فصل چهارم

ارتباط با پایگاه داده ها

MySQL Database

- With PHP, you can connect to and manipulate databases.
- MySQL is the most popular database system used with PHP.

MySQL Database

- What is MySQL?
  - MySQL is a database system used on the web
  - MySQL is a database system that runs on a server
  - MySQL is ideal for both small and large applications
  - MySQL is very fast, reliable, and easy to use
  - MySQL uses standard SQL

MySQL Database

- What is MySQL?
  - MySQL compiles on a number of platforms
  - MySQL is free to download and use
  - MySQL is developed, distributed, and supported by Oracle Corporation
  - MySQL is named after co-founder Monty Widenius's daughter: My

⚠ 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)

MySQL Database

- Database Queries
  - A query is a question or a request.
  - We can query a database for specific information and have a recordset returned.
  - Look at the following query (using standard SQL):

```
SELECT LastName FROM Employees
```

  - The query above selects all the data in the "LastName" column from the "Employees" table

MySQL Database

- Download MySQL Database
  - If you don't have a PHP server with a MySQL Database, you can download it for free here: http://www.mysql.com
- Facts About MySQL Database
  - MySQL is the de-facto standard database system for web sites with HUGE volumes of both data and end-users (like Facebook, Twitter, and Wikipedia).
  - Another great thing about MySQL is that it can be scaled down to support embedded database applications.
  - Look at http://www.mysql.com/customers/ for an overview of companies using MySQL.

MySQL Connect

- 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.

MySQL Connect

- 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.

MySQL Connect

- Open a Connection to MySQL MySQLi Object-Oriented

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

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

MySQL Connect

- Open a Connection to MySQL 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";
?>
```

MySQL Connect

- Open a Connection to MySQL PDO

```
<?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();
    }
?>
```

MySQL Connect

- Open a Connection to MySQL PDO
  - ⚠ In the PDO example above we have also ***specified a database (myDB)***. PDO require a valid database to connect to. If no database is specified, an exception is thrown.
  - ⚠ A great benefit of PDO is that it has an exception class to handle any problems that may occur in our database queries. If an exception is thrown within the `try{ }` block, the script stops executing and flows directly to the first `catch(){ }` block.

MySQL Connect

- Close the Connection
  - The connection will be closed automatically when the script ends. To close the connection before, use the following:
    - Example (MySQLi Object-Oriented)

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

    - Example (MySQLi Procedural)

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

    - Example (PDO)

```
$conn = null;
```

Create Database

- A database consists of one or more tables.
- You will need special CREATE privileges to create or to delete a MySQL database.
- The CREATE DATABASE statement is used to create a database in MySQL.
- The following examples create a database named "myDB":

Create a MySQL Database Using MySQLi

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
//Create connection
$conn = new mysqli($servername, $username, $password);
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
//Create database
$sql = "CREATE DATABASE myDB";
if ($conn->query($sql) === TRUE) {
    echo "Database created successfully";
} else {
    echo "Error creating database: " . $conn->error;
}
$conn->close();
?>
```

⚠ When you create a new database, you must only specify the first three arguments to the mysqli object (servername, username and password).

Create Tables

- A database table has its own unique name and consists of columns and rows.
- The CREATE TABLE statement is used to create a table in MySQL.
- We will create a table named "MyGuests", with five columns: "id", "firstname", "lastname", "email" and "reg_date":

Create Tables

- Notes on the table:
  - The data type specifies what type of data the column can hold. For a complete reference of all the available data types, go to our Data Types reference.
  - After the data type, you can specify other optional attributes for each column:
    - NOT NULL - Each row must contain a value for that column, null values are not allowed
    - DEFAULT value - Set a default value that is added when no other value is passed
    - UNSIGNED - Used for number types, limits the stored data to positive numbers and zero
    - AUTO INCREMENT - MySQL automatically increases the value of the field by 1 each time a new record is added
    - 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

Create Tables

- Each table should have a primary key column (in this case: the "id" column). Its value must be unique for each record in the table.

```
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 DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
)
```

Create Tables

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

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
```

Create Tables

```php
// 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 DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
)";

if ($conn->query($sql) === TRUE) {
    echo "Table MyGuests created successfully";
} else {
    echo "Error creating table: " . $conn->error;
}

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

Insert Data

- After a database and a table have been created, we can start adding data in them.
- Here are some syntax rules to follow:
  - 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

Insert Data

- 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` with default update of current_timesamp (like the "reg_date" column), it is no need to be specified in the SQL query; MySQL will automatically add the value.

Insert Data

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

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
```

Insert Data

```php
$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if ($conn->query($sql) === TRUE) {
    echo "New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}

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

Get ID of Last Inserted Record

- 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.
- In the table "MyGuests", the "id" column is an AUTO_INCREMENT field:

```
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 DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
)
```

- The following examples are equal to the examples from the previous example, except that we have added one single line of code to retrieve the ID of the last inserted record. We also echo the last inserted ID:

Get ID of Last Inserted Record

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

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
```

Get ID of Last Inserted Record

```
$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if ($conn->query($sql) === TRUE) {
    $last_id = $conn->insert_id;
    echo "New record created successfully. Last inserted ID is:
" . $last_id;
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}

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

Insert Multiple Records

- Multiple SQL statements must be executed with the `mysqli_multi_query()` function.
- The following examples add three new records to the "MyGuests" table:

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

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
```

Insert Multiple Records

```
$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Mary', 'Moe', 'mary@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Julie', 'Dooley', 'julie@example.com')";

if ($conn->multi_query($sql) === TRUE) {
    echo "New records created successfully";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}

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

Prepared Statements

- Prepared statements are very useful against SQL injections.
- 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

Prepared Statements

- Compared to executing SQL statements directly, prepared statements have three main advantages:
  - *Prepared statements reduce parsing time as the preparation on the query is done only once (although the statement is executed multiple times)*
  - *Bound parameters minimize bandwidth to the server as you need send only the parameters each time, and not the whole query*
  - *Prepared statements are very useful against SQL injections, because parameter values, which are transmitted later using a different protocol, need not be correctly escaped. If the original statement template is not derived from external input, SQL injection cannot occur.*

Prepared Statements

- Prepared Statements in MySQLi

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

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// prepare and bind
$stmt = $conn->prepare("INSERT INTO MyGuests (firstname,
lastname, email) VALUES (?, ?, ?)");
$stmt->bind_param("sss", $firstname, $lastname, $email);
```

Prepared Statements

```
// set parameters and execute
$firstname = "John";
$lastname = "Doe";
$email = "john@example.com";
$stmt->execute();

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

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


echo "New records created successfully";

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

Prepared Statements

- Code lines to explain from the example:

```
INSERT INTO MyGuests (firstname, lastname, email) VALUES (?, ?, ?)
```

- In our SQL, we insert a question mark (?) where we want to substitute in an integer, string, double or blob value.
- Then, have a look at the bind_param() function:

```
$stmt->bind_param("sss", $firstname, $lastname, $email);
```

- This function binds the parameters to the SQL query and tells the database what the parameters are. The "sss" argument lists the types of data that the parameters are. The s character tells mysql that the parameter is a string.

Prepared Statements

- The argument may be one of four types:
  - i - integer
  - d - double
  - s - string
  - b - BLOB
- We must have one of these for each parameter.
- By telling mysql what type of data to expect, we minimize the risk of SQL injections.

⚠ If we want to insert any data from external sources (like user input), it is very important that the data is sanitized and validated.

Select Data From MySQL

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

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

- or we can use the * character to select ALL columns from a table:

```
SELECT * FROM table_name
```

- In DB course, you have learned similar queries
- As you know, you can specify the condition(s) for SELECT query:

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

Select Data From MySQL

- The following example selects the id, firstname and lastname columns from the MyGuests table and displays it on the page:(MySQLi)

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

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
```

Select Data From MySQL

```
$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
    // output data of each row
    while($row = $result->fetch_assoc()) {
        echo "id: " . $row["id"]. " - Name: " .
$row["firstname"]. " " . $row["lastname"]. "<br>";
    }
} else {
    echo "0 results";
}
$conn->close();
?>
```

Delete Data From MySQL Table

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

```
DELETE FROM table_name
WHERE some_column = some_value
```

Notice the WHERE clause in the DELETE syntax: The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

- Let's look at the "MyGuests" table:

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Moe	mary@example.com	2014-10-23 10:22:30
3	Julie	Dooley	julie@example.com	2014-10-26 10:48:23

Select Data From MySQL

- The following examples delete the record with id=3 in the "MyGuests" table:

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

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
```

Select Data From MySQL

```php
// sql to delete a record
$sql = "DELETE FROM MyGuests WHERE id=3";

if ($conn->query($sql) === TRUE) {
    echo "Record deleted successfully";
} else {
    echo "Error deleting record: " . $conn->error;
}

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

- After the record is deleted, the table will look like this:

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Moe	mary@example.com	2014-10-23 10:22:30

Update Data In a MySQL Table

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

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

⚠ Notice the WHERE clause in the UPDATE syntax: The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

- Let's look at the "MyGuests" table:

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Moe	mary@example.com	2014-10-23 10:22:30

Select Data From MySQL

- The following examples update the record with id=2 in the "MyGuests" table:

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

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
```

Select Data From MySQL

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

if ($conn->query($sql) === TRUE) {
    echo "Record updated successfully";
} else {
    echo "Error updating record: " . $conn->error;
}

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

- After the record is updated, the table will look like this:

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Doe	mary@example.com	2014-10-23 10:22:30

Limit Data Selections From a MySQL Database

- MySQL provides a LIMIT clause that is used to specify the number of records to return.
- The LIMIT clause makes it easy to code multi page results or pagination with SQL, and is very useful on large tables. Returning a large number of records can impact on performance.
- Assume we wish to select all records from 1 - 30 (inclusive) from a table called "Orders". The SQL query would then look like this:

```
$sql = "SELECT * FROM Orders LIMIT 30";
```

Limit Data Selections From a MySQL Database

- When the SQL query above is run, it will return the first 30 records.
- What if we want to select records 16 - 25 (inclusive)?
- Mysql also provides a way to handle this: by using OFFSET.
- The SQL query below says "return only 10 records, start on record 16 (OFFSET 15)":

```
$sql = "SELECT * FROM Orders LIMIT 10 OFFSET 15";
```

- You could also use a shorter syntax to achieve the same result:

```
$sql = "SELECT * FROM Orders LIMIT 15, 10";
```

- Notice that the numbers are reversed when you use a comma