

Unit Four

4. Connecting to Databases

4.1. Introduction

One of the reasons for PHP's popularity as a Web scripting language is its support for a wide range of relational database systems. This support makes it easy for Web developers to create data-driven Web sites and to prototype new Web applications quickly and efficiently.

PHP supports more than fifteen different database engines, including Microsoft SQL Server, IBM DB2, PostgreSQL, MySQL, and Oracle. Using database in applications helps us to: Read/write data, Store more data, have better organized data, faster access to data, easier to manipulate and relate data to other data.

Database: is a set of tables. We should have 1 database for 1 application.

Tables: is a set of rows and columns. It represents a single concept such as products, customers, orders etc. We can create relationships among tables.

Columns: a set of data of single data type. Ex. FirstName, LastName, Email, Password etc. columns have types such as strings, integers, float, date etc.

Rows: single record of data. Ex. "Abebe", "Kebede", "abe@gmail.com", "password"

Field: is the intersection of a row and a column. Ex. FirstName: "Abebe"

Index: data structure on a table to increase look up speed.

Foreign key: table columns whose values references rows in another table. It is the foundation of relational table.

PHP allows developers to interact with databases in two ways: by using a customized database specific extension, or by using the database-neutral PHP Data Objects (PDO) extension. While the PDO extension is more portable, many developers still find it preferable to use the native database extension, especially when the native extension offers better performance or more features than the PDO version.

Of the various database engines supported by PHP, the most popular one by far is MySQL. Both PHP and MySQL are open-source.

3.1. Connect to a MySQL server/existing Database

PHP provides us different APIs to deal with MySQL server databases: mysql(Original MySQL), mysqli(MySQL improved) and PDO(PHP Data Objects). The differences between these APIs are shown on the table given below:

	mysql	mysqli	PDO
Introduced	v2.0	v5.0	v5.1
Deprecated	v5.5	-	-
Included with PHP	Yes	Yes	Yes
Pre-configured for MySQL	Yes	Yes	
Other databases supported			Yes
Procedural Interface	Yes	Yes	
Object-oriented Interface		Yes	Yes
Prepared statements		Yes	Yes

PHP database interactions in five steps:

- ✓ Create a database connection
- ✓ Perform Database query
- ✓ Use returned data if any
- ✓ Release returned data
- ✓ Close database connection

Creating a database connection:

Before we enable do anything with MySQL in PHP, we should first connect to the MySQL server using specific connection variables. Connection variables consist of the following common parameters, of which the first one is required while others are optional:-

- ✓ **Host name:** This is the name of the server. We can change to whatever host is acting as MySQL server.
- ✓ **User name:** The root user of the system.
- ✓ **User's password:-** This is encrypted written with the form for security.

The common function in PHP that uses for server connection is **mysql_connect()** or **mysqli_connect()** function. This function has the following syntax:- **mysql_connect("hostname", "user", "pass")** to connect with MySQL server. PHP provides **mysql_connect** function to open a database connection. This function can take up to five parameters and returns a MySQL link identifier on success, or FALSE on failure.

The five parameters are the three above and the two below options.

new_link	Optional - If a second call is made to <code>mysql_connect()</code> with the same arguments, no new connection will be established; instead, the identifier of the already opened connection will be returned.
client_flags	Optional - 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

Therefore, *mysql_connect()* function written as follows with respective parameters:-

```
mysql_connect(server,username,passwd,new_link,client_flag);
```

There are also functions in PHP which have different purposes. For instance,

- ✓ **mysql_select_db("database name")** or **mysqli_select_db("connection","database name")** : Equivalent to the MySQL command USE; makes the selected database the active one.
- ✓ **mysql_query("query")**: Used to send any type of MySQL command to the server.
- ✓ **mysql_fetch_rows("results variable from query")**: Used to return a row of the entire results of a database query.
- ✓ **mysqli_affected_rows()**: Print out affected rows from different queries:
- ✓ **mysql_fetch_array("results variable from query")**: Used to return several rows of the entire results of a database query.
- ✓ **mysql_free_result("result variable from query")**: Used to release the returned results.
- ✓ **mysql_error()**: Shows the error message that has been returned directly from the MySQL server.

