine two or more result sets from multiple tables together. The syntax of using MySQL UNION is as follows:

```
SELECT statement
UNION [DISTINCT | ALL]
SELECT statement
UNION [DISTINCT | ALL]
.....
```

In order to use UNION statement, there are some rules you need to follow:

- The number of columns in each SELECT statement has to be the same .
- The data type of the column in the column list of the SELECT statement must be the same or at least convertible.

By default the MySQL UNION removes all duplicate rows from the result set even if you don't explicit using DISTINCT after the keyword UNION.

If you use UNION ALL explicitly, the duplicate rows remain in the result set. You only use this in the cases that you want to keep duplicate rows or you are sure that there is no duplicate rows in the result set.

Suppose you want to combine ***customers*** and ***employees*** information into one result set, you use the following query:

```
SELECT customerNumber id, contactLastname name
FROM customers
UNION
SELECT employeeNumber id,firstname name
FROM employees;
```

Here is the excerpt of the output:

```
 id  name
------  ---------------
  103  Schmitt
  112  King
```

```
114  Ferguson
119  Labrune
121  Bergulfsen
124  Nelson
```

When using ORDER BY to sort the result with UNION, you have to use it in the last SQL SELECT statement. It would be the best to parenthesize all the SELECT statements and place ORDER BY at the end.

Suppose you want to sort the combination of ***employees*** and ***customers*** in the query above by ***name*** and ***ID*** in ascending order.

```
(SELECT customerNumber id,contactLastname name
FROM customers)
UNION
(SELECT employeeNumber id,firstname name
FROM employees
ORDER BY name,id;
```

What will be displayed in the output if we don't use alias for each column in the SELECT statements? MySQL will use the column names of the first SELECT statement as the label of the output.

```
(SELECT customerNumber, contactLastname
FROM customers)
UNION
(SELECT employeeNumber, firstname
FROM employees)
ORDER BY contactLastname, customerNumber;
```

MySQL also provides you another option to sort the result set based on column position in the ORDER BY clause as the following query:

```
(SELECT customerNumber, contactLastname
FROM customers)
UNION
(SELECT employeeNumber,firstname
FROM employees)
ORDER BY 2,1
```

MySQL INNER JOIN

Introducing MySQL INNER JOIN clause

The *MySQL INNER JOIN* clause is an optional part of SQL SELECT statement. The MySQL INNER JOIN clause appears immediately after the FROM clause. Before using MySQL INNER JOIN clause, you have to specify the following criteria:

- First, you need to specify the tables you want to join with the main table. The main table appear in the FROM clause. The table you want to join with appear after keyword INNER JOIN. Theoretically, you can join a table with unlimited number of tables. However, for better performance you should limit the number of tables to join based on join conditions and volume of data in those tables.

- Second, you need to specify the join condition or join predicate. The join condition appears after the keyword ON of MySQL INNER JOIN clause. The join condition is the rule for matching rows between the main table and other tables being joined with.
  The syntax of the MySQL INNER JOIN is as follows:

```
SELECT column_list
FROM t1
INNER JOIN t2 ON join_condition1
....
WHERE where_condition;
```

For example, if you join two tables A and B, the MySQL INNER JOIN clause compares each record of the table A with each record of table B to find all pair of records that satisfy the join-condition. When the join-condition are satisfied, column values for each matched pair of record of table A and table B are combined into a returned record. Note that the records on both tables have to match based on the join-condition. If no record on both table A and B matches, the query will return an empty result.

Avoid column ambiguous error in MySQL INNER JOIN

If you join multiple tables that has column with similar name, you have to use table qualifier to refer to column to avoid column ambiguous error. Suppose if table *tbl_A* and *tbl_B* has the same column *M*. In the SELECT statement with MySQL INNER JOIN clause, you have to refer to column *M* by using the table qualifier as *tbl_A.M* or *tbl_B.M* (*table_name.column_name*).

Another effective way to avoid column ambiguous is by using table alias. For example, you can give A as the table alias of the table *tbl_A* and refer to the column *M* as *A.M* so you don't have to type again and again the long table name in your SQL statement.

Example of MySQL INNER JOIN clause

Let's take a look at two tables: products and orderDetails in our sample database.

The products table is the master data table that stores all products. Whenever a product is sold, it is stored in the orderDetails table with other information. The link between products table and orderDetails table is productCode.
Now, if you want to know what product was sold in which order, you can use the *MySQL INNER JOIN* clause as follows:

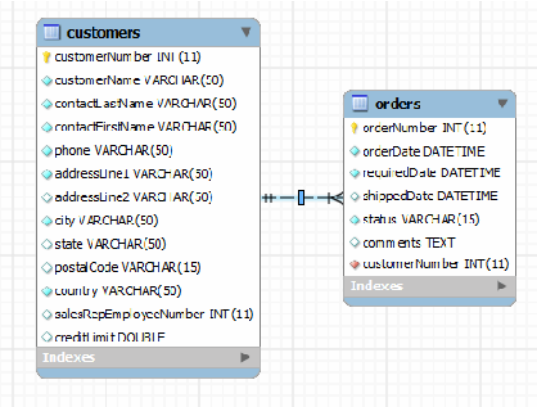
```
SELECT A.productCode, A.productName, B.orderNumber
FROM products A
INNER JOIN orderDetails B on A.productCode = B.productCode;
```

productCode	productName	orderNumber
S18_1749	1917 Grand Touring Sedan	10100
S18_2248	1911 Ford Town Car	10100
S18_4409	1932 Alfa Romeo 8C2300 Spider Sport	10100
S24_3969	1936 Mercedes Benz 500k Roadster	10100
S18_2325	1932 Model A Ford J-Coupe	10101
S18_2795	1928 Mercedes-Benz SSK	10101
S24_1937	1939 Chevrolet Deluxe Coupe	10101
S24_2022	1938 Cadillac V-16 Presidential Limousine	10101
S18_1342	1937 Lincoln Berline	10102
S18_1367	1936 Mercedes-Benz 500K Special Roadster	10102
S10_1949	1952 Alpine Renault 1300	10103
S10_4962	1962 LanciaA Delta 16V	10103
S12_1666	1958 Setra Bus	10103
S18_1097	1940 Ford Pickup Truck	10103

The MySQL INNER JOIN clause compares each row of table products and orderDetails table to find a pair of rows that has the same productCode. If a pair of rows that have the same, the product code, product name and order number are combined into a returned row.

MySQL LEFT JOIN

Introducing to MySQL LEFT JOIN



The MySQL LEFT JOIN clause is an optional element of the SQL SELECT statement to allow you to retrieve data from additional tables. The MySQL LEFT JOIN clause consists of LEFT JOIN keyword followed by the second table you want to join. Next element is the ON keyword followed by the join condition. In the join condition, you specify the columns to be used for matching row in the two tables.

The syntax of MySQL is as follows:

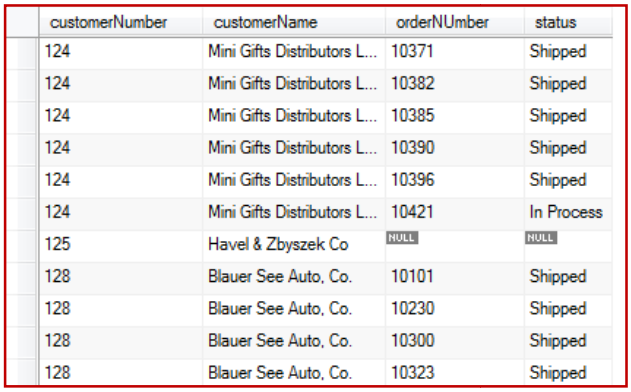
```
SELECT t1.c1, t1.c2,...t2.c1,t2.c2
FROM t1
LEFT JOIN t2 ON t1.c1 = t2.c1 ...(join_condition)
WHERE where_condition;
```

The MySQL LEFT JOIN clause works like this: when a row from the left table matches a row from the right table based on join_condition, the row's content are selected as an output row. When row in the left table has no match it is still selected for output, but combined with a "fake" row from the right table that contains NULL in all columns. In short, the MySQL LEFT JOIN clause allows you to select all rows from the left table even there is no match for them in the right table.

