PHP and MySQL Programming/Print version

PHP and MySQL Programming

Contents

Introduction to PHP

Welcome to PHP

PHP started off as a collection of Perl scripts used to aid in the maintaining of personal home pages. Hence, it originally stood for "Personal Home Page." However, the language has grown into a huge open source initiative, used by thousands of amateur and professional programmers world wide. It now is officially known as "PHP: Hypertext Preprocessor," which is a recursive acronym. (For some strange reason, open source programmers seem to love recursive acronyms...)

The reason it is known as a preprocessor and not merely as a processor, is due to the fact that the php files are sent by the webserver to the PHP preprocessor to be processed. the resulting HTML (Hyper Text Markup Language), is then sent across the internet to the web browser requesting the php page.

PHP has traditionally been used for adding dynamic content onto previously static webpages; however, due to the rapid development of the language, it is now used for writing command line tools, and now has GTK bindings allowing the programmer to write GUI applications!

Since the beginning of version 5 of PHP, with the inclusion of Zend Engine II as PHP's foundation, PHP now can be used to write Object Oriented applications.

You can download PHP free of charge (as it/is an Open Source programming language) from http://www.php.net

Syntax Overview

PHP tags:

<?php [code] ?>

- Enclose PHP code
- Embedded in normal HTML code
- Within PHP tags, statements are separated by a ; (generally also followed by a new line).

Example:

```
<?php
print "Hello World\n";
$date = date("Y-m-d H:i:s");
print "The date and time at the moment is $date";
?>
```

Commenting

//

Comments out one line

Example:

```
echo "Hello"; // Everything from the hash is commented out
```

Example:

```
// This entire line is commented out
```

#

Same function as //

/* (text) */

Comments out everything between the /* and the */

Example:

```
/*All of this is
```

```
commented out.
Even this line!*/
```

Variables

Variables in PHP are denoted by the $ prefix.

Example:

```
$a = "Hello World"; # this assigns the string "Hello World" to $a.
$b = "$a, I'm Ralfe"; # this assigns "Hello World, I'm Ralfe" to $b.
$b = $a.", I'm Ralfe"; # exactly the same as the previous example.
```

PHP supports dynamic variables.

Example:

```
$c = “response”;

$$c = “Hello Ralfe”; # this assigns “Hello Ralfe” to $response.
```

PHP variables do not have to be declared ahead of time, nor do they require a type definition. PHP handles all data type conversions.

- Example:

```
$a = 4;

$b = 12;

print “The value of a is $a.”; # Uses a as a String.

$c = $a + $b; # $a is now used as an Integer again.
```

PHP supports Boolean variables, which can be assigned either a one or a zero, or a true or false.

Example:

```
$a = true;

$b = 1;  # $a and $b are equal, though not identical.

$c = false;

$d = 0; # $c and $d are equal, though not identical.
```

Helpful Definitions:

Equal vs Identical

Equal
: values are equal in value, but may be of differing types. Uses == comparison operator. E.g. false is equal to 0.

Identical
: values are equal in value and of the same type. Uses === comparison operator. E.g. false is identical

only to false, it is not identical to 0.

Operators

Arithmetic Operators

Example:

```
$a = 4;
$b = 2;
$a + $b = 6;  // Addition
$a - $b = 2;  // Subtraction
$a * $b = 8;  // Multiplication
$a / $b = 2;  // Division
$a % $b = 0;  // Modulus
$a++;  // Increment
$a--;  // Decrement
```

Assignment Operators

Example:

```
$a = 4;
$b = $a;
// $b = 4;
```

Comparison Operators

Example:

```
$a == $b // test if two values are equal
$a === $b // test if two values are identical
$a != $b // test if two values are not equal
$a !== $b // test if two values are not identical
$a < $b // test if the first value is less than the second
$a > $b // test if the first value is greater than the second
$a <= $b // test if the first value is less than or equal to the second
$a >= $b // test if the first value is greater than or equal to the second
```

Concatenation

Example:

```
$a = "Fill the halls ";

$b = "with poisoned ivy...";

$c = $a . $b;  # the '.' operator concatenates two variables.

// $c = "Fill the halls with poisoned ivy...";
```

Arrays

PHP supports both numerically indexed arrays as well as associative arrays.

