UNIT V

5.1 Concepts and Installation of MySQL

A *database* is a structured collection of records or data stored in a computer systemand organized in such a way that it can be quickly searched and information can berapidly retrieved.The *SQL* in MySQL stands for *Structured Query Language*. This language is looselybased on English and also used in other databases such as Oracle and Microsoft SQLServer. It is designed to allow simple requests from a database via commands such as
SELECT title FROM publications WHERE author = 'Charles Dickens';
A MySQL database contains one or more *tables*, each of which contains *records* or*rows*. Within these rows are various *columns* or *fields* that contain the data itself.

Table 8-1. Example of a simple database

Author	**Title**	**Type**	**Year**
Mark Twain	The Adventures of Tom Sawyer	Fiction	1876
Jane Austen	Pride and Prejudice	Fiction	1811
Charles Darwin	The Origin of Species	Nonfiction	1856
Charles Dickens	The Old Curiosity Shop	Fiction	1841
William Shakespeare	Romeo and Juliet	Play	1594

To uniquely identify this database, I'll refer to it as the *publications* database in theexamples that followThe main terms you need to acquaint yourself with for now are as follows:

Database
The overall container for a collection of MySQL data

Table
A subcontainer within a database that stores the actual data

Row
A single record within a table, which may contain several fields

Column
The name of a field within a row

Accessing MySQL via the Command Line

There are three main ways in which you can interact with MySQL: using a commandline, via a web interface such as phpMyAdmin, and through a programming languagelike PHP

Starting the Command-Line Interface

The following sections describe relevant instructions for Windows, OS X, and Linux.Windows users.If you installed XAMPP you will be able to access theMySQL executable from the following directory:
C:\xampp\mysql\bin

So, to enter MySQL's command-line interface, select Start→Run, enter **CMD** into theRun box, and press Return. This will call up a Windows command prompt. Fromthere, enter one the following:
C:\xampp\mysql\bin\mysql -u root

Linux users

Accessing MySQL via phpMyAdmin
http://localhost/xampp
Now click the phpMyAdmin link toward the bottom of the lefthand menu to open upthe program.

5.4 MySQL commands
5.2 MySQL structure and syntax
5.3 Types of MySQL tables and Storage engines

5.5 Integration of PHP with MySQL
The reason for using PHP as an interface to MySQL is to format the results of SQLqueries in a form visible in a web page. As long as you can log into your MySQLinstallation using your username and password, you can also do so from PHP.
However, instead of using MySQL's command line to enter instructions and view output,you will create query strings that are passed to MySQL. When MySQL returns itsresponse, it will come as a data structure that PHP can recognize instead of the formattedoutput you see when you work on the command line. Further PHP commandscan retrieve the data and format it for the web page.

5.6 Connection to the MySQL Database

The Process
The process of using MySQL with PHP is as follows:

1. Connect to MySQL and select the database to use.
2. Build a query string.
3. Perform the query.
4. Retrieve the results and output them to a web page.
5. Repeat steps 2 to 4 until all desired data has been retrieved.
6. Disconnect from MySQL.

Most websites developed with PHP contain multiple program files that will requireaccess to MySQL and will thus need the login and password details. Therefore, it'ssensible to create a single file to store these and then include that file wherever it'sneeded. Example 10-1 shows such a file, which I've called *login.php*.

```
<?php // login.php55
$hn = 'localhost';
$db = 'publications';
$un = 'username';
$pw = 'password';
?>
```

Type the example, replacing *username* and *password* with the values you use for yourMySQL database, and save it to the document root directory you set up. The hostname localhost should work as long as you're using a MySQL database onyour local system, and the database *publications* should work if you're typing theexamples.
The enclosing <?php and ?> tags are especially important for the *login.php*file in Example 10-1, because they mean that the lines between can be interpreted *only* asPHP code. If you were to leave them out and someone were to call up the file directlyfrom your website, it would display as text and reveal your secrets. But, with the tagsin place, all that person will see is a blank page. The file will correctly include in yourother PHP files.