Example of MySQL LEFT JOIN

Let's take a look at two table customers and orders: If you want to know which customer has which order and each order's status. You can use the MySQL LEFT JOIN as follows:

```
SELECT c.customerNumber, customerName,orderNUmber, o.status
FROM customer c
LEFT JOIN orders o ON c.customerNumber = o.customerNumber;
```



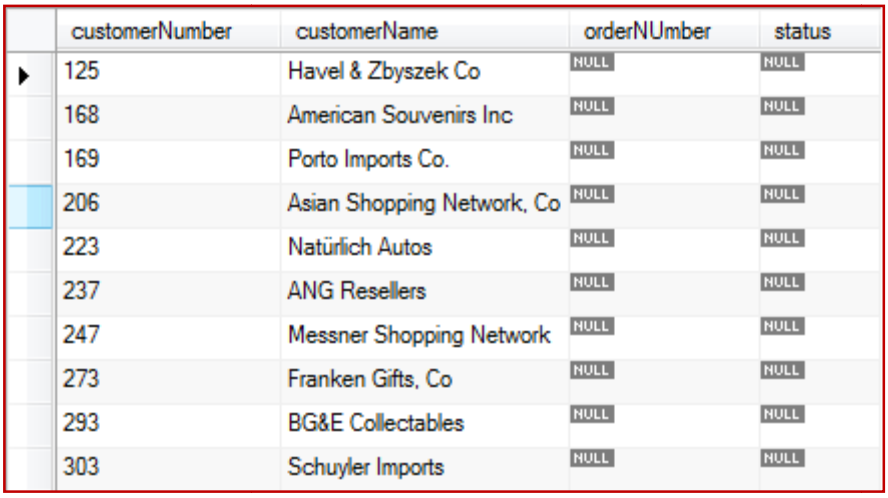
customerNumber	customerName	orderNUmber	status
124	Mini Gifts Distributors L...	10371	Shipped
124	Mini Gifts Distributors L...	10382	Shipped
124	Mini Gifts Distributors L...	10385	Shipped
124	Mini Gifts Distributors L...	10390	Shipped
124	Mini Gifts Distributors L...	10396	Shipped
124	Mini Gifts Distributors L...	10421	In Process
125	Havel & Zbyszek Co	NULL	NULL
128	Blauer See Auto, Co.	10101	Shipped
128	Blauer See Auto, Co.	10230	Shipped
128	Blauer See Auto, Co.	10300	Shipped
128	Blauer See Auto, Co.	10323	Shipped

The left table is the customers table so you see all customers are listed as the way MySQL LEFT JOIN clause works. However, there are rows that we have customer information but all order information are NULL. This means those customers do not have any order in our database. The MySQL LEFT JOIN clause is very useful when you want to find the records in the left table that are unmatched by the right table. You can accomplish this by add a WHERE clause to select only that have NULL values in a right table column.

So to find all customers who does not have any order in our database we can use the MySQL LEFT JOIN clause as follows:

```
SELECT c.customerNumber, customerName,orderNUmber, o.status
FROM customer c
LEFT JOIN orders o ON c.customerNumber = o.customerNumber
WHERE orderNumber is NULL
```

As the result, the query only returns customers which do not have any order represented by NULL values.



customerNumber	customerName	orderNUmber	status
125	Havel & Zbyszek Co	NULL	NULL
168	American Souvenirs Inc	NULL	NULL
169	Porto Imports Co.	NULL	NULL
206	Asian Shopping Network, Co	NULL	NULL
223	Natürlich Autos	NULL	NULL
237	ANG Resellers	NULL	NULL
247	Messner Shopping Network	NULL	NULL
273	Franken Gifts, Co	NULL	NULL
293	BG&E Collectables	NULL	NULL
303	Schuyler Imports	NULL	NULL

MySQL GROUP BY

The ***MySQL GROUP BY*** clause is used with SQL SELECT statement to to group selected records into a set of summary records by the one or more column's value or expression. The MySQL GROUP BY clause is an optional part of the SQL SELECT statement. The MySQL GROUP BY clause must appear after the WHERE clause or FROM clause if WHERE clause is omitted of the SQL SELECT statement. The MySQL GROUP BY clause consists of GROUP BY keyword followed by a list of expressions separated by commas.
The following illustrates the MySQL GROUP BY clause:

```
SELECT col1_,col_2,... col_n, aggregate_function(expression)
FROM table
WHERE where_conditions
GROUP BY col_1, col_2, ... col_n
ORDER BY column_list;
```

We will examine the GROUP BY clause in details. Let's take a look at the order table in our sample database.

Example of MySQL GROUP BY clause

Now, suppose you want to get groups for each order, you can use the MySQL GROUP BY clause as follows:

```
SELECT status
FROM orders
GROUP BY status
```

```
+------------+
| status     |
+------------+
| Cancelled  |
| Disputed   |
| In Process |
| On Hold    |
+------------+
4 rows in set (0.00 sec)
```

It seems that the GROUP BY clause only scans for unique occurrences in the status column and return the result set. However if you look at the data of the orders table you will see that each row in result set above is summary of records that represent a group orders that have the same status on the status column.

MySQL GROUP BY with aggregate function

The aggregate functions allow you to perform calculation of a set of records and return a single value. The most common aggregate functions are SUM, AVG, MAX, MIN and COUNT.

Aggregate functions are used with MySQL GROUP BY clause to perform calculation on each group of records on return a single value for each row. Let's say if you want to know how many orders in each status group you can use the COUNT function as follows:

```
SELECT status, count(*)
FROM orders
GROUP BY status
```

```
+------------+----------+
```

```
| status     | count(*) |
+------------+----------+
| Cancelled  |        6 |
| Disputed   |        3 |
| Resolved   |        4 |
| Shipped    |      303 |
+------------+----------+
4 rows in set (0.00 sec)
```

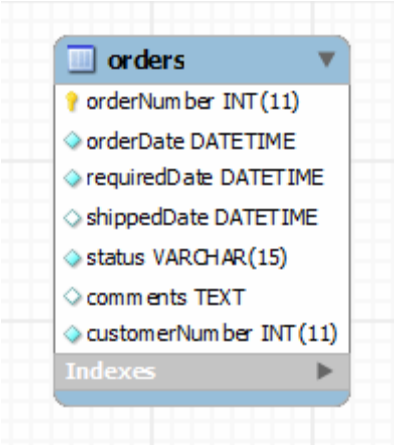
The query counts number of orders for each order's status.

MySQL GROUP BY vs. ANSI SQL GROUP BY

MySQL follows ANSI SQL. However, there are two differences the way GROUP BY works in MySQL and ANSI SQL.

- In ANSI SQL you must group by all columns you specifies in the SELECT clause. MySQL does not have this restriction. You can have additional columns in the SELECT clause that are not in the GROUP BY clause.
- MySQL also allows you to sort the group order in which the results are returned. The default order is ascending.

MySQL HAVING



Introducing MySQL HAVING clause

The MySQL HAVING clause is an optional part of and used only with the SQL SELECT statement. The MySQL HAVING clause specifies a filter condition for a group of record or an aggregate. The MySQL HAVING is often used with MySQL GROUP BY clause. When using with MYSQL GROUP BY clause, you can apply filter condition of the HAVING clause only to the columns appear in the GROUP BY clause. If the MySQL GROUP BY clause is omitted, the MySQL HAVING clause will behave like a WHERE clause. Notes that the MySQL HAVING clause applies to groups as a whole while the WHERE clause applies to individual rows.