Example:

```
$a = array(1, 2, 3, 4);

// $a[0] = 1;

// $a[1] = 2;

// $a[2] = 3;

// $a[3] = 4;

$b = array("name" => "Fred", "age" => 30);

// $b['name'] = "Fred";

// $b['age'] = 30;
```

Decision and Loop Statements

If ... else statement

Example:

```
$a = 1;
$b = 10;

if ($a > $b) {
   echo "a is greater than b";
}
else if ($a == $b) {
   echo "a is equal to b";
}
else {
   echo "a is not greater than b";
}

// OUTPUT:
// a is not greater than b
```

Switch statement

Example:

```
$a = 100;

switch($a) {
   case(10):
      echo "The value is 10";
      break;
```

```
   case(100):
      echo "The value is 100";
      break;

   case(1000):
      echo "The value is 1000";
      break;

   default:
      echo "Are you sure you entered in a valid number?";
}

// OUTPUT:
// The value is 100
```

For statement

Example:

```
for ($i = 0; $i < 10; $i++) {
   # initialize $i ; while condition ; increment statement
   echo $i;
}

// OUTPUT:
// 0123456789
```

Foreach statement

Example:

```
$a = array(1, 2, 3, 4, 5);

foreach ($a as $val){
   echo $val;
}

// OUTPUT:
// 12345
```

While statement

Example:

```
while ($row = mysql_fetch_row($result)){
   print $row[0];
}
```

Do ... while statement

Example:

```
$i = 0;
# Note that it might seem that $i will
do{
   # never be printed to the screen, but
   print $i;
   # a DO WHILE loop always executes at
} while ($i >0);
# least once!
```

Functions

Example:

```
function square_number($number) {
   return ($number * $number);
}

$answer = square_number(10);
echo "The answer is {$answer}";

// OUTPUT:
// The answer is 100
```

Classes

Example:

```
class dog {
   var $name;

   function __construct($name){
      $this->name = $name;
   }

   function bark(){
      echo "Woof! Woof!";
   }

   function who_am_i() {
      echo "My name is {$this->name}, and I am a dog";
   }
}

$the_dog = new dog("John");
$the_dog->who_am_i();

// OUTPUT:
// My name is John, and I am a dog
```

As of PHP 5.3.3, functions with the same name as the class will no longer act as a constructor. Versions prior to 5.3.3 the

```
function __construct($name)
```

line could be replaced with

```
function dog($name)
```

to achieve the same effect. [1]

File Handling

As with most programming languages, file handling is very important for storing and retrieving a wide variety of data.

Opening and Closing Files

To open a file, we use the fopen() function. This function takes in two parameters. Firstly, a String containing the name of the file, and secondly, the mode with which to open the file with. It returns a filehandler. e.g.:

```
$filename = "text.txt";
$f = fopen($filename, 'r');
```

The above code assigns the file name of "text.txt" to a variable called $filename. Then it uses the `fopen` function to open up "text.txt" with 'r' mode (meaning "read mode") and returns a filehandler to $f.

To close a file, we simply call the fclose() function, and pass it the filehandler. e.g.:

```
fclose($f);
```

Including a File

It is very easy, in PHP, to include code (be it PHP or HTML) from one file into another. This is achieved by means of the include() function.

```
include ("objects.php");
```

In the above example, the PHP code will look within the current directory of the PHP file calling the include function, it will then open the objects.php file, and include its content at of that file at the position of the include() function.

Writing to a File

The procedure for writing to a file is:

1. Create a file handler.
2. Use the file handler to open up an input stream from a file.
3. Write Data to the file handler.
4. Close the file handler.

Here is an example of writing two lines of text to a file:

```
$filename = "file.txt";
$f = fopen($filename, 'w');
fputs($f, "Bonjour Madame.\n"); // Hello Madam.
fputs($f, "Bonjour Monsieur.\n"); // Hello Sir.
fclose($f);
```

fputs() is used to write data to the file. It takes 2 variables, firstly, the file handler, and secondly, a String to write to the file handler.

