

COMP30680

Web Application Development

PHP part 2 – Connecting to a Database

David Coyle

d.coyle@ucd.ie

PHP and MySQL

With PHP, you can connect to and manipulate databases.

MySQL is the most popular database system used with PHP. It is the de-facto standard database system for web sites with HUGE volumes of both data and end-users (like Facebook, Twitter, and Wikipedia).

The data in a MySQL database are stored in tables. A table is a collection of related data, and it consists of columns and rows.

PHP combined with MySQL are cross-platform (you can develop in Windows and serve on a Unix platform).

XAMPP includes **PHP 5** and **MariaDB**. MariaDB is a community-developed fork of the MySQL relational database management system intended to remain free under the GNU GPL.

PHP and MySQL

To make full use a PHP and MySQL you need some familiarity with SQL and know how to write queries:

```
SELECT LastName FROM Employees
```

This is a simple query that selects all the data in the LastName column for the Employees table in database.

This lecture and the supporting materials on w3schools walk through key operations for connecting to and manipulating a MySQL database using PHP:

- Connecting to a database
- Creating a database
- Creating a table in a database
- Inserting data into a database
- Selecting data
- Deleting data
- Updating data

Materials: http://www.w3schools.com/php/php_mysql_intro.asp

PHP and MySQL - options

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.

Which should you use?

Both MySQLi and PDO have their advantages:

PDO will work on 12 different database systems, where as 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.

My advice: If PHP is new to you, it's probably best to go with PDO.

Connecting to MySQL

Before we can access data in the MySQL database, we need to be able to connect to the database server.

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

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

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

This example uses the MySQLi Object-Oriented method.

To close this connection you use the following:

```
$conn->close();
```

Connecting to MySQL

MySQLi Procedural

```
<?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";
?>
```

The code to close this connection is:

```
mysqli_close($conn);
```

Connecting to MySQL

PDO (PHP Data Objects)

```
<?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();
}

?>
```

The code to close this connection is:

```
$conn = null;
```

Notice that in the PDO example above specifies a database (myDB). PDO requires a valid database to connect to. If no database is specified, an exception is thrown.

Connecting to MySQL

PDO (PHP Data Objects)

```
<?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();
}

?>
```

The code to close this connection is:

```
$conn = null;
```

A great benefit of PDO is that it has an exception class to handle any problems that may occur in database queries.

If an exception is thrown within the try{ } block, the script stops executing and flows directly to the first catch(){ } block.

Notice that in the PDO example above we have also specified a database (myDB). PDO requires a valid database to connect to. If no database is specified, an exception is thrown.

Using a config file

Rather than repeating connection information in lots of PHP files, it is often handy to create a file that contains the information needed to connect to your databases:

```
<?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();
}

?>
```

Using a config file

Rather than repeating connection information in lots of PHP files, it is often handy to create a file that contains the information needed to connect to your databases:

dbconfig.php

```
1  <?php
2      $host = 'localhost';
3      $dbname = 'myDB';
4      $username = 'root';
5      $password = '';
6  ?>
7
```

```
1  <?php
2  require_once 'dbconfig.php';
3
4  try {
5      $conn = new PDO("mysql:host=$host;dbname=$dbname", $username, $password);
6      echo "Connected to $dbname at $host successfully.";
7  } catch (PDOException $pe) {
8      die("Could not connect to the database $dbname :". $pe->getMessage());
9  }
10 ?>
11
```

Create a database

The **CREATE DATABASE** statement is used to create a database in MySQL.

```
<?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);
    $sql = "CREATE DATABASE myDBPDO";
    // use exec() because no results are returned
    $conn->exec($sql);
    echo "Database created successfully<br>";
}
catch(PDOException $e)
{
    echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

Create MySQL Tables

The **CREATE TABLE** statement is used to create a table in MySQL.

The command below creates a table named "MyGuests", with five columns: "id", "firstname", "lastname", "email" and "reg_date":

```
CREATE TABLE MyGuests (  
  id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,  
  firstname VARCHAR(30) NOT NULL,  
  lastname VARCHAR(30) NOT NULL,  
  email VARCHAR(50),  
  reg_date TIMESTAMP  
)
```

The data type specifies what type of data the column can hold. After the data type, you can specify other optional attributes for each column. E.g.:

- NOT NULL - Each row must contain a value for that column, null values are not allowed
- UNSIGNED - Used for number types, limits the stored data to positive numbers and zero
- PRIMARY KEY - Used to uniquely identify the rows in a table. The column with PRIMARY KEY setting is often an ID number, and is often used with AUTO_INCREMENT

PDO create table

```
try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // sql to create table
    $sql = "CREATE TABLE MyGuests (
    id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    firstname VARCHAR(30) NOT NULL,
    lastname VARCHAR(30) NOT NULL,
    email VARCHAR(50),
    reg_date TIMESTAMP
    )";

    // use exec() because no results are returned
    $conn->exec($sql);
    echo "Table MyGuests created successfully";
}
catch(PDOException $e)
{
    echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

Insert Data

After a database and a table have been created, we can start adding data to them.