Examples of MySQL HAVING clause

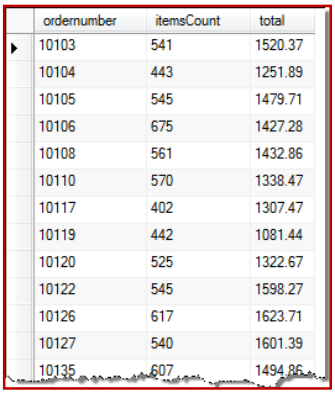
We have *orderDetails* table in our sample database. We can use the MySQL GROUP BY clause to get all orders, number of items sold and total values in each order as follows:

```
SELECT ordernumber, sum(quantityOrdered) AS itemsCount, sum(priceeach) AS total
FROm orderdetails
GROUP BY ordernumber;
```

ordernumber	itemsCount	total
10100	151	301.84
10101	142	352
10102	80	138.68
10103	541	1520.37
10104	443	1251.89
10105	545	1479.71
10106	575	1427.28
10107	229	793.21
10108	561	1432.86
10109	212	700.89

Now you can ask what order has total value greater than $1000. In this case, you need to use the MySQL HAVING clause on aggregate to answer that question.

```
SELECT ordernumber, sum(quantityOrdered) AS itemsCount, sum(priceeach) AS total
FROM orderdetails
GROUP BY  orderdetails
HAVING total > 1000;
```



ordernumber	itemsCount	total
10103	541	1520.37
10104	443	1251.89
10105	545	1479.71
10106	675	1427.28
10108	561	1432.86
10110	570	1338.47
10117	402	1307.47
10119	442	1081.44
10120	525	1322.67
10122	545	1598.27
10126	617	1623.71
10127	540	1601.39
10135	607	1494.86

We use column alias for the aggregate *sum(priceeach)* as *total* so in the HAVING clause we just have to specify that column alias *total* instead of typing the aggregate *sum(priceeach)* again.

You can use a complex condition in the MySQL HAVING clause such as OR, AND operators. For example if you want to know what order has total value greater than $1000 and has more than 600 items in it. You can use the following query to find out:

```
SELECT ordernumber, sum(quantityOrdered) AS itemsCount, sum(priceeach) AS total
FROM orderdetails
GROUP BY ordernumber
HAVING total > 1000 AND itemscount > 600;
```

ordernumber	itemsCount	total
10106	675	1427.28
10126	617	1623.71
10135	607	1494.86
10165	670	1794.94
10168	642	1472.5
10204	619	1619.73
10207	615	1560.08
10212	612	1541.83
10222	717	1389.51
10262	605	1217.38
10275	601	1455.41
10310	619	1656.26
10312	601	1494.19

The MySQL HAVING clause is useful only with the MySQL GROUP BY clause for building output of high-level reports. For example, you can use the MySQL HAVING clause to answer questions like how many order has total values more than 1000 this month, this quarter and this year?...

MySQL INSERT Statement

INSERT statement allows you to insert one or more rows to the table. In MySQL, the INSERT statement forms are listed as follows:

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE][INTO] table_name [(column_name,...)]
VALUES ((expression | DEFAULT),...),(...),...;
```

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE] [INTO] table_name [(column_name,...)];
```

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE] [INTO] table_name
SET column_name=(expression|DEFAULT),...
```

As you can see INTO in the INSERT statement is optional. In the first form, you insert a new data row into an existing table by specifying the column name and data for each.

As an example to insert a new office to the *offices* table in the sample database you can do as follows:

```
INSERT INTO offices (officeCode,city,phone,addressLin1,addressLine2,state,country,postalCode)
VALUES ('8', 'surat', '+261 2475754', 'nanpura', 'abc street', 'gujarat', 'india', '395001');
```

In the second form, instead of providing explicit data, you select it from other table by using SELECT statement. This form allows you to copy some data or some part of data from other table to the inserted table.

As an example, we can create a temporary table and insert all offices which locate in US into that one by using this query:

```
INSERT INTO temp_table
SELECT * FROM offices WHERE country='india' ;
```

The third form enables you to specify the column you want to insert the data. For example, we have the query like this:

```
INSERT INTO productlines
SET productline='Cars'  ;
```

It means we only insert the data into *productLine* column in *productLines* table.

MySQL UPDATE Statement

*Updating existing data is one of the most important task when you work with database. In this tutorial, you will learn how to use **MySQL UPDATE** statement to update data in database tables.*

SQL UPDATE statement is used to update existing data in database tables. It can be used to change values of single row, group of rows or even all rows in a table. In MySQL, the SQL UPDATE statement form is as below:

```
UPDATE [LOW_ PRIORITY] [IGNORE] table_name [, table_name...]
SET column_name1=expr1[,column_name2=expr2...]
[WHERE condition ]
```

Let's examine the MySQL UPDATE statement in details:

- Followed by the UPDATE keyword is the name of a table you want to change the data. In MySQL, you can change the data of multiple tables at a time. If an UPDATE command violates any integrity constraint, MySQL does not perform the update and it will issue an error message.

- The SET clause determines the columns' name and the changed values. The changed values could be a constant value, expression or even a subquery.
- WHERE clause determines which rows of the tables will be updated. It is an optional part of SQL UPDATE statement. If WHERE clause is ignored, all rows in the tables will be updated. The WHERE clause is so important that you should not forget. Sometimes you want to change just one row of a table but you forget WHERE clause so you accidentally update the whole table.
- LOW_ PRIORITY keyword is used to delay the execution until no other client applications reading data from the table. This is used for controlling the update process in MySQL database server.
- IGNORE keyword is used to execute the update even errors occurred during the execution. The errors in the update process could be duplicate value on unique column, or new data does not match with the column data type. In the first situation data is not updated and in the second one MySQL tries to convert the data into closest valid values.

Let's practice with a couple of examples in our sample database to understand more about SQL UPDATE statement.

In ***employees*** table, if you want to update the email of Mary Patterson with employeeNumber 1 with the new email as mary-patterso@classicmodelcars.com, you can execute the following query:

```
SELECT firstname, lastname,email
FROM employees
WHERE employeeNumber =1;
```

```
+-----------+-----------+-------------------------------+
| lastname  | firstname | email                         |
+-----------+-----------+-------------------------------+
| Patterson | Mary      | mpatterso@classicmodelcars.com |
+-----------+-----------+-------------------------------+
1 row in set (0.02 sec)
```

Update her email to the new email mary-patterso@classicmodelcars.com

```
UPDATE employees
SET email = 'mary-patterso@classicmodelcars.com'
WHERE employeeNumber = 1;
```

Execute the select query above again; you will see the email change to the new value.