We issue this connection command with the PHP function called `mysql_connect()` or `mysqli_connect()`. As with all of our PHP/MySQL statements, you can either put the information into variables, or leave them as text in MySQL query as shown below:-:

```
$host = "localhost";
```

```
$user = "root";
```

```
$pass = "";
```

```
$connect = mysql_connect($host, $user, $pass); Or simply $connect = mysql_connect("localhost", "root", "");
```

Or you can also use

```
$connect=mysqli_connect($host, $user, $pass);
```

Choosing and Creating the working Database

After establishing a MySQL connection with the code above, you then need to choose which database you will be using with this connection. This is done with the *mysql_select_db("database-name")* or *mysqli_select_db("connection","databasename")* function. If the database you are looking to work on is not available, you can create it using *mysql_query()* or *mysqli_query()* function together with CREATE command followed by database name. *mysql_query* function can take two parameters and returns TRUE on success or FALSE on failure. The parameters are:- sql and connection. The syntax of the function is:- *mysql_query(sql, connection variable);* or *mysqli_query(connection variable,sql);*

To create a database uses the following sql syntax:

```
CREATE DATABASE database_name
```

Example Using : <i>mysql</i>	Example Using : <i>mysqli</i>
------------------------------	-------------------------------

<pre> <?php \$connection=\$mysql_connect("localhost", "root", ""); If(\$connection) echo "Connected to MySQL
"; \$sql=mysql_select_db("test"); If(\$sql){ echo "Connected to Database";//display this message if database is selected }else{ \$result=mysql_query("create database test",\$connection);//create a database called test if not available If(\$result) mysql_select_db("test");//select test database else die("Database not selected:".mysql_error()); } ?> mysql_close();//closing connection </pre>	<pre> <?php \$connection=\$mysqli_connect("localhost", "root", ""); If(\$connection) echo "Connected to MySQL
"; }else{ \$result=mysqli_query(\$connection,"create database test");//create a database called test if not available If(\$result) mysqli_select_db(\$connection,"test") else die("Database not selected:".mysql_error());//select test database } mysqli_close(\$connection);//closing connection ?> </pre>
---	---

Output:

Connected to MySQL
Connected to Database

- ✓ ***mysql_query ("create database test",\$connection)***: told MySQL to create a database called test.
- ✓ ***die(mysql_error());*** will print out an error if there is a problem in the database creation process.

Closing Query

- ✓ When you are finished working with query results retrieved with the ***mysql_query()*** function, use the ***mysql_free_result()*** function to close the resultset
- ✓ To close the resultset, pass to the ***mysql_free_result()*** function the variable containing the result pointer from the ***mysql_query()*** function

Create Table MySQL

Before you enter data (rows) into a table, you must first define what kinds of data will be stored (columns). This can be done using ***Create*** sql statement.

Syntax:

CREATE TABLE *table_name* (*column_name1 data_type,column_name2 data_type,....*)

We are now going to design a MySQL query to summon our table from database test.

