WAMP = Windows ~ Apache ~ MySQL ~ PHP
LAMP = Linux ~ Apache ~ MySQL ~ PHP

To get a WAMP server in operation:
1. Install MySQL ODBC Driver
2. Install YellowTip Server Package
3. Configure MySQL — Create Users
4. Test http://localhost
5. Install Eskuel
6. Install MyQRY Generator
7. Install EditPad Lite
8. “Install” PuTTY and WinSCP
The basics of PHP ~ MySQL web applications lies primarily with the database design. If the database is well-designed, then all the PHP code needs to do is pull the data from the tables and display it according to whatever arrangement the web designer has in mind.

The MySQL Server — Logging in

You will get your first opportunity to check if your MySQL server initiated and accepted your username and password. For convenience, it is wise to add the system path to the MySQL server program to the PATH statement on your computer.

1. Click [Start] —> [Run] and type “cmd” to open the command window.
2. Type “c:\Program Files\YellowTip\Mysql\Bin\mysql.exe” -uadmin -uadmin
3. Success will be a mysql > prompt

To prevent the tedium of typing such a long string each time you want to launch the mysql command line interpreter, add the path to the mysql.exe binary to the system PATH variable. See diagrams below and on the following page.
Select the “Environment Variables” button from the bottom of the System Properties panel.

Select “Path” from the System variables list and then click the [Edit] button.

Add the path to the mysql.exe daemon and click [OK].
The mysql command line interpreter works exactly the same in Win32 and Linux.

```
mysql> \h
```

By typing “\h” at the mysql prompt, the server will provide a simplified “help” index.

```
List of all MySQL commands:
  (Commands must appear first on line and end with ';')

  help  (\h)  Display this help.
  ?      (\?)  Synonym for 'help'.
  clear (\c)  Clear command.
  connect (\r)  Reconnect to the server. Optional arguments are db and host.
  ego   (\G)  Send command to mysql server, display result vertically.
  exit  (\q)  Exit mysql. Same as quit.
  go    (\g)  Send command to mysql server.
  notee (\t)  Don't write into outfile.
  print (\p)  Print current command.
  prompt (\R)  Change your mysql prompt.
  quit  (\q)  Quit mysql.
  rehash (\#)  Rebuild completion hash.
  source (\.)  Execute a SQL script file. Takes a file name as an argument.
  status (\s)  Get status information from the server.
  tee   (\T)  Set outfile [to outfile]. Append everything into given outfile
  use   (\u)  Use another database. Takes database name as argument.

Connection id: 477  (Can be used with mysqladmin kill)
```

[The remaining mysql commands will be shown “black on white” for printing purposes]

A few commands are useful to get an idea of how MySQL works. Each command must end with a semicolon (;)

To show the databases on the server, type:

```
mysql > show databases;
```

On a freshly installed server, this will produce something similar to the following:

```
mysql> show databases;
++----------------+
| Database        |
+----------------+
| mysql           |
| test            |
+----------------+
2 rows in set (0.00 sec)
```

There are 2 default databases: mysql and test. You are advised not to tamper with the mysql database, as it contains tables that are essential to the operation of the server. The “test” database is provided for your use. You can also create databases for any and all projects you want to tackle.
To use a specific database, type:

```
mysql> use mysql;
Database changed
mysql>
```

To show tables contained in the mysql database, type:

```
mysql> show tables;
```

To describe the “db” table, type:

```
mysql> describe db;
```

Each “Field” in a table is equivalent to a column. There are 15 columns comprising the “db” table. The field names are somewhat cryptic. Mostly, these fields govern the privileges assigned to any given database.

For comparison, try querying the “test” database:

```
mysql> use test;
Database changed
mysql> show tables;
Empty set (0.00 sec)
```
CREATE A DATABASE:

```sql
mysql> create database workshop;
Query OK, 1 row affected (0.00 sec)
mysql>
```

DROP (DELETE) A DATABASE:

```sql
mysql> drop database workshop;
Query OK, 0 rows affected (0.00 sec)
mysql>
```

CREATE A TABLE:
(The following will create a table called “users” with 6 fields or columns)

```sql
CREATE TABLE users (  userid INT (11) NOT NULL AUTO_INCREMENT,  firstname VARCHAR (30) ,  lastname VARCHAR (30) ,  username VARCHAR (20) ,  password VARCHAR (20) ,  security INT (2)  PRIMARY KEY (userid) );
```

After the table is created, use the previous commands to verify:

```sql
mysql> show tables;
+-----------------------------+
| Tables_in_workshop          |
+-----------------------------+
| users                       |
+-----------------------------+
1 rows in set (0.00 sec)

mysql> desc users;
+---------------+--------+------------+---------+-----------------+-------------+------------------+
| Field         | Type   | Null | Key | Default | Extra          |
+---------------+--------+------------+---------+-----------------+-------------+------------------+
| userid        | int(11)|      | PRI | NULL    | auto_increment  |
| firstnmae     | varchar(30)| YES |     | NULL    |                |
| lastname      | varchar(30)| YES |     | NULL    |                |
| username      | varchar(20)| YES |     | NULL    |                |
| password      | varchar(20)| YES |     | NULL    |                |
| security      | int(2) |      |     | NULL    |                |
+---------------+--------+------------+---------+-----------------+-------------+------------------+
6 rows in set (0.00 sec)
```