```
+-----------+-----------+-----------------------------------+
| lastname  | firstname | email                             |
+-----------+-----------+-----------------------------------+
| Patterson | Mary      | mary-patterso@classicmodelcars.com |
+-----------+-----------+-----------------------------------+
1 row in set (0.00 sec)
```

MySQL DELETE Statement

*The **MySQL DELETE** statement not only allows you to delete record from one table but also multiple tables.* To remove all rows or records from a database table you use ***MySQL DELETE*** statement. The following illustrates **MySQL DELETE** statement:

```
DELETE [LOW_PRIORITY] [QUICK]
FROM table_name
WHERE conditions [ORDER BY …][LIMIT rows]
```

```
DELETE [LOW_PRIORITY] [QUICK] table_name[.*] [, table_name[.*] …]
FROM table-references
[WHERE where_definition]
```

```
DELETE [LOW_PRIORITY] [QUICK]
FROM table_name[.*] [, table_name[.*] …]
USING table-references
[WHERE where_definition]
```

Let's examine the MySQL DELETE statements in details as below:

- In the first form of MySQL DELETE statement, followed the DELETE FROM part is the table name where you want to delete records. The WHERE clause specifies condition to limit which rows you want to to remove. If a record meets WHERE condition, it will be removed from the database table permanently. If the WHERE clause is ignored in the MySQL DELETE statement, all rows of the table are deleted.
- The second form of MySQL DELETE statement, It deletes records from multiple tables which reference to other table.
- The third form of MySQL DELETE statement is quite similar to the second one except Using keyword is used instead of FROM keyword.
  Let's have a couple of examples of using MySQL DELETE statement in the sample database.

It is recommended that you make a copy of *employee* table before practicing with the MySQL DELETE statement.

Suppose you want to delete all employees in an office with *officeNumber* is 4, just execute the following query:

```
DELETE FROM employees
WHERE officeCode = 4
```

To delete all employees from all offices, just remove the WHERE condition as follows:

```
DELETE FROM employees
```

It will remove all rows from *employees* table.

If you want to delete all employee who work for office with *officecode* 1 and also that office. You can use the second form of MySQL DELETE statement to delete data from multiple tables as follows:

```
DELETE employees,offices
```

```
FROM employees,offices
WHERE employees.officeCode = offices.officeCode  AND offices.officeCode = 1
```

You can achieve the same above effect by using the third form of DELETE statement as below:

```
DELETE FROM employees,offices
USING employees,offices
WHERE employees.officeCode = offices.officeCode  AND offices.officeCode = 1;
```

MySQL REPLACE Statement

How to Use MySQL REPLACE to Insert or Update Data

MySQL REPLACE statement is a MySQL extension to SQL standard. MySQL REPLACE works like INSERT statements with the following rules:

- If the record being inserting does not exist, MySQL REPLACE will insert a new record.
- If the record being inserting exists, MySQL REPLACE will delete the old record first and then insert a new record with new values.

In order to use **MySQL REPLACE** you need to have at least both INSERT and DELETE privileges for the table.

The first form of **MySQL REPLACE** is similar to the INSERT statement except the keyword INSERT is replaced by the REPLACE keyword as follows:

```
REPLACE INTO table_name(column_name1,column_name2,…)
VALUES (value1, value2,…)
```

For example if you want to insert a new office into *offices* table, you use the following query:

```
REPLACE INTO offices(officecode,city)
VALUES(8,'San Jose');
```

Note that all values of missing columns in the REPLACE statement will be set to default value.

Now if you want to update the inserted office with San Mateo city, we can do it as follows:

```
REPLACE INTO offices(officecode,city)
VALUES(8,'San Mateo')
```

You see that two rows affected by the query above because the existing record is deleted and the new record is inserted.

The second form of MySQL REPLACE like UPDATE statement as follows:

```
REPLACE INTO table_name
SET column_name1 = value1 AND column2 = value2;
```

Note that there is no WHERE clause is specified in the REPLACE statement.

For example, if you want to update office in San Mateo with *officecode* 8 you can do it as follows:

```
REPLACE INTO offices
SET officecode = 8 and city='Surat';
```

The third form of MySQL REPLACE is similar to SQL INSERT INTO SELECT statement as follows:

```
REPLACE INTO table_name1(column_name1,column_name2,…)
SELECT coumn_name1, column_name2…
FROM table_name2
WHERE where _condition
```

Let's say if we want to copy the office with officecode 1, we can do it as follows:

```
REPLACE INTO offices(officecode,city,phone,addressline1,addressline2,state,country,postalcode)
SELECT (SELECT MAX(officecode) + 1 FROM offices),
city,phone,addressline1,addressline2,state,country,postalcode
FROM offices
WHERE officecode = 1;
```

There are several points you need to know before using MySQL REPLACE statement:

- If you are developing an application that potentially supports not only MySQL database, try to avoid using MySQL REPLACE because other databases may not support REPLACE statement. You can use the combination of INSERT and DELETE statement instead.
- If you are using MySQL REPLACE in the table which has triggers associate with it and if delete of duplicate key happens, the triggers are fired in the following orders:
    1. Before insert
    2. Before delete
    3. After delete
    4. After insert
- It is preferred using MySQL UPDATE statement to using MySQL REPLACE in case you just want to update data. Because MySQL UPDATE is a much faster performer than MySQL REPLACE.

Integration of PHP and MySQL

Before you can do anything with MySQL in PHP you must first establish a connection to your web host's MySQL database. For that you have set of functions which may able you to handle the MySQL database from PHP with the medium of server.

A set of functions use for integration among PHP and MySQL database server are shown under the following table:

PHP MySQL Functions

PHP: indicates the earliest version of PHP that supports the function.

Function	Description
mysql_affected_rows()	Returns the number of affected rows in the previous MySQL operation
mysql_change_user()	Deprecated. Changes the user of the current MySQL connection
mysql_client_encoding()	Returns the name of the character set for the current connection
mysql_close()	Closes a non-persistent MySQL connection
mysql_connect()	Opens a non-persistent MySQL connection
mysql_create_db()	Deprecated. Creates a new MySQL database. Use mysql_query() instead
mysql_data_seek()	Moves the record pointer
mysql_db_name()	Returns a database name from a call to mysql_list_dbs()
mysql_db_query()	Deprecated. Sends a MySQL query. Use mysql_select_db() and mysql_query() instead
mysql_drop_db()	Deprecated. Deletes a MySQL database. Use mysql_query() instead
mysql_errno()	Returns the error number of the last MySQL operation
mysql_error()	Returns the error description of the last MySQL operation
mysql_escape_string()	Deprecated. Escapes a string for use in a mysql_query. Use mysql_real_escape_string() instead
mysql_fetch_array()	Returns a row from a recordset as an associative array and/or a numeric array
mysql_fetch_assoc()	Returns a row from a recordset as an associative array
mysql_fetch_field()	Returns column info from a recordset as an object
mysql_fetch_lengths()	Returns the length of the contents of each field in a result row
mysql_fetch_object()	Returns a row from a recordset as an object
mysql_fetch_row()	Returns a row from a recordset as a numeric array
mysql_field_flags()	Returns the flags associated with a field in a recordset
mysql_field_len()	Returns the maximum length of a field in a recordset
mysql_field_name()	Returns the name of a field in a recordset

mysql_field_seek()	Moves the result pointer to a specified field
mysql_field_table()	Returns the name of the table the specified field is in
mysql_field_type()	Returns the type of a field in a recordset
mysql_free_result()	Free result memory
mysql_get_client_info()	Returns MySQL client info
mysql_get_host_info()	Returns MySQL host info
mysql_get_proto_info()	Returns MySQL protocol info
mysql_get_server_info()	Returns MySQL server info
mysql_info()	Returns information about the last query
mysql_insert_id()	Returns the AUTO_INCREMENT ID generated from the previous INSERT operation
mysql_list_dbs()	Lists available databases on a MySQL server
mysql_list_fields()	Deprecated. Lists MySQL table fields. Use mysql_query() instead
mysql_list_processes()	Lists MySQL processes
mysql_list_tables()	Deprecated. Lists tables in a MySQL database. Use mysql_query() instead
mysql_num_fields()	Returns the number of fields in a recordset
mysql_num_rows()	Returns the number of rows in a recordset
mysql_pconnect()	Opens a persistent MySQL connection
mysql_ping()	Pings a server connection or reconnects if there is no connection
mysql_query()	Executes a query on a MySQL database
mysql_real_escape_string()	Escapes a string for use in SQL statements
mysql_result()	Returns the value of a field in a recordset
mysql_select_db()	Sets the active MySQL database
mysql_stat()	Returns the current system status of the MySQL server
mysql_tablename()	Deprecated. Returns the table name of field. Use mysql_query() instead
mysql_thread_id()	Returns the current thread ID
mysql_unbuffered_query()	Executes a query on a MySQL database (without fetching / buffering the result)

We will see the required functions in order they use in the integration.

mysql_connect() Function

Usage

The mysql_connect() function opens a non-persistent MySQL connection.This function returns the connection on success, or FALSE and an error on failure. You can hide the error output by adding an '@' in front of the function name.

Syntax

mysql_connect(server,user,pwd,newlink,clientflag)

Parameter	Description
server	Optional. Specifies the server to connect to (can also include a port number. e.g. "hostname:port" or a path to a local socket for the localhost). Default value is "localhost:3306"
user	Optional. Specifies the username to log in with. Default value is the name of the user that owns the server process
pwd	Optional. Specifies the password to log in with. Default is ""
newlink	Optional. If a second call is made to mysql_connect() with the same arguments, no new connection will be established; instead, the identifier of the already opened connection will be returned
clientflag	Optional. Can be a combination of the following constants: • MYSQL_CLIENT_SSL - Use SSL encryption • MYSQL_CLIENT_COMPRESS - Use compression protocol • MYSQL_CLIENT_IGNORE_SPACE - Allow space after function names • MYSQL_CLIENT_INTERACTIVE - Allow interactive timeout seconds of inactivity before closing the connection

Example