Reading from a File

Reading from a file goes along much the same format as writing to one. The function used to read from a file is fgets() function. Which takes in two variables. Firstly, the filehandler from which to read, and secondly,

the amount of data to retrieve.

```
$filename = "text.txt";
$f = fopen($filename, 'r');
$line = fgets($f, 1024);
print $line;
print $total_contents;
fclose($f);
```

The above code opens up a file, reads the first line of the file, and then displays it. If we wanted to rather read the entire contents of the file into a variable, we could replace the $line=fgets() line with:

```
 $total_contents = fread($f, filesize($filename));
```

Database Connectivity

You can only do so much with storing information in files. When you need to store large amounts of data, and perform intensive number crunching on that data, there is nothing better than a good SQL database. In this section, we'll discuss connecting PHP to a MySQL database and perform queries and retrieve data back from the database.

Opening a Connection to a MySQL Database

The first thing that we need to do before we can interact with any database, is to open up a connection to that database server. This is done by using the `mysql_connect()` function, which returns a database handler, much like a file handler when dealing with files. The database handler is then used to selecting the active database to use.

Here is the code to setup a connection to the database server, and to select a database to use:

```
$db = "database1";
$link = mysql_connect("localhost", "username", "password");
mysql_select_db($db, $link);
```

Creating a Query

Once a connection to a database has been made, you will inevitably want to perform a database query. To create a query on the selected database, we use the `mysql_query()` function. If you use a `SELECT` query, then the data returned from that query will be passed to the `mysql_query()` function, which will in turn return it to a variable which you can specify. In the following example, two queries are made, the first does not return any data, and the second does.

```
// A Query without any returned data
mysql_query ("INSERT INTO `table1` ('val1', 'val2')");
// A Query with returned data
$query = mysql_query("SELECT * FROM `table1`");
```

Retrieving data from a SELECT Query

There are many methods for retrieving data from a `SELECT` query.

If we take, for example, the following code:

```
# --- Connect To Database ---
$db = "db1";
$link = mysql_connect("localhost", "user", "pass");
mysql_select_db($db, $link);

# --- Select Info from Database ---
$result = mysql_query ("SELECT val1, val2 FROM tbl1");
```

To now retrieve the data from the `$result` variable, we can use one of many methods. The recommended method, however, is to sequentially go through each row of the table, storing it into a one-dimensional array. We do this by using the `mysql_fetch_row()` function, passing it the variable into which the result is stored. Here is a simple example:

```
while ($row = mysql_fetch_row($result)){
   foreach ($row as $field) {
      print "$field . ";
   }
   print "";
}
```

This will simply output the result in table-like format.

Here is another example of using this method:

```
$counter = 0;
while ($row = mysql_fetch_row($result)){
   $val1[$counter] = $row[0];
   $val2[$counter] = $row[1];
   $counter++;
}
$numRows = $counter;
```

The above example, simply splits the results up into multiple one-dimensional arrays, for easy manipulation.

Closing a Database Connection

It is not always necessary to close a connection when you are finished, but it is advised. It is, however, necessary to close the connection to the database if you want to open up a new connection to a different database.

To close a connection to a database, we use the `mysql_close()` function, as follows:

```
mysql_close();
```

Error Handling

It is useful when debugging, and even when you just want to make sure that a database does not behave unexpectedly. Once a query has been created via the `mysql_query()` function, any error messages generated will be stored in the `mysql_error()` function. Here is a sample code snippet to display a error

message. However, when there is no error messages, a blank string is returned.

```
print mysql_error();
```

Session Handling

HTML and PHP are "stateless" languages. Meaning that they are incapable of retaining a state between pages. To get around this serious limitation, we use sessions. With sessions, session variables are stored in a file on the web server and so are accessible across multiple pages.

Starting a Session

Before we can start using session variables, we need to start a session. This needs to be done on every page that makes use of session variables. It is important to note that a session must be started **before anything is outputted**. Here is the code to start a session:

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

Please make sure that there are NO SPACES before the "<?php" (php starting tag), as that will return errors!