Example Using : <i>mysql</i>	Example Using : <i>mysqli</i>
<pre> <?php // Make a MySQL Connection \$server="localhost"; \$dbuser="root"; \$dbpassword=""; \$dbname="test"; \$connection=mysql_connect(\$server,\$dbuser,\$dbpass word); if(\$connection) { \$sql=mysql_select_db(\$dbname); if(\$sql) { // Create a MySQL table in the selected database \$sql2="CREATE TABLE example(id INT NOT NULL AUTO_INCREMENT, PRIMARY KEY(id), name VARCHAR(30), age INT)"; \$result=mysql_query(\$sql2,\$connection); if(\$result) echo "Table Created!"; else die("Table Creation failed:".mysql_error()); }else{ die("Database selection failed:".mysql_error()); } }else{ die("Mysql server Connection Failed:".mysql_error()); } mysql_close();//closing connection ?> </pre>	<pre> <?php // Make a MySQL Connection \$server="localhost"; \$dbuser="root"; \$dbpassword=""; \$dbname="test"; \$connection=mysqli_connect(\$server,\$dbuser,\$dbpass word,\$dbname); if(\$connection) { // Create a MySQL table in the selected database \$sql2="CREATE TABLE example(id INT NOT NULL AUTO_INCREMENT, PRIMARY KEY(id), name VARCHAR(30), age INT)"; \$result=mysqli_query(\$connection,\$sql2); if(\$result) echo "Table Created!"; else die("Table Creation failed:".mysqli_error(\$connection)); }else{ die("Mysql server Connection Failed:".mysqli_error(\$connection)); } mysqli_close(\$connection);//closing connection ?> </pre>

3.2. Send/Insert Data to a Database

When data is put into a MySQL table it is referred to as inserting data. When inserting data it is important to remember the exact names and types of the table's columns. If you try to place a 500 word essay into a column that only accepts integers of size three, you will end up with errors. Inserting data into a table can be performed using **Insert into** sql statement.

Syntax:

INSERT INTO table_name VALUES (value1, value2, value3,...) Or

INSERT INTO table_name (column1, column2,...) VALUES (value1, value2,...)

```

<html><style>
#style{ display:block;
width:220px;
height:120px;
border:1px solid black;
line-height:30px;
}
#style p{
margin-left:20px;
margin-right:20px;
}</style><body>
<?php
if(isset($_POST['submit']))
{
    //getting name and age from the user
    $name=$_POST['name'];
    $age=$_POST['age'];
    // Make a MySQL Connection
    $connection=mysql_connect("localhost", "root", "");
    if($connection){
    $sql=mysql_select_db("test");
    if($sql){
    // Insert a row of information into the table "example"
    $sql2="INSERT INTO example(name, age) VALUES('$name', '$age') ";
    $result=mysql_query($sql2,$connection);

```

The screenshot shows a web form with a white background and a thin black border. It contains two text input fields. The first field is labeled 'Name:' and contains the text 'name'. The second field is labeled 'Age:' and contains the text 'age'. Below the input fields are two buttons: 'Submit' and 'Reset', both with a light gray background and black text.

```

    If($result)
        echo "Data Inserted!";
    else
        die("Data not inserted:".mysql_error());
} else
    die("Database not selected:".mysql_error());

}else{
    die("Connection Failed:".mysql_error());
}

mysql_close();//closing connection
?>
<div id="style"><p><form action="" method="post">
Name:<input type="text" name="name" required="" placeholder="name"><br>
Age:<input type="text" name="age" required="" placeholder="age"><br>
<input type="submit" value="Submit" name="submit"><input type="reset" value="Reset">
</form></p></div></body></html>

```

			id	name	age
<input type="checkbox"/>			1	Abebe	17
<input type="checkbox"/>			2	Ermias	19
<input type="checkbox"/>			3	Samson	24
<input type="checkbox"/>			4	Amen	16

If you want to use mysqli, replace mysql connection and data insertion by the following code:

// Make a MySQL Connection

```

    $connection=mysqli_connect("localhost", "root", "", "test");
    If($connection){
        // Insert a row of information into the table "example"
        $sql2="INSERT INTO example(name, age) VALUES('$name', '$age') ";
        $result=mysqli_query($connection,$sql2);
        If($result)
            echo "Data Inserted!";
        else
            die("Data not inserted:".mysqli_error($connection));
    } else
        die("Database not selected:".mysqli_error($connection));
    }else{
        die("Connection Failed:".mysqli_error($connection));
    }

    mysqli_close($connection);//closing connection
?>

```

3.3. Retrieve Data from a Database

Usually most of the work done with MySQL involves pulling down data from a MySQL database. In MySQL, data is retrieved with the "SELECT" keyword. Think of SELECT as working the same way as it does on your computer. If you want to copy some information in a document, you would first select the desired information, then copy and paste.