The $hn variable will tell PHP which computer to use when connecting to a database.This is required, because you can access MySQL databases on any computer connectedto your PHP installation, and that potentially includes any host anywhere on theWeb.

Connecting to a MySQL Database

Now that you have the *login.php*file saved, you can include it in any PHP files thatwill need to **access the database by using the require_once statement. This is preferable to an include statement, as it will generate a fatal error if the file is not found**

Also, using require_once instead of require means that the file will be read in only when it has not previously been included, which prevents wasteful duplicate disk accesses. Example 10-2 shows the code to use.

Example 10-2. Connecting to a MySQL server with mysqli

```
<?php
require_once 'login.php'; // include ‘filename’;
$conn = new mysqli($hn, $un, $pw, $db);
if ($conn->connect_error) die($conn->connect_error);
?>
```

This example creates a new object called $conn by calling a new instance of themysqlimethod, passing all the values retrieved from the *login.php*file. Error checkingis achieved by referencing the $conn->connect_error property.

The ->operator indicates that the item on the right is a property or method of theobject on the left. In this case, if connect_errorhas a value, then there was an error,so we call the die function and display that property, which details the connectionerror.

Building and executing a query

Sending a query to MySQL from PHP is as simple as issuing it using the querymethod of a connection object. Example 10-3 shows you how to use it.

Example 10-3. Querying a database with mysqli

```
<?php
$query = "SELECT * FROM classics";
$result = $conn->query($query);
//   mysqli_query($conn, $sql) or $conn->query($sql)
if (!$result) die($conn->error);
?>
```

Here the variable $query is assigned a string containing the query to be made, andthen passed to the query method of the $conn object, which returns a result that we place in the object $result. If $result is FALSE, there was a problem and the error property of the connection object will contain the details, so the die function is called to display that error. All the data returned by MySQL is now stored in an easily interrogatable format in the $result object.

For successful SELECT, SHOW, DESCRIBE or EXPLAIN queries mysqli_query() will return a mysqli_result object. For other successful queries mysqli_query() will return TRUE.

Fetching a result

Once you have an object returned in $result, you can use it to extract the data youwant, one item at a time, using the fetch_assoc method of the object.

Example 10-4combines and extends the previous examples into a program that you can type and run yourself to retrieve these results (as depicted in Figure 10-1). I suggest that yousave this script using the filename *query.php*(or use the file saved in the free archiveof files available at *lpmj.net*).

Example 10-4. Fetching results one cell at a time

```
<?php // query.php
require_once 'login.php';
$conn = new mysqli($hn, $un, $pw, $db);
if ($conn->connect_error) die($conn->connect_error);

$query = "SELECT * FROM classics";
$result = $conn->query($query);

if (!$result) die($conn->error);
$rows = $result->num_rows;

for ($j = 0 ; $j < $rows ; ++$j)
{
$result->data_seek($j);

echo 'Author: ' . $result->fetch_assoc()['author'] . '<br>';
$result->data_seek($j);
echo 'Title: ' . $result->fetch_assoc()['title'] . '<br>';
$result->data_seek($j);
echo 'Category: ' . $result->fetch_assoc()['category'] . '<br>';
$result->data_seek($j);
echo 'Year: ' . $result->fetch_assoc()['year'] . '<br>';
$result->data_seek($j);
echo 'ISBN: ' . $result->fetch_assoc()['isbn'] . '<br><br>';
}
$result->close();
$conn->close();
?>
```

Fetching a row

To fetch one row at a time, replace the forloop from Example 10-4 with the onehighlighted in bold in Example 10-5, and you will find that you get exactly the sameresult that was displayed in Figure 10-1. You may wish to save this revised file usingthe name *fetchrow.php*.