It is worth noting, that the way in which the server distinguishes between multiple sessions being implemented on it simultaneously is by session ID's. When a unique session is created, it is assigned a Session ID, which the browser retains, and is used to let the we server know which session to use.

Writing Session Variables

Once a session is created, variables may be stored in the `$_SESSION[]` array variable. Here is an example:

```
session_start();
$_SESSION['user_name'] = "Administration";
```

Retrieving Session Variables

Once a session is created, and variables are stored, they may be retrieved from the `$_SESSION[]` array. Here is an example:

```
session_start();
if (isset($_SESSION['user_name'])){
   $user_name = $_SESSION['user_name'];
   print $user_name;
}
```

The above example starts a session, then checks if the session variable 'user_name' has been created (with the `isset()` function), if it has, then it assigns its value to `$user_name`, and prints it to the screen.

Destroying a Session

To destroy a session, we use the `session_destroy()` function. This is useful if we want to, for example, log a user off of a web-application. Thus we would use the following code:

```
session_start();
session_destroy();
```

Note that we first need to start the session before we destroy it. This is because the `session_destroy()` function destroys the currently active session.

Command Line Programming

PHP was traditionally used to help web administrators automate various mundane tasks and to add dynamism to the web pages. However, PHP (in its modern incarnation) is capable of much more. It is now being used in ways traditional programming languages are used. It is now possible to remove PHP applications from the constraints of the Internet server. Now, PHP is being used to write GUI applications as well as command line applications. We shall now look at writing PHP applications for the command line.

Getting the PHP-CLI

If you are running Linux, you should get the corresponding package for the PHP-CLI (http://www.php-cli.com). In Debian, it is just a matter of typing:

```
$> apt-get install php5-cli
```

In Windows, the CLI should be included with the PHP files. Refer to http://www.php-cli.com/ and http://www.php.net for more help.

Initial Considerations

We start off our command line applications with the following line:

```
#!/bin/php
```

Note: You do not include this line in Windows!

This should be the first line of the file, as it tells Linux which interpreter to use for the file.

The next thing you should do, is to make the file executable. This is done by the following Unix command:

```
chmod +x filename.php
```

Getting Keyboard Input

In order for us to retrieve keyboard input from the command line, we will need to use a little trick. Basically, we create a file handler to a special file called the Standard Input.

Here is the code to create this file handler:

```
$stdin = fopen("php://stdin", 'r');
```

We can now use `$stdin` just as if it were a normal file that we had opened for reading. (We obviously could not write to it, as it is Standard INPUT)

Here is a useful function to get user input:

```
function getinput(){
   $stdin = fopen("php://stdin", 'r');
   $input = fgets($stdin, 1024);
   $input = trim($input);
   fclose($stdin);
   return $input;
}
```

Output Formatting

The main consideration to make when formatting output in command line programming as opposed to generating HTML output is that a new line is not created by a
tag, in fact, any HTML tags you put into the output, will simply be displayed as plain text.

In order to create a new line in the command line, we need to use a control character, \n . Another useful function when doing command line programming is this:

```
function output($message){
   print $message."\n";
}
```

External Resources and Links

PHP CLI (http://www.php-cli.com) - ALL about running PHP script from command line: tutorials, options, examples, PHP CLI and PHP CGI difference.

Form Handling

In order to get user input without using the command line, then you will use HTML forms. PHP has various functions and techniques for handling the input from HTML forms. In specific, we will use the special array variables `$_GET[]` and `$_POST[]`.

HTML Forms

Before we start using these special variables, lets have a look at how to set up the HTML forms to interact

with PHP. Here is a simple example form:

```
<html>
<head><title>HTML Form Example</title></head>
<body>
   <form method="POST" action="form_handler.php">
     username: <input type="text" name="username"> <br>
     password: <input type="password" name="password"> <br>
     <input type="submit" value="GO">
   </form>
</body>
</html>
```

The above example sets up a form with a `method` and an `action` attribute. The `method` defines how the data from the form is to be transferred to the page defined by `action`. The `method` can either be *GET* or *POST*. If *GET* is used, then all the form elements will be appended to the url of the target page (specified by the `action` attribute), in url encoded format (`target?attrib1=val1&attrib2=val2&attrib3=val3` etc...) These attributes are easily accessed in PHP by using the `$_GET[]` array.

