PHP MySQL Insert Data

Insert Data Into MySQL Using MySQLi and PDO

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

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,...)

To learn more about SQL, please visit our SQL tutorial.

In the previous chapter we created an empty table named "MyGuests" with five columns: "id", "firstname", "lastname", "email" and "reg_date". Now, let us fill the table with data.

Note: 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.

The following examples add a new record to the "MyGuests" table:

Example (MySQLi Object-oriented)

```
<?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);
}
```

```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();
?>
```

Example (MySQLi Procedural)

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

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

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

if (mysqli_query($conn, $sql)) {
```

```
  echo "New record created successfully";
} else {
  echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Example (PDO)

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

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);
  echo "New record created successfully";
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

PHP MySQL Get Last Inserted ID

Get ID of The 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 page (PHP Insert Data Into MySQL), 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:

Example (MySQLi Object-oriented)

```
<?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);
}

$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();
?>
```

Example (MySQLi Procedural)

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

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}
$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if (mysqli_query($conn, $sql)) {
```

```
  $last_id = mysqli_insert_id($conn);
  echo "New record created successfully. Last inserted ID is: " . $last_id;
} else {
  echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Example (PDO)

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

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;
?>
```

PHP MySQL Insert Multiple Records

Insert Multiple Records Into MySQL Using MySQLi and PDO

Multiple SQL statements must be executed with the `mysqli_multi_query()` function.

The following examples add three new records to the "MyGuests" table:

Example (MySQLi Object-oriented)

```
<?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);
}

$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();
?>
```

Note that each SQL statement must be separated by a semicolon.

Example (MySQLi Procedural)

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

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

$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 (mysqli_multi_query($conn, $sql)) {
  echo "New records created successfully";
} else {
  echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}
```

```
mysqli_close($conn);
?>
```

The PDO way is a little bit different:

Example (PDO)

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

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();
}

$conn = null;
?>
```

PHP MySQL Prepared Statements

Prepared statements are very useful against SQL injections.

Prepared Statements and Bound Parameters

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

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 in MySQLi

The following example uses prepared statements and bound parameters in MySQLi:

Example (MySQLi with Prepared Statements)

```
<?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);

// 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();
?>
```

Code lines to explain from the example above:

"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.

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.

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

Prepared Statements in PDO

The following example uses prepared statements and bound parameters in PDO:

Example (PDO with Prepared Statements)

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

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);

  // 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();

  echo "New records created successfully";
} catch(PDOException $e) {
  echo "Error: " . $e->getMessage();
}
$conn = null;
?>
```