```
<?php
      $con = mysql_connect("localhost","mysql_user","mysql_pwd");
      if (!$con)
        {
        die('Could not connect: ' . mysql_error());
        }
// some code
mysql_close($con);
?>
```

mysql_pconnect() Function

Definition and Usage

The mysql_pconnect() function opens a persistent MySQL connection.This function returns the connection on success, or FALSE and an error on failure. You can hide the error output by adding an '@' in front of the function name.

mysql_pconnect() is much like mysql_connect(), but with two major differences:

- This function will try to find a connection that's already open, with the same host, username and password. If one is found, this will be returned instead of opening a new connection
- The connection will not be closed when the execution of the script ends (mysql_close() will not close connection opened by mysql_pconnect()). It will stay open for future use

Syntax

mysql_pconnect(server,user,pwd,clientflag)

Parameter	Description
server	Optional. Specifies the server to connect to (can also include a port number. e.g. "hostname:port" or a path to a local socket for the localhost). Default value is "localhost:3306"
user	Optional. Specifies the username to log in with. Default value is the name of the user that owns the server process
pwd	Optional. Specifies the password to log in with. Default is ""
clientflag	Optional. Can be a combination of the following constants: • MYSQL_CLIENT_SSL - Use SSL encryption • MYSQL_CLIENT_COMPRESS - Use compression protocol • MYSQL_CLIENT_IGNORE_SPACE - Allow space after function names • MYSQL_CLIENT_INTERACTIVE - Allow interactive timeout seconds of inactivity before closing the connection

Example

```
<?php
$con = mysql_pconnect("localhost","mysql_user","mysql_pwd");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
?>
```

mysql_select_db() Function

Definition and Usage

The mysql_select_db() function sets the active MySQL database.This function returns TRUE on success, or FALSE on failure.

Syntax

mysql_select_db(database,connection)

Parameter	Description
database	Required. Specifies the database to select.
connection	Optional. Specifies the MySQL connection. If not specified, the last connection opened by mysql_connect() or mysql_pconnect() is used.

Example

```
<?php
$con = mysql_connect("localhost", "peter", "abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

$db_selected = mysql_select_db("test_db", $con);

if (!$db_selected)
  {
  die ("Can\'t use test_db : " . mysql_error());
  }

mysql_close($con);
?>
```

mysql_query() Function

Definition and Usage

The mysql_query() function executes a query on a MySQL database.This function returns the query handle for SELECT queries, TRUE/FALSE for other queries, or FALSE on failure.

Syntax

mysql_query(query,connection)

Parameter	Description
query	Required. Specifies the SQL query to send (should not end with a semicolon)
connection	Optional. Specifies the MySQL connection. If not specified, the last connection opened by mysql_connect() or mysql_pconnect() is used.

Note: This function fetches and buffers the recordset automatically. To run an unbuffered query, use mysql_unbuffered_query() instead.

Example 1

```
<?php
$con = mysql_connect("localhost","mysql_user","mysql_pwd");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

$sql = "SELECT * FROM Person";
mysql_query($sql,$con);

// some code

mysql_close($con);
?>
```

Example 2

Create a new database with the mysql_query() function:

```
<?php
$con = mysql_connect("localhost","mysql_user","mysql_pwd");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

$sql = "CREATE DATABASE my_db";
if (mysql_query($sql,$con))
  {
  echo "Database my_db created";
  }
else
  {
  echo "Error creating database: " . mysql_error();
  }
?>
```

mysql_result() Function

Definition and Usage

The mysql_result() function returns the value of a field in a recordset. This function returns the field value on success, or FALSE on failure.

Syntax

mysql_result(data,row,field)

Parameter	Description
data	Required. Specifies which result handle to use. The data pointer is the return from the mysql_query() function
row	Required. Specifies which row number to get. Row numbers start at 0
field	Optional. Specifies which field to get. Can be field offset, field name or table.fieldname. If this parameter is not defined mysql_result() gets the first field from the specified row

Note: This function is slower than mysql_fetch_row(), mysql_fetch_array(), mysql_fetch_assoc() and mysql_fetch_object().

Note: This function should not be used together with mysql_fetch_row(), mysql_fetch_array(), mysql_fetch_assoc() or mysql_fetch_object().

Example

```
<?php
$con = mysql_connect("localhost", "peter", "abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
$db_selected = mysql_select_db("test_db", $con);
$sql = "SELECT * from Person";
$result = mysql_query($sql,$con);
echo mysql_result($result,0);
mysql_close($con);
?>
```

The output of the code above could be:

```
Refsnes
```

mysql_num_rows() Function

Definition and Usage

The mysql_num_rows() function returns the number of rows in a recordset. This function returns FALSE on failure.

Syntax

mysql_num_rows(data)

Parameter	Description
data	Required. Specifies which data pointer to use. The data pointer is the result from the mysql_query() function

Example

```
<?php
$con = mysql_connect("localhost", "peter", "abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

$db_selected = mysql_select_db("test_db",$con);

$sql = "SELECT * FROM person";
$result = mysql_query($sql,$con);
echo mysql_num_rows($result);

mysql_close($con);
?>
```

The output of the code above could be:

```
4
```

mysql_num_fields() Function

Definition and Usage

The mysql_num_fields() function returns the number of fields in a recordset. This function returns FALSE on failure.

Syntax

mysql_num_fields(data)

Parameter	Description
Data	Required. Specifies which data pointer to use. The data pointer is the result from the mysql_query() function

Example

```
<?php
$con = mysql_connect("localhost", "peter", "abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
$db_selected = mysql_select_db("test_db",$con);
$sql = "SELECT * FROM person";
$result = mysql_query($sql,$con);
echo mysql_num_fields($result);
mysql_close($con);
?>
```

The output of the code above could be:

```
4
```

mysql_affected_rows() Function

Definition and Usage

The mysql_affected_rows() function returns the number of affected rows in the previous MySQL operation. This function returns the number of affected rows on success, or -1 if the last operation failed.

Syntax

mysql_affected_rows(connection)

Parameter	Description
connection	Optional. Specifies the MySQL connection. If not specified, the last connection opened by mysql_connect() or mysql_pconnect() is used.

Example

```
<?php
$con = mysql_connect("localhost","mysql_user","mysql_pwd");
if (!$con)
  {
  die("Could not connect: " . mysql_error());
  }
mysql_select_db("mydb");
mysql_query("DELETE FROM mytable WHERE id < 5");
$rc = mysql_affected_rows();
echo "Records deleted: " . $rc;
mysql_close($con);
?>
```

The output of the code above could be:

```
Records deleted: 4
```

mysql_fetch_array() Function

Definition and Usage

The mysql_fetch_array() function returns a row from a recordset as an associative array and/or a numeric array. This function gets a row from the mysql_query() function and returns an array on success, or FALSE on failure or when there are no more rows.

Syntax

mysql_fetch_array(data,array_type)

Parameter	Description
data	Required. Specifies which data pointer to use. The data pointer is the result from the mysql_query() function
array_type	Optional. Specifies what kind of array to return. Possible values: • MYSQL_ASSOC - Associative array • MYSQL_NUM - Numeric array • MYSQL_BOTH - Default. Both associative and numeric array

Note: After the data is retrieved, this function moves to the next row in the recordset. Each subsequent call to mysql_fetch_array() returns the next row in the recordset. Field names returned by this function are case-sensitive.

Example

```
<?php
$con = mysql_connect("localhost", "peter", "abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
$db_selected = mysql_select_db("test_db",$con);
$sql = "SELECT * from Person WHERE Lastname='Refsnes'";
$result = mysql_query($sql,$con);
print_r(mysql_fetch_array($result));
mysql_close($con);
?>
```

The output of the code above could be:

```
Array
(
[0] => Refsnes
[LastName] => Refsnes
[1] => Kai Jim
[FirstName] => Kai Jim
[2] => Taugata 2
[Address] => Taugata 2
[3] => 22
[Age] => 22
)
```

mysql_error() Function

Definition and Usage

The mysql_error() function returns the error description of the last MySQL operation. This function returns an empty string ("") if no error occurs.

Syntax

mysql_error(connection)

Parameter	Description
connection	Optional. Specifies the MySQL connection. If not specified, the last connection opened by mysql_connect() or mysql_pconnect() is used.

Example

In this example we will try to log on to a MySQL server with the wrong username and password:

```
<?php
$con = mysql_connect("localhost","wrong_user","wrong_pwd");
if (!$con)
  {
  die(mysql_error());
  }
mysql_close($con);
?>
```

The output of the code above could be:

```
Access denied for user 'wrong_user'@'localhost'
(using password: YES)
```

mysql_free_result() Function

Definition and Usage

The mysql_free_result() function frees memory used by a result handle. This function returns TRUE on success, or FALSE on failure.

Syntax

mysql_free_result(data)

Parameter	Description
data	Required. Specifies which result handle to free. The result handle is the return from the mysql_query() function

Tip: PHP automatically releases the memory used by result handles at the end of the script. However, this function can be useful when you want to free up memory between large queries.

Example

```
<?php
$con = mysql_connect("localhost", "peter", "abc123");
if (!$con)
{
die('Could not connect: ' . mysql_error());
}
$db_selected = mysql_select_db("test_db",$con);

$sql = "SELECT * from Person";
$result = mysql_query($sql,$con);
print_r(mysql_fetch_row($result));

// Free memory
mysql_free_result($result);

$sql = "SELECT * from Customers";
$result = mysql_query($sql,$con);
print_r(mysql_fetch_row($result));

mysql_close($con);
?>
```

mysql_fetch_assoc() Function

Definition and Usage

The mysql_fetch_assoc() function returns a row from a recordset as an associative array. This function gets a row from the mysql_query() function and returns an array on success, or FALSE on failure or when there are no more rows.

Syntax

mysql_fetch_assoc(data)

Parameter	Description
data	Required. Specifies which data pointer to use. The data pointer is the result from the mysql_query() function

Note: After the data is retrieved, this function moves to the next row in the recordset. Each subsequent call to mysql_fetch_assoc() returns the next row in the recordset. Field names returned by this function are case-sensitive.

Example

```
<?php
$con = mysql_connect("localhost", "peter", "abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

$db_selected = mysql_select_db("test_db",$con);
$sql = "SELECT * from Person WHERE Lastname='Refsnes'";
$result = mysql_query($sql,$con);
print_r(mysql_fetch_assoc($result));

mysql_close($con);
?>
```

The output of the code above could be:

```
Array
(
[LastName] => Refsnes
[FirstName] => Kai Jim
[Address] => Taugata 2
[Age] => 22
)
```

mysql_close() Function

Definition and Usage

The mysql_close() function closes a non-persistent MySQL connection. This function returns TRUE on success, or FALSE on failure.

Syntax

mysql_close(connection)

Parameter	Description
connection	Optional. Specifies the MySQL connection to close. If not specified, the last connection opened by mysql_connect() is used.

Tip: Using mysql_close() isn't required, as non-persistent connections are automatically closed at the end of the script.

Example

```
<?php
$con = mysql_connect("localhost","mysql_user","mysql_pwd");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
// some code
mysql_close($con);
?>
```

Exercise

1. How to create database in php
2. Explain MySQL table types in detail.
3. Differentiate mysql_connect() and mysql_pconnect()?
4. What are the necessary functions/steps to follow to fire any query?
5. Differentiate mysql_fetch_array() and mysql_fetch_assoc()?
6. Explain mysql_free_result() and mysql_close().
7. Write query to find second highest value from a table?

Part - 2

Appendix

A – HTML Tags

B – HTML Events

C – HTML Entities

HTML Tags

This section is having a quick review of all the HTML tags available..

HTML Basic Syntax:

- HTML Element names and attribute names are not case sensitive.
- HTML Documents start with a <!doctype...> statement, followed by a *header* and a text body all enclosed in <html>...</html>.
- HTML Header is enclosed in <head>....</head> tags.
- HTML Body is enclosed in <body>....</body> tags.
- HTML Comments are written as <!-- A comment -->.

HTML Basic Document:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 4.0//EN">
<html>
<head>
<title>Document Title like HTML Tutorial</title>
</head>
<body>
  Document Text with other tags will come here.
</body>
</html>
```

Header elements:

- **<head>** - Opening tag for the head of the document. The following optional tags can be placed inside the head.
- **<title>...</title>** -Document title (not part of the text), recommended maximum length 64 characters.
- **<link ...>** - Relationships for the document as a whole: common attributes are rel, rev, href.
- **<base href="url">** - Specifies the base URL of the document. This is used when dereferencing relative URLs in the page.
- **<base href="url" target="...">** - Specifies the base URL of the document. This is used when dereferencing relative URLs in the page. Also specifies the base target frame that all links will default to.
- **<meta ...>** - Embed meta-information as if given by the server: attributes http-equiv, name, content.
  <meta http-equiv="refresh" content="N" > - Same page will be reloaded automatically after N seconds.
  <meta http-equiv="refresh" content="N" url="http://www.example.com> - Same other page will refresh automatically after N seconds.
  <meta http-equiv="expires" content="Wed, 08 Aug 2007 01:21:00 GMT" > - Specifies an expiration date for the page so that it will be reloaded after a certain date.
  <meta http-equiv="keywords" content="keyword1, keyword2,..." > - Specifies various keywords available on the page and to be used by the search engine.

<meta http-equiv="description" content="A short description of the site" > - Specifies small description of the page and to be used by the search engine.

- **<style type="text/css" href="URL" />** - Specifies a CSS file to be used for the web page.
- **<script type="text/scripttype" href="URL" />** - Specifies a Javascript of VBscript file to be used for the web page.
- **<noscript> ... </noscript>** - Encloses anything you want displayed by browsers that do not support inline scripts. This goes inside the <script> tags.
- **</head>** - Closing tag for the head of the document.