If the *POST* method is specified, then the data will be put into a special server variable, and accessed in PHP by using the built-in `$_POST[]` array.

Note that in the HTML code, the `name` attribute of the INPUT tags are used as the reference strings in the `$_GET` and `$_POST` arrays, thus it is very important to always use unique values for the `name` attribute.

GET

To retrieve `GET` variables in the address url, we use the following method:

```
$username = $_GET['username'];
$password = $_GET['password'];
echo "Username: $username \n Password: $password \n";
```

Please note that `GET` is not good to use for this example, because you will generally not want username and password details being displayed in the URL. Rather use POST for login details and authentication.

POST

To retrieve `POST` variables, we use the following method:

```
$username = $_POST['username'];
$password = $_POST['password'];
print "Username: $username \n Password: $password \n";
```

Note that when `POST` is used, the posted data is completely invisible to the user.

Further Reading

- HyperText Markup Language/Forms

Introduction to MySQL

Introduction to MySQL

MySQL, the most popular open source SQL database, is developed and provided by MySQL AB, a commercial company which gets its income from providing services around the MySQL database. The MySQL software delivers a very fast, multi-threaded, multi-user, and robust SQL database server, intended for mission-critical, heavy-load production systems as well as for embedding into mass-deployed software.

The official way to pronounce MySQL is "My Ess Que Ell" (though in practice many users pronounce it as "my sequel"). Michael ("Monty") Widenius, the creator of MySQL, is unsure where the name originally came from, though he says that it may have come from his daughter My. Two more of Widenius's children, Max and Maria, have also lent their names to MySQL products (MaxDB and the Maria table type respectively).

MySQL Licenses

The MySQL software is released under the GNU General Public License (GPL), which is probably the best known open source license. The software is provided "as is" and is without any warranty.

The MySQL software has Dual Licensing, which means you can use the MySQL software free of charge under the GPL. You can also purchase commercial MySQL licenses from MySQL AB if you do not wish to be bound by the terms of the GPL.

You need to purchase commercial non-GPL MySQL licenses:

- If you distribute MySQL Software with your non open source software.
- If you want warranty from MySQL AB for the MySQL software.
- If you want to support MySQL development.

Creating a Database

Introduction

MySQL is a relational database. In a relational database, data is stored as a collection of fields called a row. Like fields, rows are in turn collectively stored as a table. Tables are stored collectively in a database. One MySQL server can store multiple databases, and have users who have various rights to each database.

Nowadays, MySQL is the one of the most popular relational database among others as it is simple, free (open source), easy to maintain, and cost effective.

In order to store data into MySQL, we will need to prepare a space in MySQL for a database.

SQL Code

The code for creating such a database in SQL (Structured Query Language) is a simple query:

```
mysql> CREATE DATABASE `database_name`;
```

(Without the ' ' single quotes around the database name)

Using the Database

After a database has been created, you have to tell MySQL that to make it the active database before you can start running further queries on it. You do this as follows:

```
mysql> USE `database_name`;
```

Creating a Table

Before creating a table, please read the previous section on Creating a Database.

A Table resides inside a database. Tables contain rows, which are made up of a collection of common fields or columns. Here is a sample output for a SELECT * query:

```
mysql> SELECT * FROM `books`;
+------------+--------------+------------------------------+------+
| ISBN       | Author       | Title                        | Year |
+------------+--------------+------------------------------+------+
| 1234567890 | Poisson, R.W | Programming PHP and MySQL    | 2006 |
| 5946253158 | Wilson, M    | Java Secrets                 | 2005 |
| 8529637410 | Moritz, R    | C from Beginners to Advanced | 2001 |
+------------+--------------+------------------------------+------+
```

As you can see, We have rows (horizontal collection of fields), as well as columns (the vertical attributes and values).