Example 10-5. Fetching results one row at a time

```
<?php //fetchrow.php
require_once 'login.php';
$conn = new mysqli($hn, $un, $pw, $db);
if ($conn->connect_error) die($conn->connect_error);
$query = "SELECT * FROM classics";
$result = $conn->query($query);
if (!$result) die($conn->error);
$rows = $result->num_rows;
for ($j = 0 ; $j < $rows ; ++$j)
{
$result->data_seek($j);
$row = $result->fetch_array(MYSQLI_ASSOC);
echo 'Author: ' . $row['author'] . '<br>';
echo 'Title: ' . $row['title'] . '<br>';
```

**echo 'Category: ' . $row['category'] . '
';**
**echo 'Year: ' . $row['year'] . '
';**
**echo 'ISBN: ' . $row['isbn'] . '

';}**$result->close();$conn->close();?>
The fetch_array method can return three types of array according to the valuepassed to it:

MYSQLI_NUM
Numeric array. Each column appears in the array in the order in which you defined itwhen you created (or altered) the table. In our case, the zeroth element of the arraycontains the Author column, element 1 contains the Title, and so on.

MYSQLI_ASSOC
Associative array. Each key is the name of a column. Because items of data are referencedby column name (rather than index number), use this option where possible inyour code to make debugging easier and help other programmers better manageyour code.

MYSQLI_BOTH
Associative and numeric array.
Associative arrays are usually more useful than numeric ones because you can refer toeach column by name, such as $row['author'], instead of trying to remember whereit is in the column order. So this script uses an associative array, leading us to passMYSQLI_ASSOC.

Available in version of PHP
mysqli_fetch_array() and mysqli_fetch_row()

Closing a connection
PHP will eventually return the memory it has allocated for objects after you have finishedwith the script, so in small scripts, you don't usually need to worry about releasingmemory yourself. However, if you're allocating a lot of result objects or fetchinglarge amounts of data, it can be a good idea to free the memory you have been usingto prevent problems later in your script.This becomes particularly important on higher-traffic pages, because the amount ofmemory consumed in a session can rapidly grow. Therefore, note the calls to theclose methods of the objects $result and $conn in the preceding scripts, as soon aseach object is no longer needed, like this:
$result->close();
$conn->close();

mysqli_free_result() - methods recuperates any memory consumed by a result set
OR mysqli_close()

A Practical Example
It's time to write our first example of inserting data in and deleting it from a MySQLtable using PHP. I recommend that you type Example 10-6 and save it to your webdevelopment directory using the filename *sqltest.php*. You can see an example of theprogram's output in Figure 10-2.

Example 10-6. Inserting and deleting using sqltest.php
```
<?php // sqltest.php
require_once 'login.php';
$conn = new mysqli($hn, $un, $pw, $db);
if ($conn->connect_error) die($conn->connect_error);

if (isset($_POST['delete']) &&isset($_POST['isbn']))
{
$isbn = get_post($conn, 'isbn');
$query = "DELETE FROM classics WHERE isbn='$isbn'";
```

```
$result = $conn->query($query);
if (!$result) echo "DELETE failed: $query<br>" .
$conn->error . "<br><br>";
}

if (isset($_POST['author']) &&
isset($_POST['title']) &&
isset($_POST['category']) &&
isset($_POST['year']) &&
isset($_POST['isbn']))
{
$author = get_post($conn, 'author');
$title = get_post($conn, 'title');
$category = get_post($conn, 'category');
$year = get_post($conn, 'year');
$isbn = get_post($conn, 'isbn');
$query = "INSERT INTO classics VALUES" .
"('$author', '$title', '$category', '$year', '$isbn')";
$result = $conn->query($query);
if (!$result) echo "INSERT failed: $query<br>" .
$conn->error . "<br><br>";
}

echo<<<_END
<form action="sqltest.php" method="post"><pre>
Author <input type="text" name="author">
Title <input type="text" name="title">
Category <input type="text" name="category">
Year <input type="text" name="year">
ISBN <input type="text" name="isbn">
<input type="submit" value="ADD RECORD">
</pre></form>

_END;
$query = "SELECT * FROM classics";
$result = $conn->query($query);
if (!$result) die ("Database access failed: " . $conn->error);
$rows = $result->num_rows;
for ($j = 0 ; $j < $rows ; ++$j)
{
$result->data_seek($j);
$row = $result->fetch_array(MYSQLI_NUM);

echo<<<_END
<pre>
Author $row[0]
Title $row[1]
Category $row[2]
Year $row[3]
ISBN $row[4]
</pre>

<form action="sqltest.php" method="post">
```

```
<input type="hidden" name="delete" value="yes">
<input type="hidden" name="isbn" value="$row[4]">
<input type="submit" value="DELETE RECORD"></form>
_END;
}

$result->close();
$conn->close();
functionget_post($conn, $var)
{
return $conn->real_escape_string($_POST[$var]);
}
?>
```