Body Elements:

- **<body>...</body>** - Encloses the main body of the document.
- **<hn>...</hn>** - Makes the enclosed text a heading of various sizes where n is any number ranging from 1 to 6, and 1 creates the biggest heading while 6 creates the smallest.
- **<basefont size="n">** - Sets the default font properties for the entire page.
- **<isindex attributes>** - Displays a text box indicating the presence of a searchable index. Simply adding this tag will not create a searchable page. The server must be set up to support it.
- **<img src="URL" attributes>** - Places an inline image into the document.
- **<map attributes>...</map>** - Specifies a collection of hot spots that define a client-side image map. The <area> tag can be used inside to define the hot spots.
- **<area attributes>...</area>** - Specifies the shape and size of a hot spot to be used in the definition of a client-side image map. Used inside the <map> tag.
- **<marquee attributes>...</marquee>** - Places a scrolling text marquee into the document.
- **<applet attributes>...</applet>** - Inserts a Java applet in the HTML document. Any text placed between the opening and closing <applet> tags will be displayed by browsers that do not support JAVA.
- **<embed attributes>...</embed>** - Inserts an embedded multimedia object, such as a sound file or video, into the page.
- **...** - When used with the HREF attribute, the enclosed text and/or graphic becomes a link to another document or anchor. When used with the NAME attribute, the enclosed text and/or graphic becomes an anchor.
- **<ol attributes>...
** - Puts the enclosed items marked with <li>, in a numbered list.
- **<ul attributes>...
** - Puts the enclosed items marked with <li>, in a bulleted list.
- **<dl>...</dl>** - Creates a definition list. Within this container, <dt> specifies a definition term and <dd> specifies the definition.

Frame Elements:

- **<frameset attributes>...</frameset>** - Defines a set of frames that will make up the page. The <frame>, and <noframes> tags go inside this. The <frameset> tag is used instead of the <body> tag. You can, however, include a <body> tag inside the <noframes> tags for browsers that do not support frames.
- **<frame attributes />** - Defines a single frame within a frameset.
- **<iframe attributes>...</iframe>** - Defines a floating frame. Does not need to be placed within a frameset.

- **<noframes>…</noframes>** - Placed inside the <frameset>, anything between the beginning and ending of this tag is viewable only by browsers that do not support frames. This tag is used to create pages that are compatible with older browsers that do not support frames.

Table Elements:

- **<table attributes>…</table>** -Creates a table that can include any number of rows.
- **<caption attributes>…</caption>** -Specifies the caption of the table.
- **<tr attributes>…</tr>** - Specifies a table row. It can enclose the table heading and table data.
- **<th attributes>…</th>** - Specifies a table heading.
- **<td attributes>…</td>** - Specifies a table data cell.
- **<colgroup attributes />** - Specifies the properties of one or more columns. This tag generally goes right after the opening <table> tag.
- **<col attributes />** - Used with the <colgroup> tag, this specifies the properties of one column. This tag overrides any attributes specified in the <colgroup> tag that comes right before it.
- **<tbody>…</tbody>** - Encloses the body of your table. This tag is optional unless you are using the <thead> or <tfoot> tags. It used to separate the rows in the table from those in the header or footer.
- **<tfoot>…</tfoot>** - Encloses the table rows that are to be used as a footer. It is an optional tag and comes right after the ending <tbody> element.
- **<thead>…</thead>** - Encloses the table rows that are to be used as a header. It is an optional tag and comes before the opening <tbody> element.

Form Elements:

- **<from attributes>…</from>** - Specifies a form. Forms can be used to send user input to the server in the form of NAME/VALUE pairs.
- **<input attributes />** - Specifies a control or input area for a form, from which a NAME/VALUE pair will be returned to the server. It could be Checkbox, Raidobox, password, text, reset, submit, hidden and image.
- **<select attributes>…</select>** - Creates a drop-down list of items. The list items are defined by the <option> tags placed inside the opening and closing <select> tag.
- **<option value="…" />** - Specifies an item in the drop down list. Placed within the opening and closing <select> tags. Any text following the <option> tag is what the user will see in the list.
- **<textarea attributes>…</textarea>** - Creates a multi-lined text entry box. Any text placed in between the tags is used as the default text string that is displayed when the page is loaded.
- **<button attributes>…</button>** - allows you to have push buttons on forms that more closely resemble push buttons available in Windows and other applications.

Text Formatting Elements:

- **<address>…..< /address>** - Encloses the signature file of the author of the page. Text is displayed in italics.
- **<acronym>…..< /acronym>** - indicates an acronym in the text.

- **<b>…< /b>** - Boldfaces the enclosed text.
- **<big>…< /big>** - Makes the enclosed text one size larger.
- **<blink>…..< /blink>** - Makes the enclosed text blink continually.
- **<blockquote>…..< /blockquote>** - Encloses a long quote. Both the left and right margins are indented.
- **
** - Inserts a line break.
- ** …..< /center>** - Centers the enclosed elements.
- **<cite>…..< /cite>** - Encloses a citation such as the title of a book or paper.
- **<code>…..< /code>** - Encloses a sample of code. The text is rendered in small font.
- **<comment>…..< /comment>** - Encloses a comment. Text inside the tags is ignored unless it contains HTML code.
- **<del>…..< /del>** - To mark the document text that has been deleted since a previous version.
- **<dfn>…..< /dfn>** - Encloses a definition. Text inside the tags is formatted to look like a definition.
- ** …< /div>** - Specifies the alignment of the enclosed elements. Can be used to divide a document into sections that are aligned differently.
- **…< /em>** - Emphasis on the enclosed text (Italics).
- **<font attributes>…< /font>** - Sets the font properties for the enclosed text.
- **<fieldset attributes>…< /fieldset>** - Allows you can group related form fields, making your form easier to read and use.
- **<hr attributes />** - Inserts a horizontal line.
- **<i>…< /i>** - The enclosed text is italics.
- **<ins>…< /ins>** - To mark parts of a document that have been added since the document's last version.
- **<label>…< /label>** - Allows you to lable a tag.
- **<kbd>…< /kbd>** - Specifies text to be entered at the keyboard. Text is rendered as bold and fixed-width.
- **<p attributes>…< /p>** - Designates the enclosed text as a plain paragraph.
- **<q>…</q>** - acts much the same as the <blockquote> tag, but applies to shorter quoted sections, ones that don't need paragraph breaks.
- **<pre>…..< /pre>** - Displays text in fixed-width type without collapsing spaces.
- **<s>…..< /s>** - Displays text with a line through it. The <strike> tag does exactly the same.
- **<samp>…< /samp>** - Indicates sample output from a form or program. Text is rendered in small font.
- **<small>…< /small>** - Makes the enclosed text one size smaller.
- **<spacer attributes>…< /spacer>** - Inserts blocks of spaces into HTML documents.
- **…< /strong>** - Stronger emphasis on the enclosed text.
- **<sub>…< /sub>** - Renders the enclosed text in subscript.
- **<sup>…< /sup>** - Renders the enclosed text in superscript.
- **<tt>…< /tt>** - The enclosed text is typewriter font.
- **<u>…< /u>** - The enclosed text in underlined.
- **<var>…< /var>** - Specifies a variable. Text is rendered in small fixed-width type.
- **<wbr>** - Causes text enclosed by the NOBR tags to wrap only if necessary.

HTML Events

When a user visit your website, they do things like click on text and images and given links, hover over things etc. These are examples of what JavaScript calls events.

We can write our event handlers in Javascript of vbscript and can specify these event handlers as a value of event tag attribute. The HTML 4.01 specification defines 19 event attributes as listed below:

<body> and <frameset> Level Events:

There are only two attributes which can be used to trigger any javascript or vbscript code when there is any event occurs on document level.

Attribute	Value	Description
Onload	script	Script runs when a HTML document loads
onunload	script	Script runs when a HTML document unloads

NOTE: Here script refer to any VBScript or JavaScript function or piece of code.

<form> Level Events:

There are following six attributes which can be used to trigger any javascript or vbscript code when there is any event occurs on form level.

Attribute	Value	Description
onchange	script	Script runs when the element changes
onsubmit	script	Script runs when the form is submitted
Onreset	script	Script runs when the form is reset
onselect	script	Script runs when the element is selected
Onblur	script	Script runs when the element loses focus
onfocus	script	Script runs when the element gets focus

Keyboard Events

There are following three events which are generated by keyboard. These events are not valid in base, bdo, br, frame, frameset, head, html, iframe, meta, param, script, style, and title elements.