Creating a Table

The SQL code for creating a table is as follows:

```
mysql> CREATE TABLE `table_name` (
       `field1` type NOT NULL|NULL default 'default_value',
       `field2` type NOT NULL|NULL default 'default_value',
       ...
       );
```

Example

Here is an example of creating a table called `books`:

```
mysql> CREATE TABLE `books` (
       `ISBN` varchar(35) NOT NULL default ,
       `Author` varchar(50) NOT NULL default ,
       `Title` varchar(255) NOT NULL default ,
```

```
      `Year` int(11) NOT NULL default '2000'
      );
```

Getting Information about Tables

To get a list of tables:

```
mysql> SHOW TABLES;
```

Which produces the following output:

```
+-------------------+
| Tables_in_library |
+-------------------+
| books             |
+-------------------+
1 row in set (0.19 sec)
```

To show the CREATE query used to create the table:

```
mysql> SHOW CREATE TABLE `books`;
```

Which produces the following output:

```
+-------+-------------------------------------------+
| Table | Create Table
+-------+-------------------------------------------+
| books | CREATE TABLE `books` (
  `ISBN` varchar(35) NOT NULL default ,
  `Author` varchar(50) NOT NULL default ,
  `Title` varchar(255) NOT NULL default ,
  `Year` int(11) NOT NULL default '2000'
) TYPE=MyISAM |
+-------+-------------------------------------------+
1 row in set (0.05 sec)
```

And then to show the same information, in a tabulated format:

```
mysql> DESCRIBE `books`;
```

Which produces the following output:

```
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| ISBN   | varchar(35)  |      |     |         |       |
| Author | varchar(50)  |      |     |         |       |
| Title  | varchar(255) |      |     |         |       |
| Year   | int(11)      |      |     | 2000    |       |
+--------+--------------+------+-----+---------+-------+
4 rows in set (0.18 sec)
```

Basic Queries

Introduction

Here are a few basic queries to get you going. For more detailed information, have a look at the online documentation on the MySQL website http://dev.mysql.com/doc/refman/5.1/en/index.html .

SELECT

The SELECT query is the standard means by which to retrieve information from the database. The syntax for the SELECT query is as follows:

```
mysql> SELECT field1, field2, ...
       FROM table_name
       [WHERE condition1
            AND condition2
            ...
       [ORDER BY field1, field2, etc...]
       [LIMIT number]];
```

Example:

```
mysql> SELECT * FROM books;

mysql> SELECT Title, Author
       FROM books
       WHERE Year > 2001
       AND Year < 2006
       ORDER BY Author ASC
       LIMIT 100;
```

INSERT

The INSERT statement is used to insert data into a particular table of a database. The Syntax is as follows:

```
mysql> INSERT INTO table_name
       (field1, field2, etc...)
       VALUES ('val1', 'val2', etc...);
```

Example:

```
mysql> INSERT INTO books
       (ISBN, Title, Author, Year)
       VALUES ('1234567890', 'Programming PHP and MySQL', 'Poisson, R.W', 2006);
```

DELETE

The DELETE statement is used to remove row(s) from a table. The syntax is as follows:

```
mysql> DELETE FROM table_name
       [WHERE condition1, condition2 etc...];
```