To show the contents of a table, use the following command:

```sql
mysql> select * from users;
Empty set (0.00 sec)
mysql>
```

There is no data in the table (yet).
To check that this update did indeed take place, try a SELECT query:

**DISPLAY THE DATA IN A TABLE:**

```sql
mysql> select * from users;
<table>
<thead>
<tr>
<th>userid</th>
<th>firstname</th>
<th>lastname</th>
<th>username</th>
<th>password</th>
<th>security</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bill</td>
<td>Gates</td>
<td>bgates</td>
<td>micro$oft</td>
<td>3</td>
</tr>
</tbody>
</table>
```

2 rows in set (0.00 sec)

**DELETE A ROW (RECORD) FROM THE TABLE:**

```sql
mysql> delete from users where userid = 1;
Query OK, 1 row affected (0.02 sec)
```

**UPDATING A RECORD IN A TABLE:**

```sql
mysql> UPDATE users SET firstname='William' WHERE userid=1;
Query OK, 1 row affected (0.00 sec)
```

**UPDATING MORE THAN ONE FIELD IN A RECORD:**

```sql
mysql> UPDATE users
    SET userid='1',
    firstname='Steven',
    lastname='Jobs',
    username='sjobs',
    password='NeXT',
    security='3'
WHERE userid='1';
Query OK, 1 row affected (0.00 sec)
```

To check that this update did indeed take place, try a SELECT query:

```sql
mysql> select * from users;
<table>
<thead>
<tr>
<th>userid</th>
<th>firstname</th>
<th>lastname</th>
<th>username</th>
<th>password</th>
<th>security</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Steven</td>
<td>Jobs</td>
<td>sjobs</td>
<td>NeXT</td>
<td>3</td>
</tr>
</tbody>
</table>
```

1 row in set (0.00 sec)
**PHP Scripting**

If you can write HTML code, you are on your way to developing PHP scripts. A typical PHP script is loaded with HTML.

A PHP-enabled web server is configured to recognize:
1. The ".php" file extension
2. The presence of `<?` and `?>` within a file

When developing PHP scripts, it is imperative that you view the files using a browser that is pointing to the server’s URL. Conventional HTML documents can be viewed without the need of a web server. Not true for PHP scripts.

On the YellowTip Web Server, files should be placed in the “htdocs” folder (shown below):

![My Computer](image)

This is called the “web root directory”. On a conventional Linux server, the location is typically in `/var/www/html` or perhaps in `/home/httpd/htdocs` or some other location that is specified in the Apache `httpd.conf` file.

On the SME (E-smith) server, the location of the web server “root” directory is:

`/home/e-smith/files/primary/html`

Any document placed in this directory is “on the web”

To test the server (and to find out quite a lot about it), all we have to do is create a PHP script with a single function as the code.

Using Editpad Lite, create the following file within the YellowTip or Apache web root and name it `myphpinfo.php`

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

A successfully configured server will present something similar to the following:

![PHP Version 4.3.7](image)

This is only the upper panel. A print preview will reveal over 10 pages of information!

Mixing HTML and PHP is quite easy to do. The following example will print the current date and time each time the page refreshes.

```html
<html>
<head>
    <title>HTML/PHP Mix</title>
</head>
<body>
The current date and time is:
    <b>
<? echo date("F j, Y, g:i a"); ?>
    </b>
</body>
</html>
```

More complex pages and scripts are simply combinations of additional HTML and PHP.
Introduction to Database-Backed Web Site Design with PHP and MySQL

Making the PHP/MySQL Connection

Before you can make a connection to a MySQL database on your server, you will need to have not only a database to work with, but a privileged user account set up on the MySQL server. Here’s how:

```
mysqladmin -uadmin -padmin create dbname
mysql -uadmin -padmin
Welcome to the MySQL monitor. Commands end with ; or \g. Your MySQL connection id is 12 to server version: 3.23.58
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> grant all privileges on dbname.* to dbuser@localhost identified by 'dbpass';
Query OK, 0 rows affected (0.20 sec)
mysql> flush privileges;
Query OK, 0 rows affected (0.09 sec)
```

Naturally, you will need to change the “dbname”, “dbuser” and “dbpass” to match the system you are designing.