Before attempting to retrieve data, be sure that you have created a table that contains some data. In this example, we will output the first entry of our MySQL "examples" table to the web browser.

The SELECT statement is used to select data from a database.

Syntax: *SELECT column_name(s) FROM table_name*

Example:

Example Using : <i>mysql</i>	Example Using : <i>mysqli</i>
<pre> <?php // Make a MySQL Connection \$connection=mysql_connect("localhost", "root", ""); If(\$connection){ \$sql=mysql_select_db("test"); If(\$sql){ // Retrieving all records from "example" table \$sql2="select * from example"; \$result=mysql_query(\$sql2,\$connection) ; If(\$result){//Displaying the values using a table echo "<table border=1><tr><th>Id</th>"; echo "<th>Name</th><th>Age</th></tr>"; While(\$row=mysql_fetch_array(\$result)) { echo "<tr><td>".\$row['id']. "</td>"; echo "<td>".\$row['name']. "</td>"; echo "<td>".\$row['age']. "</td></tr>"; } echo "</table>"; }else die("Record Not Selected:".mysql_error()); } else die("Database not </pre>	<pre> <?php // Make a MySQL Connection \$connection=mysqli_connect("localhost", "root", "", "test"); If(\$connection){ // Retrieving all records from "example" table \$sql="select * from example "; \$result=mysqli_query(\$connection,\$sql) ; If(\$result){ //Displaying the values using a table echo "<table border=1><tr><th>Id</th>"; echo "<th>Name</th><th>Age</th></tr>"; While(\$row=mysqli_fetch_assoc(\$result)) { echo "<tr><td>".\$row['id']. "</td>"; echo "<td>".\$row['name']. "</td>"; echo "<td>".\$row['age']. "</td></tr>"; } echo "</table>"; }else die("Record Not Selected:".mysqli_error(\$connection)); }else{ die("Connection Failed:".mysqli_error(\$connection)); </pre>

```

selected:".mysql_error());
    }else{
        die("Connection Failed:".mysql_error());
    }
?>
}
mysqli_close($connection);//closing connection
?>

```

Output:

Id	Name	Age
1	Abebe	17
2	Ermias	19
3	Samson	24
4	Amen	16

When you select items from a database using *mysql_query*, the data is returned as a MySQL result. Since we want to use this data in our table we need to store it in a variable. *\$result* now holds the result from our *mysql_query*.

The *mysql_fetch_array* function gets the next-in-line associative/numeric array from a MySQL result. The *mysql_fetch_row* function gets the next-in-line numeric array from a MySQL result. The *mysqli_fetch_assoc* function gets the next-in-line associative array from a MySQL result. By putting it in a while loop it will continue to fetch the next array until there is no next array to fetch. This function can be called as many times as you want, but it will return FALSE when the last associative array has already been returned.

By placing this function within the conditional statement of the while loop,

- ✓ We can retrieve the next associative array from our MySQL Resource, *\$result*, so that we can print out the id, name and age of that person.
- ✓ We can tell the while loop to stop printing out information when the MySQL Resource has returned the last array, as False is returned when it reaches the end and this will cause the while loop to halt.

In our MySQL table "example" there are only three fields: id, name and age. These fields are the keys to extracting the data from our associative array. To get the id, name and age we use *\$row['id']*, *\$row['name']* and *\$row['age']* respectively. The html table tag is used to let the output look better. The above select statement retrieves everything from the example table. If you want to retrieve specific record, you can use where clause in the select statement. For example:
mysql_query("select * from example where id=1");

3.4. Modify/Updating Existing Data

Imagine that you have a MySQL table that holds the information of all the employees in your company. One of the columns in this table is called "Seniority" and it holds an integer value of how many months an employee has worked at your company. Unfortunately for you, your job is to update these numbers every month.