5.7 Creating and Deleting MySQL database using PHP

Creating a Table

Let's assume that you are working for a wildlife park and need to create a database tohold details about all the types of cats it houses. You are told that there are nine *families*of cats—Lion, Tiger, Jaguar, Leopard, Cougar, Cheetah, Lynx, Caracal, andDomestic—so you'll need a column for that. Then each cat has been given a *name*, sothat's another column, and you also want to keep track of their *ages*, which is another.

Of course, you will probably need more columns later, perhaps to hold dietaryrequirements, inoculations, and other details, but for now that's enough to get going.A unique identifier is also needed for each animal, so you also decide to create a columnfor that called *id*.

Example 10-7 shows the code you might use to create a MySQL table to hold thisdata, with the main query assignment in bold text.

Example 10-7. Creating a table called cats

```
<?php
require_once 'login.php';
$conn = new mysqli($hn, $un, $pw, $db);
if ($conn->connect_error) die($conn->connect_error);
$query = "CREATE TABLE cats (
id SMALLINT NOT NULL AUTO_INCREMENT,
family VARCHAR(32) NOT NULL,
name VARCHAR(32) NOT NULL,
age TINYINT NOT NULL,
PRIMARY KEY (id)
)";
$result = $conn->query($query);
if (!$result) die ("Database access failed: " . $conn->error);
?>
```

As you can see, the MySQL query looks pretty similar to how you would type itdirectly in the command line, except that there is no trailing semicolon, as none isneeded when you are accessing MySQL from PHP.

Describing a Table

When you aren't logged into the MySQL command line, here's a handy piece of codethat you can use to verify that a table has been correctly created from inside abrowser. It simply issues the query DESCRIBE cats and then outputs an HTML tablewith four headings—*Column*, *Type*, *Null*, and *Key*—underneath which all columnswithin the table are shown. To use it with other tables, simply replace the name catsin the query with that of the new table (see Example 10-8).

Example 10-8. Describing the table cats

```
<?php
require_once 'login.php';
$conn = new mysqli($hn, $un, $pw, $db);
if ($conn->connect_error) die($conn->connect_error);
$query = "DESCRIBE cats";
$result = $conn->query($query);
if (!$result) die ("Database access failed: " . $conn->error);
$rows = $result->num_rows;
echo "<table><tr><th>Column</th><th>Type</th><th>Null</th><th>Key</th></tr>";
for ($j = 0 ; $j < $rows ; ++$j)
{
$result->data_seek($j);
$row = $result->fetch_array(MYSQLI_NUM);
echo "<tr>";
for ($k = 0 ; $k < 4 ; ++$k) echo "<td>$row[$k]</td>";
echo "</tr>";
}
echo "</table>";
?>
```

Dropping a Table

Dropping a table is very easy to do and is therefore very dangerous, so be careful.Example 10-9 shows the code that you need. However, I don't recommend that youtry it until you have been through the other examples, as it will drop the table *cats*and you'll have to re-create it using Example 10-7.

Example 10-9. Dropping the table cats

```
<?php
require_once 'login.php';
$conn = new mysqli($hn, $un, $pw, $db);
if ($conn->connect_error) die($conn->connect_error);
$query = "DROP TABLE cats";
$result = $conn->query($query);
if (!$result) die ("Database access failed: " . $conn->error);
?>
```

5.8 Updating, Inserting, Deleting records in the MySQL database

Adding Data

Let's add some data to the table by using the code in Example 10-10.*Example 10-10. Adding data to table cats*

```
<?php
require_once 'login.php';
$conn = new mysqli($hn, $un, $pw, $db);
if ($conn->connect_error) die($conn->connect_error);
$query = "INSERT INTO cats VALUES(NULL, 'Lion', 'Leo', 4)";
$result = $conn->query($query);
if (!$result) die ("Database access failed: " . $conn->error);
?>
```