Example:

```
mysql> DELETE FROM books
       WHERE ISBN = '1234567890';
```

ALTER

ALTER is used to modify the structure of a table. Here is the syntax for ALTER:

```
ALTER [IGNORE] TABLE tbl_name
    alter_specification [, alter_specification] ...

alter_specification:
    ADD [COLUMN] column_definition [FIRST | AFTER col_name ]
  | ADD [COLUMN] (column_definition,...)
  | ADD INDEX [index_name] [index_type] (index_col_name,...)
  | ADD [CONSTRAINT [symbol]]
        PRIMARY KEY [index_type] (index_col_name,...)
  | ADD [CONSTRAINT [symbol]]
        UNIQUE [INDEX] [index_name] [index_type] (index_col_name,...)
  | ADD FULLTEXT [INDEX] [index_name] (index_col_name,...)
      [WITH PARSER parser_name]
  | ADD SPATIAL [INDEX] [index_name] (index_col_name,...)
  | ADD [CONSTRAINT [symbol]]
        FOREIGN KEY [index_name] (index_col_name,...)
        [reference_definition]
  | ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
  | CHANGE [COLUMN] old_col_name column_definition
        [FIRST|AFTER col_name]
  | MODIFY [COLUMN] column_definition [FIRST | AFTER col_name]
  | DROP [COLUMN] col_name
  | DROP PRIMARY KEY
  | DROP INDEX index_name
  | DROP FOREIGN KEY fk_symbol
  | DISABLE KEYS
  | ENABLE KEYS
  | RENAME [TO] new_tbl_name
  | ORDER BY col_name
  | CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
  | [DEFAULT] CHARACTER SET charset_name [COLLATE collation_name]
  | DISCARD TABLESPACE
  | IMPORT TABLESPACE
  | table_options
  | partition_options
  | ADD PARTITION (partition_definition)
  | DROP PARTITION partition_names
  | COALESCE PARTITION number
  | REORGANIZE PARTITION partition_names INTO (partition_definitions)
  | ANALYZE PARTITION partition_names
  | CHECK PARTITION partition_names
  | OPTIMIZE PARTITION partition_names
  | REBUILD PARTITION partition_names
  | REPAIR PARTITION partition_names
  | REMOVE PARTITIONING
```

DROP

DROP is used to completely remove a table or database. The syntax is as follows:

```
mysql> DROP table_name;
```

TRUNCATE

TRUNCATE is used when you want to remove the data, but not the structure of a table. The syntax is as

follows:

```
mysql> TRUNCATE table_name;
```

Common Functions and Operators

COUNT

```
mysql_query(" SELECT COUNT(*|`field_name`) FROM `table_name` ");
```

SUM

```
mysql> SELECT SUM(`field_name`) FROM `table_name`;
```

XML and PHP

Introduction

XML (eXtensible Markup Language) is used in mainstream development. It might have started off as an attempt at a web standard, but is now used even in more traditional applications as a document standard. For example, the Open Document Format employed by Sun in their StarOffice and OpenOffice suites is based on XML.

Because of its wide-spread use in the IT industry, it is fitting that we as PHP developers know how to make use of XML files in our PHP applications.

XML Structure

Since XML documents are extensible, there are no limits to the tags that you can create to define data with. Here is an example of a simple XML document:

```
<?xml version="1.0"?>
<document>
   <title>Isn't this simple!</title>
   <body>XML is as simple as pie. :-)</body>
</document>
```

The reason that it looks so simple, is because it is so simple! Just as in HTML, elements are enclosed by angled brackets: "<" and ">", where the start element differs from the end element by the exclusion of a

forward slash: "/".

Creating an XML Parser in PHP

Defining the XML Parser

In PHP, you define an XML Parser by using the xml_parser_create() function as shown below.

```
<?
$parser = xml_parser_create(ENCODING);
?>
```

You can think of the $parser variable in terms of a parsing engine for the XML document. Note that the ENCODING can be either:

1. ISO-8859-1 (default)

2. US-ASCII

3. UTF-8

Defining the Element Handlers

Element handlers are defined by means of the xml_set_element_handler() function as follows:

```
<?
xml_set_element_handler(XML_PARSER, START_FUNCTION, END_FUNCTION);
?>
```

The three arguments accepted by the xml_set_element_handler() function are:

1. XML_PARSER - The variable that you created when you called the xml_parser_create() function.

2. START_FUNCTION - The name of the function to call when the parser encounters a start element.

3. END_FUNCTION - The name of the function to call when the parser encounters an end element.

e.g.:

```
<?
$parser = xml_parser_create();
xml_set_element_handler($parser, "startElement", "endElement");
?>
```

Defining Character Handlers

Character handlers are created by means of the set_character_handler() function as follows:

```
<?
xml_set_character_handler(XML_PARSER, CHARACTER_FUNCTION);
?>
```

The two arguments accepted by the set_character_handler() function are:

1. XML_PARSER - The variable that you created when you called the xml_parser_create() function.

2. CHARACTER_FUNCTION - The name of the function to call when the parser encounters character data.

Starting the Parser

To finally start the parser, we call the xml_parse() function as follows:

```
<?
xml_parse(XML_PARSER, XML);
?>
```