Attribute	Value	Description
onkeydown	script	Script runs when key is pressed
onkeypress	script	Script runs when key is pressed and released
onkeyup	script	Script runs when key is released

Other Events:

There following other 7 events which are generated by mouse when it comes in contact of any HTML tag. These events are not valid in base, bdo, br, frame, frameset, head, html, iframe, meta, param, script, style, title elements.

Attribute	Value	Description
Onclick	script	Script runs when a mouse click
ondblclick	script	Script runs when a mouse double-click
onmousedown	script	Script runs when mouse button is pressed
onmousemove	script	Script runs when mouse pointer moves
onmouseout	script	Script runs when mouse pointer moves out of an element
onmouseover	script	Script runs when mouse pointer moves over an element
onmouseup	script	Script runs when mouse button is released

HTML and XHTML Entities

Characters with special meaning in HTML and XHTML

Some characters are reserved in HTML XHTML. For example, you cannot use the greater than and less than signs or angle brackets within your text because the browser could mistake them for markup.

HTML and XHTML processors must support the five special characters listed in the table that follows.

Symbol	Description	Entity Name	Number Code
"	quotation mark	&quot;	&#34;
'	apostrophe	&apos;	&#39;
&	ampersand	&amp;	&#38;
<	less-than	&lt;	&#60;
>	greater-than	&gt;	&#62;

To write an element and attribute into your page so that the code is shown to the user rather than being processed by the browser (for example as) you would write:

```
&div id=&quot;character&quot;&gt;
```

There is also a long list of special characters that HTML 4.0.aware processors should support. In order for these to appear in your document, you can use either the numerical code or the entity name. For example, to insert a copyright symbol you could use either of the following:

```
&copy; 2007
or
&#169; 2007
```

ISO 8859-1 Character set is the most widely used character set. A complete reference of ISO 885901 character set is given below.

ISO 8859-1 Symbol Entities

Result	Description	Entity Name	Number Code
	non-breaking space		
¡	inverted exclamation mark	&iexcl;	&#161;
¤	currency	&curren;	&#164;
¢	cent	&cent;	&#162;
£	pound	&pound;	&#163;
¥	yen	&yen;	&#165;
¦	broken vertical bar	&brvbar;	&#166;
§	section	&sect;	&#167;

¨	spacing diaeresis	¨	¨
©	copyright	©	©
ª	feminine ordinal indicator	ª	ª
«	angle quotation mark (left)	«	«
¬	negation	¬	¬
	soft hyphen		
®	registered trademark	®	®
™	trademark	™	™
¯	spacing macron	¯	¯
°	degree	°	°
±	plus-or-minus	±	±
²	superscript 2	²	²
³	superscript 3	³	³
´	spacing acute	´	´
µ	micro	µ	µ
¶	paragraph	¶	¶
·	middle dot	·	·
¸	spacing cedilla	¸	¸
¹	superscript 1	¹	¹
º	masculine ordinal indicator	º	º
»	angle quotation mark (right)	»	»
¼	fraction 1/4	¼	¼
½	fraction 1/2	½	½
¾	fraction 3/4	¾	¾
¿	inverted question mark	¿	¿
×	multiplication	×	×
÷	division	÷	÷

ISO 8859-1 Character Entities

Result	Description	Entity Name	Number Code
À	capital a, grave accent	À	À

Á	capital a, acute accent	Á	Á
Â	capital a, circumflex accent	Â	Â
Ã	capital a, tilde	Ã	Ã
Ä	capital a, umlaut mark	Ä	Ä
Å	capital a, ring	Å	Å
Æ	capital ae	Æ	Æ
Ç	capital c, cedilla	Ç	Ç
È	capital e, grave accent	È	È
É	capital e, acute accent	É	É
Ê	capital e, circumflex accent	Ê	Ê
Ë	capital e, umlaut mark	Ë	Ë
Ì	capital i, grave accent	Ì	Ì
Í	capital i, acute accent	Í	Í
Î	capital i, circumflex accent	Î	Î
Ï	capital i, umlaut mark	Ï	Ï
Ð	capital eth, Icelandic	Ð	Ð
Ñ	capital n, tilde	Ñ	Ñ
Ò	capital o, grave accent	Ò	Ò
Ó	capital o, acute accent	Ó	Ó
Ô	capital o, circumflex accent	Ô	Ô
Õ	capital o, tilde	Õ	Õ
Ö	capital o, umlaut mark	Ö	Ö
Ø	capital o, slash	Ø	Ø
Ù	capital u, grave accent	Ù	Ù
Ú	capital u, acute accent	Ú	Ú
Û	capital u, circumflex accent	Û	Û
Ü	capital u, umlaut mark	Ü	Ü

Ý	capital y, acute accent	Ý	Ý
Þ	capital THORN, Icelandic	Þ	Þ
ß	small sharp s, German	ß	ß
à	small a, grave accent	à	à
á	small a, acute accent	á	á
â	small a, circumflex accent	â	â
ã	small a, tilde	ã	ã
ä	small a, umlaut mark	ä	ä
å	small a, ring	å	å
æ	small ae	æ	æ
ç	small c, cedilla	ç	ç
è	small e, grave accent	è	è
é	small e, acute accent	é	é
ê	small e, circumflex accent	ê	ê
ë	small e, umlaut mark	ë	ë
ì	small i, grave accent	ì	ì
í	small i, acute accent	í	í
î	small i, circumflex accent	î	î
ï	small i, umlaut mark	ï	ï
ð	small eth, Icelandic	ð	ð
ñ	small n, tilde	ñ	ñ
ò	small o, grave accent	ò	ò
ó	small o, acute accent	ó	ó
ô	small o, circumflex accent	ô	ô
õ	small o, tilde	õ	õ
ö	small o, umlaut mark	ö	ö
ø	small o, slash	ø	ø

ù	small u, grave accent	ù	ù
ú	small u, acute accent	ú	ú
û	small u, circumflex accent	û	û
ü	small u, umlaut mark	ü	ü
ý	small y, acute accent	ý	ý
þ	small thorn, Icelandic	þ	þ
ÿ	small y, umlaut mark	ÿ	ÿ

Other Entities Supported by HTML Browser:

Result	Description	Entity Name	Number Code
Œ	capital ligature OE	Œ	Œ
œ	small ligature oe	œ	œ
Š	capital S with caron	Š	Š
š	small S with caron	š	š
Ÿ	capital Y with diaeres	Ÿ	Ÿ
ˆ	modifier letter circumflex accent	ˆ	ˆ
˜	small tilde	˜	˜
	en space		
	em space		
	thin space		
	zero width non-joiner	‌	‌
	zero width joiner	‍	‍
	left-to-right mark	‎	‎
	right-to-left mark	‏	‏
–	en dash	–	–
—	em dash	—	—
‘	left single quotation mark	‘	‘
’	right single quotation mark	’	’
,	single low-9 quotation mark	‚	‚
“	left double quotation mark	“	“
”	right double quotation mark	”	”

„	double low-9 quotation mark	„	„
†	dagger	†	†
‡	double dagger	‡	‡
…	horizontal ellipsis	…	…
‰	per mille	‰	‰
‹	single left-pointing angle quotation	‹	‹
›	single right-pointing angle quotation	›	›
€	euro	€	€

Bibliography

List of Websites

- www. www.php.net
- www.mysql.com
- www.apache.com
- www.w3schools.com
- www.tizag.com
- www.quackit.com
- www.tutorialspoint.com
- www.mysqltutorial.org

List of Books

- PHP Cookbook
- PHP – MySQL Bible
- Learning PHP - MySQL