You may wish to add a couple more items of data by modifying $query as follows andcalling up the program in your browser again:

```
$query = "INSERT INTO cats VALUES(NULL, 'Cougar', 'Growler', 2)";
$query = "INSERT INTO cats VALUES(NULL, 'Cheetah', 'Charly', 3)";
```

Retrieving Data

Now that some data has been entered into the *cats* table, Example 10-11 shows howyou can check that it was correctly inserted.

Example 10-11. Retrieving rows from the cats table

```
<?php
require_once 'login.php';
$conn = new mysqli($hn, $un, $pw, $db);
if ($conn->connect_error) die($conn->connect_error);
$query = "SELECT * FROM cats";
$result = $conn->query($query);
if (!$result) die ("Database access failed: " . $conn->error);
$rows = $result->num_rows;
echo "<table><tr><th>Id</th><th>Family</th><th>Name</th><th>Age</th></tr>";
for ($j = 0 ; $j < $rows ; ++$j)
{
$result->data_seek($j);
$row = $result->fetch_array(MYSQLI_NUM);
echo "<tr>";
for ($k = 0 ; $k < 4 ; ++$k) echo "<td>$row[$k]</td>";
echo "</tr>";
}
echo "</table>";
?>
```

This code simply issues the MySQL query SELECT * FROM cats and then displays all the rows returned.

Updating Data

Changing data that you have already inserted is also quite simple.

```
<?php
require_once 'login.php';
$conn = new mysqli($hn, $un, $pw, $db);
if ($conn->connect_error) die($conn->connect_error);
$query = "UPDATE cats SET name='Charlie' WHERE name='Charly'";
$result = $conn->query($query);
if (!$result) die ("Database access failed: " . $conn->error);
?>
```

Deleting Data

Growler the cougar has been transferred to another zoo, so it’s time to remove himfrom the database; see Example 10-13.

Example 10-13. Removing Growler the cougar from the cats table

```
<?php
require_once 'login.php';
$conn = new mysqli($hn, $un, $pw, $db);
if ($conn->connect_error) die($conn->connect_error);
$query = "DELETE FROM cats WHERE name='Growler'";
$result = $conn->query($query);
if (!$result) die ("Database access failed: " . $conn->error);
?>
```

This uses a standard DELETE FROM query, and when you run Example 10-11, you cansee that the row has been removed in the following output:

PHP 5 and later can work with a MySQL database using:

- **MySQLi extension** (the "i" stands for improved)
- **PDO (PHP Data Objects)**

Earlier versions of PHP used the MySQL extension. However, this extension was deprecated in 2012.

Should I Use MySQLi or PDO?

If you need a short answer, it would be "Whatever you like".

Both MySQLi and PDO have their advantages:

PDO will work on 12 different database systems, whereas MySQLi will only work with MySQL databases.

So, if you have to switch your project to use another database, PDO makes the process easy. You only have to change the connection string and a few queries. With MySQLi, you will need to rewrite the entire code - queries included.

Both are object-oriented, but MySQLi also offers a procedural API.

Both support Prepared Statements. Prepared Statements protect from SQL injection, and are very important for web application security.

Example (MySQLi Object-Oriented)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = newmysqli($servername, $username, $password);

// Check connection
```

```php
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
}
echo"Connected successfully";
?>
```

Example (MySQLi Procedural)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = mysqli_connect($servername, $username, $password);

// Check connection
if (!$conn) {
   die("Connection failed: " . mysqli_connect_error());
}
echo"Connected successfully";
?>
```

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";

try {
   $conn = new PDO("mysql:host=$servername;dbname=myDB", $username, $password);
   // set the PDO error mode to exception
   $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
   echo"Connected successfully";
   }
catch(PDOException $e)
   {
   echo"Connection failed: " . $e->getMessage();
   }
?>
```