**NOTE**: There is a big difference between the MySQL user and the user accounts on the server itself.

Once the database, the database user, and the database password have been created, you can write a few lines of PHP to connect to the database. In a file named “dbconnect.php”:

```
<?
// DATABASE SETTINGS
$database = "dbname";
$db_user = "dbuser";
$db_pass = "dbpass";
$db_host = "localhost";

// CONNECT TO DATABASE //
$link = mysql_connect($db_host, $db_user, $db_pass)
    or die("Impossible to connect : " . mysql_error());
mysql_select_db($database) or die(mysql_error());

?>
```

This file can be included in any script that requires a connection to the database.
Using INCLUDES to Simplify Scripting

As shown on the previous page, the dbconnect.php script will take care of connecting to the MySQL server and stores it in a variable ($link) that can be used throughout your project.

```php
<?
include("./dbconnect.php");
include("./header.php");

$query = "SELECT field1, field2, field3
FROM
    table
WHERE  field1 = 'some value'
";

$result = mysql_query($query) or die(mysql_error());
$numrows = mysql_num_rows($result);

if($numrows == 0){
    echo "<hr><h3>There are no requests that
   match your query</h3><hr>";
}
while ($row = mysql_fetch_assoc($result))
{
    echo "First Field = " . $row["field1"] . " , " ;
    echo "Second Field = " . $row["field2"] . " , " ;
    echo "Third Field = " . $row["field3"] ;

    mysql_free_result($result);
    mysql_close($link);

    include("./footer.php");
}
?>
```

This script:
1. Connects to the database server
2. Prints out a nice page heading (which is pre-made in header.php)
3. Stores the SQL query and result of the query in variables
4. Runs a “WHILE” routine to display the database values
5. Closes the MySQL connections
6. Prints out a nice page footer (which is pre-made in footer.php)
Basic Logical Building Blocks

if/else Construct

```php
$haveCoupon = true;
if( $haveCoupon )
{
    $itemPrice = 5;
}
```

Further...

```php
$haveCoupon = true;
if( $haveCoupon )
{
    $itemPrice = 5;
} else
{
    $itemPrice = 10;
}
```

And yet further...

```php
$brush_price = 5;
echo "<table border="1"
    align="center">
<tr><th>Quantity</th>
<th>Price</th></tr>
```

```php
for ( 
    $counter = 10;
    $counter <= 100;
    $counter += 10
) {
    echo "<tr><td>
    echo $counter;
    echo "$brush_price * $counter;
    echo "</td></tr>
```

While Loops

```php
$count = 0;
while( $count < 10 )
{
    $count++;
}
```

Expanded...

```php
$brush_price = 5;
$count = 10;
```

```php
echo "<table border="1"
    align="center">
    echo "<tr><th>Quantity</th>
    echo "<th>Price</th></tr>
```

You can use if-then blocks to show different HTML sections depending on different conditions.

```php
<?
    if (some-condition == true)
    {
    Some HTML code goes here
    } else {
    Some different HTML code here
    }
```
Introduction to Database-Backed Web Site Design with PHP and MySQL

FORMS

Step by Step:

1. We first created an HTML form "order.html" that had two input fields specified, "item" and "quantity".
2. The HTML form points to "process.php" and the method is set to "post". It can also be set to "get".
3. The "process.php" script receives the information that was posted by setting new variables equal to the values in the $_POST associative array (or $_GET associative array as the case might be).
4. The PHP echo function is then used to output the customers order.

**order.html Code:**

```html
<html><body>
<h4>Tizag Art Supply Order Form</h4>
<form action="process.php" method="post">
  <select name="item">
    <option>Paint</option>
    <option>Brushes</option>
    <option>Erasers</option>
  </select>
  Quantity: <input name="quantity" type="text" />
  <input type="submit" />
</form>
</body></html>
```

**process.php Code:**

```php
<html><body>
<?php
$quantity = $_POST['quantity'];
$item = $_POST['item'];
echo "You ordered ". $quantity . " " . $item . ".<br/>";
echo "Thank you for ordering from Tizag Art Supplies!";
?>
</body></html>
```
SESSIONS

```php
session_start();
session_register("username","password");
include("dbconnect.php");
$emp_id = $_POST["employee_ID"];
if(!$_SESSION['username']){
    $username = $_POST["username"];
    $password = $_POST["password"];
} else {
    $username = $_SESSION["username"];
    $password = $_SESSION["password"];
}
if (($username == "" && $password == "")) {
    login_form();
} else {
    process_login($username, $password);
}
?>
```

**Initialize Session**
This must be at the top of every page that will call/use a Session variable.

```php
session_start() ;
```

**Register (declare) a Session variable**
To add a Session variable that will be carried till the browser is closed or you kill it yourself.

```php
session_register("mySessionVariable");
```

**Assigning value to a Session variable**
Values added to a Session variable can be letters (ABCabc), numbers (1, 200, 999.99, etc.) making them good for usernames/userID's when logged into a site.

```php
$_SESSION["mySessionVariable"] = "MEMBER";
```

**Unregister a Session variable**
Logging out requires we unregister any and all Session variables used for our application and the users' account.

```php
session_unregister("mySessionVariable");
```

**Destroy a Session**
We should always destroy a Session to help complete a logout.

```php
session_destroy();
```