You may be thinking that you'll have to open up your MySQL administration tool and edit each entry by hand. That would take hours. On the other hand, you could master MySQL and have an automated script that you run each month to get the job done for you.

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

Syntax

UPDATE table_name SET column1=value, column2=value2,... WHERE some_column=some_value

Example: In the example table we have 4 records. The person whose has an id number of 4 is turned to 17. So change the persons age accordingly.

Example Using : <i>mysql</i>	Example Using : <i>mysqli</i>
<pre> <?php // Make a MySQL Connection \$connection=mysql_connect("localhost", "root", ""); If(\$connection){ \$sql=mysql_select_db("test"); If(\$sql){ // Updating record \$sql2=" update example set age=17 where id=4 "; \$result=mysql_query(\$sql2,\$connection) ; If(\$result){ echo "Age successfully updated!"; }else die("Record Not updated:".mysql_error()); } else die("Database not selected:".mysql_error()); }else{ die("Connection Failed:".mysql_error()); } } </pre>	<pre> <?php // Make a MySQL Connection \$connection=mysqli_connect("localhost", "root", "", "test"); If(\$connection){ // Updating record \$sql="update example set age=17 where id=4 "; \$result=mysqli_query(\$connection,\$sql) ; If(\$result){ echo "Age successfully updated!"; }else die("Record Not Updated:".mysqli_error(\$connection)); }else{ die("Connection Failed:".mysqli_error(\$connection)); } mysqli_close(\$connection);//closing connection ?> </pre>

?>	
----	--

Output:

Age successfully updated!

In the above example:

- ✓ **UPDATE** - Performs an update MySQL query
- ✓ **SET** - The new values to be placed into the table follow SET
- ✓ **WHERE** - Limits which rows are affected

3.5. Remove Existing Data

Maintenance is a very common task that is necessary for keeping MySQL tables current. From time to time, you may even need to delete items from your database. Some potential reasons for deleting a record from MySQL include when: someone deletes a post from a forum.

The DELETE query is very similar to the UPDATE Query. We need to choose a table, tell MySQL to perform the deletion, and provide the requirements that a record must have for it to be deleted.

Syntax:

DELETE from table_name where column_name comparison_operator value

For example: delete all records in which age is below 18.

<i>Example Using : mysql</i>	<i>Example Using : mysqli</i>
<pre><?php // Make a MySQL Connection \$connection=mysql_connect("localhost", "root", ""); If(\$connection){ \$sql=mysql_select_db("test"); If(\$sql){ // deleting records \$sql2="delete from example where age<18 "; \$result=mysql_query(\$sql2,\$connection) ; If(\$result){ echo "Records Deleted!"; } }</pre>	<pre><?php // Make a MySQL Connection \$connection=mysqli_connect("localhost", "root", "", "test"); If(\$connection){ // Updating records \$sql=" delete from example where age<18"; \$result=mysqli_query(\$connection,\$sql) ; If(\$result){ echo "Records Deleted!"; }else die("Record Not Deleted:".mysqli_error(\$connection)); }</pre>

<pre> }else die("Record Not deleted:".mysql_error()); } else die("Database not selected:".mysql_error()); }else{ die("Connection Failed:".mysql_error()); } ?> </pre>	<pre> }else{ die("Connection Failed:".mysql_error(\$connection)); } mysqli_close(\$connection);//closing connection ?> </pre>
--	--

Output:

Records Deleted!

3.6. Data base security using server side scripting

Nowadays, databases are cardinal components of any web based application by enabling websites to provide varying dynamic content. Since very sensitive or secret information can be stored in a database, you should strongly consider protecting your databases.