The syntax rules are important:

- The SQL query must be quoted in PHP
- String values inside the SQL query must be quoted
- Numeric values must not be quoted
- The word NULL must not be quoted
- The INSERT INTO statement is used to add new records to a MySQL table:

```
INSERT INTO table_name (column1, column2, column3,...)  
VALUES (value1, value2, value3,...)
```

If a column is AUTO_INCREMENT (like the "id" column) or TIMESTAMP (like the "reg_date" column), it is no need to be specified in the SQL query; MySQL will automatically add the value.

Insert Data - PDO

```
try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";
    // use exec() because no results are returned
    $conn->exec($sql);
    $last_id = $conn->lastInsertId();
    echo "New record created successfully. Last inserted ID is: " . $last_id;
}
catch(PDOException $e)
{
    echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

Insert Data - PDO

```
try {  
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);  
    // set the PDO error mode to exception  
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);  
    $sql = "INSERT INTO MyGuests (firstname, lastname, email)  
VALUES ('John', 'Doe', 'john@example.com')";  
    // use exec() because no results are returned  
    $conn->exec($sql);  
    $last_id = $conn->lastInsertId();  
    echo "New record created successfully. Last inserted ID is: " . $last_id;  
}  
catch(PDOException $e)  
{  
    echo $sql . "<br>" . $e->getMessage();  
}  
  
$conn = null;  
?>
```

If we perform an INSERT or UPDATE on a table with an AUTO_INCREMENT field, we can get the ID of the last inserted/updated record immediately.

Insert Multiple Records

```
try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // begin the transaction
    $conn->beginTransaction();
    // our SQL statements
    $conn->exec("INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')");
    $conn->exec("INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Mary', 'Moe', 'mary@example.com')");
    $conn->exec("INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Julie', 'Dooley', 'julie@example.com')");

    // commit the transaction
    $conn->commit();
    echo "New records created successfully";
}
catch(PDOException $e)
{
    // roll back the transaction if something failed
    $conn->rollback();
    echo "Error: " . $e->getMessage();
}
```

Create a series of queries to execute and then commit them.

See w3schools for MySQLi versions.

Select Data

The SELECT statement is used to select data from one or more tables:

```
SELECT column_name(s) FROM table_name
```

```
SELECT * FROM table_name
```


See [php_query.php](#) - it does several things:

1. Connects to a database.
2. An SQL query that selects the firstname, lastname, and job title columns from the Employees table.
3. Prepares and then executes a query that contains this query.
4. Puts the resulting data into a variable called \$q using the FETCH_ASSOC mode.
5. Using a while loop it loops through the results and populates a table with the data selected from the database.

Delete Data

The DELETE statement is used to delete records from a table:

```
try {  
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);  
    // set the PDO error mode to exception  
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);  
  
    // sql to delete a record  
    $sql = "DELETE FROM MyGuests WHERE id=3";  
  
    // use exec() because no results are returned  
    $conn->exec($sql);  
    echo "Record deleted successfully";  
}  
catch(PDOException $e)  
{  
    echo $sql . "<br>" . $e->getMessage();  
}
```



The WHERE clause specifies which record or records should be deleted. If you omit the WHERE clause, all records will be deleted!

Update Data

```
try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $sql = "UPDATE MyGuests SET lastname='Doe' WHERE id=2";

    // Prepare statement
    $stmt = $conn->prepare($sql);

    // execute the query
    $stmt->execute();

    // echo a message to say the UPDATE succeeded
    echo $stmt->rowCount() . " records UPDATED successfully";
}
catch(PDOException $e)
{
    echo $sql . "<br>" . $e->getMessage();
}
```

Prepared statements

A prepared statement is a feature used to execute the same (or similar) SQL statements repeatedly with high efficiency.

Prepared statements basically work like this:

1. Prepare: An SQL statement template is created and sent to the database. Certain values are left unspecified, called parameters (labeled "?").

Example: `INSERT INTO MyGuests VALUES(?, ?, ?)`

2. The database parses, compiles, and performs query optimization on the SQL statement template, and stores the result without executing it.
3. Execute: At a later time, the application binds the values to the parameters, and the database executes the statement. The application may execute the statement as many times as it wants with different values

PDO example

```
// prepare sql and bind parameters
$stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email)
VALUES (:firstname, :lastname, :email)");
$stmt->bindParam(':firstname', $firstname);
$stmt->bindParam(':lastname', $lastname);
$stmt->bindParam(':email', $email);

// insert a row
$firstname = "John";
$lastname = "Doe";
$email = "john@example.com";
$stmt->execute();

// insert another row
$firstname = "Mary";
$lastname = "Moe";
$email = "mary@example.com";
$stmt->execute();

// insert another row
$firstname = "Julie";
$lastname = "Dooley";
$email = "julie@example.com";
$stmt->execute();
```

Questions, Suggestions?

Next:

PHP file management.