The two arguments accepted by the xml_parse() function are:

1. The variable that you created when you called the xml_parser_create() function.

2. The XML that is to be parsed.

e.g.:

```
<?
$f = fopen ("simple.xml", 'r');
$data = fread($f, filesize("simple.xml"));
xml_parse($parser, $data);
?>
```

Cleaning Up

After parsing an XML document, it is considered good practice to free up the memory that is holding the parser. This is done by calling the xml_parser_free() function as follows:

```
<?
xml_parser_free(XML_PARSER);
?>
```

Example

```
<?
# --- Element Functions ---

function startElement($parser, $name, $attributes){
   # ... some code
}

function endElement ($parser, $name){
   # ... some code
}

function characterData ($parser, $data){
   # ... some code
}

function load_data($file){
   $f = fopen ($file, 'r');
   $data = fread($f, filesize($file));
   return $data;
}

# --- Main Program Body ---
```

```
$file = "simple.xml";
$parser = xml_parser_create();
xml_set_element_handler($parser, "startElement", "endElement");
xml_set_character_data_handler($parser, "characterData");
xml_parse ($parser, load_data($file));
xml_parser_free($parser);
?>
```

Parsing XML Documents

We have seen the steps needed to successfully parse a XML document with PHP. Lets take a moment to reflect on how these steps are interconnected.

When a XML parser is initialized, php will go through the XML file. When a starting tag is found, a predefined function created by you, the programmer, is called. The same thing happens when php encounters the text between tags, and the end tags.

Here is a complete example of parsing XML documents. This example is a RSS reader which can be used to display News Articles from any RSS feed which conforms to RSS 1.0 standards.

Example

```
<html>
<head>
<title> Google Articles </title>
</head>
<body>
&lt;h2>Google Articles&lt;/h2>
&lt;dl>
<?php

$insideitem = false;
$tag = "";
$title = "";
$description = "";
$link = "";

function startElement($parser, $name, $attrs) {
        global $insideitem, $tag, $title, $description, $link;
        if ($insideitem) {
                $tag = $name;
        }
        elseif ($name == "ITEM") {
                $insideitem = true;
        }
}

function endElement($parser, $name) {
        global $insideitem, $tag, $title, $description, $link;
        if ($name == "ITEM") {
                printf("&lt;dt>&lt;b><a href='%s'>%s</a>&lt;/b>&lt;/dt>",
                trim($link),trim($title));
                printf("&lt;dd>%s&lt;/dd>", trim($description));
                $title = "";
                $description = "";
                $link = "";
                $insideitem = false;
        }
}

function characterData($parser, $data) {
       global $insideitem, $tag, $title, $description, $link;
        if ($insideitem) {
                switch ($tag) {
                        case "TITLE":
                                $title .= $data;
                                break;
                        case "DESCRIPTION":
                                $description .= $data;
```

```
                                break;
                        case "LINK":
                                $link .= $data;
                                break;
                }
        }
}

$xml_parser = xml_parser_create();
xml_set_element_handler($xml_parser, "startElement", "endElement");
xml_set_character_data_handler($xml_parser, "characterData");
# $fp = fopen("http://www.newsforge.com/index.rss", 'r')
$fp = fopen("http://news.google.co.za/nwshp?hl=en&tab=wn&q=&output=rss", 'r')
        or die("Error reading RSS data.");
while ($data = fread($fp, 4096)) {
        xml_parse($xml_parser, $data, feof($fp))
       or die(sprintf("XML error: %s at line %d",
       xml_error_string(xml_get_error_code($xml_parser)),
        xml_get_current_line_number($xml_parser)));
}
fclose($fp);
xml_parser_free($xml_parser);
?>
&lt;/dl>
</body>
</html>
```

Dumping Database Contents into an XML File

1. http://php.net/manual/en/language.oop5.decon.php

Retrieved from "https://en.wikibooks.org/w/index.php?title=PHP_and_MySQL_Programming/Print_version&oldid=3115676"