To retrieve or to store any information you need to connect to the database, send a legitimate query, fetch the result, and close the connection.

Encryption in PHP

Once an attacker gains access to your database directly (bypassing the web server), stored sensitive data may be exposed or misused, unless the information is protected by the database itself. Encrypting the data is a good way to mitigate this threat, but very few databases offer this type of data encryption.

The easiest way to work around this problem is to first create your own encryption package, and then use it from within your PHP scripts. PHP provides us with different types of encryptions such as: md5, sha1, hash, crypt, hashed_password etc.

Example:

```

<?php
    $pass="12345678";
    echo "md5 encryption $pass=".md5($pass)."<br>";
    echo "sha1 encryption $pass=".sha1($pass)."<br>";
    echo "hash encryption $pass=".hash('sha1',$pass)."<br>";

```

```
echo "crypt encryption $pass=".crypt($pass,$salt);
?>
```

Output:

```
md5 encryption 12345678=25d55ad283aa400af464c76d713c07ad
sha1 encryption 12345678=7c222fb2927d828af22f592134e8932480637c0d
hash encryption 12345678=7c222fb2927d828af22f592134e8932480637c0d
crypt encryption 12345678=$1$.90.tj5.$CG0sUopGFc1ADWxBqDjPu.
```

In the above example, the **salt** parameter is optional. However, **crypt ()** creates a weak password without the salt. Make sure to specify a strong enough salt for better security.

SQL Injection

SQL injection attacks are extremely simple to defend against, but many applications are still vulnerable. Consider the following SQL statement:

```
<?php
$sql = "INSERT INTO users (reg_username,reg_password,reg_email) VALUES ('$_POST['reg_username']',
'$reg_password', '{$_POST['reg_email']}')";
?>
```

This query is constructed with `$_POST`, which should immediately look suspicious.

Assume that this query is creating a new account. The user provides a desired username and an email address. The registration application generates a temporary password and emails it to the user to verify the email address. Imagine that the user enters the following as a username:

```
bad_guy', 'mypass', ''), ('good_guy
```

This certainly doesn't look like a valid username, but with no data filtering in place, the application can't tell. If a valid email address is given (`shiflett@php.net`, for example), and `1234` is what the application generates for the password, the SQL statement becomes the following:

```
<?php
$sql = "INSERT INTO users (reg_username,reg_password,reg_email) VALUES ('bad_guy', 'mypass', ''),
('good_guy','1234',
'shiflett@php.net')";
?>
```

Rather than the intended action of creating a single account (`good_guy`) with a valid email address, the application has been tricked into creating two accounts, and the user supplied every detail of the `bad_guy` account.

While this particular example might not seem so harmful, it should be clear that worse things could happen once an attacker can make modifications to your SQL statements.

For example, depending on the database you are using, it might be possible to send multiple queries to the database server in a single call. Thus, a user can potentially terminate the existing query with a semicolon and follow this with a query of the user's choosing.

MySQL, until recently, does not allow multiple queries, so this particular risk is mitigated. Newer versions of MySQL allow multiple queries, but the corresponding PHP extension (`ext/mysqli`) requires that you use a separate function if you want to send multiple queries (`mysqli_multi_query()` instead of `mysqli_query()`). Only allowing a single query is safer, because it limits what an attacker can potentially do.

Protecting against SQL injection is easy:

- ✓ ***Filter your data:*** This cannot be overstressed. With good data filtering in place, most security concerns are mitigated, and some are practically eliminated.
- ✓ ***Quote your data:*** If your database allows it (MySQL does), put single quotes around all values in your SQL statements, regardless of the data type.
- ✓ ***Escape your data:*** Sometimes valid data can unintentionally interfere with the format of the SQL statement itself. Use `mysql_escape_string()` or `mysqli_real_escape_string()` an escaping function native to your particular database. If there isn't a specific one, `addslashes()` is a good last resort.