

Unit – 4

PHP Connect to MySQL Database

Introduction

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

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.

Open a Connection to MySQL

- Open a Connection to MySQL
- Before we can access data in the MySQL database, we need to be able to connect to the server
- PHP provides **mysql_connect** function to open a database connection.

mysqli_connect()

- The mysqli_connect() function opens a new connection to the MySQL server.

Syntax

- `mysqli_connect (host, username, password, dbname, port, socket);`

Parameter	Description
<i>host</i>	Optional. Specifies a host name or an IP address
<i>username</i>	Optional. Specifies the MySQL username
<i>password</i>	Optional. Specifies the MySQL password
<i>dbname</i>	Optional. Specifies the default database to be used
<i>port</i>	Optional. Specifies the port number to attempt to connect to the MySQL server
<i>socket</i>	Optional. Specifies the socket or named pipe to be used

Example (MySQLi Procedural)

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

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

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


mysqli_connect_error()

- The mysqli_connect_error() function returns the error description from the last connection error, if any.
- Syntax

`mysqli_connect_error();`

Return Value:	Returns a string that describes the error. NULL if no error occurred
---------------	--

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);`

PHP Create a MySQL Database

- A database consists of one or more tables.
- Create a MySQL Database Using MySQLi
- The CREATE DATABASE statement is used to create a database in MySQL.
- PHP uses **mysql_query** function to create a MySQL database.
- This function takes two parameters and **returns TRUE on success or FALSE on failure.**
 - *bool mysql_query(sql, connection);*
- When you create a new database, you must only specify the first three arguments to the mysqli_connect(**servername, username and password**).

mysqli_query() Function

- The mysqli_query() function performs a query against the database.

Syntax

mysqli_query (*connection, query, resultmode*);

Parameter	Description
<i>connection</i>	Required. Specifies the MySQL connection to use
<i>query</i>	Required. Specifies the query string
<i>resultmode</i>	Optional. A constant. Either: <ul style="list-style-type: none">•MYSQLI_USE_RESULT (Use this if we have to retrieve large amount of data)•MYSQLI_STORE_RESULT (This is default)

Example (MySQLi Procedural)

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

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

// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

// Create database
$sql = "CREATE DATABASE myDB";
if (mysqli_query($conn, $sql)) {
    echo "Database created successfully";
}
else {
    echo "Error creating database: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Selecting a Database

- Once you establish a connection with a database server then it is required to select a particular database where your all the tables are associated.
- This is required because there may be multiple databases residing on a single server and you can do work with a single database at a time.
- PHP provides function **mysql_select_db** to select a database. It returns **TRUE on success or FALSE on failure.**
 - *bool mysql_select_db (connection, db_name);*

Deleting a Database

- If a database is no longer required then it can be deleted forever. You can use pass an SQL command to `mysql_query` to delete a database.

`<?php`

```
$dbhost = 'localhost';  
$dbuser = 'root';  
$dbpass = 'root';  
$conn = mysqli_connect($dbhost, $dbuser, $dbpass);  
if(! $conn ) {  
    die('Could not connect: ' . mysql_error());  
}  
$sql = 'DROP DATABASE myDB';  
$retval = mysqli_query( $sql, $conn );  
if(! $retval ) {  
    die('Could not delete database db_test: ' . mysql_error());  
}  
echo "Database deleted successfully\n";  
mysqli_close($conn);
```

`?>`

PHP Create MySQL 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 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
)
- First create the SQL query to create the tables then execute the query using **mysql_query()** function.

- 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:
 - **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
- 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.

Example (MySQLi Procedural)

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

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

// 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
)";

if (mysqli_query($conn, $sql)) {
    echo "Table MyGuests created successfully";
} else {
    echo "Error creating table: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```


PHP Insert Data Into MySQL

- 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,...)

- Data can be entered into MySQL tables by executing SQL INSERT statement through PHP function **mysql_query**

Example (MySQLi Procedural)

- ```
<?php
$servername = "localhost";
$username = "root";
$password = "root";
$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: " . $sql . "
" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```



# PHP 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
)
```

# mysqli\_insert\_id();

- The mysqli\_insert\_id() function returns the id (generated with AUTO\_INCREMENT) used in the last query.

## Syntax

- `mysqli_insert_id(connection);`

| Parameter         | Description                                     |
|-------------------|-------------------------------------------------|
| <i>connection</i> | Required. Specifies the MySQL connection to use |

|                      |                                                                                                                                                                                                                                  |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Return Value:</b> | Returns an integer with the value of the AUTO_INCREMENT field that was updated by the last query. If the number is > max integer value, it will return a string. Returns zero if there were no update or no AUTO_INCREMENT field |
|                      |                                                                                                                                                                                                                                  |



# Example (MySQLi Procedural)

```
• <?php
$servername = "localhost";
$username = "root";
$password = "root";
$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: " . $sql . "
" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

# PHP Insert Multiple Records Into MySQL

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



# Example (MySQLi Procedural)

- ```
<?php
$servername = "localhost";
$username = "root";
$password = "root";
$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);  
?>
```


mysqli_multi_query()

- The mysqli_multi_query() function performs one or more queries against the database. The queries are separated with a semicolon.
- **Syntax**

mysqli_multi_query (*connection*, *query*);

Parameter	Description
<i>connection</i>	Required. Specifies the MySQL connection to use
<i>query</i>	Required. Specifies one or more queries, separated with semicolon

Return Value:	FALSE if the first query fails
---------------	--------------------------------

PHP 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:

- **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(?, ?, ?)`
- The database **parses, compiles, and performs query optimization on the SQL statement** template, and stores the result without executing it
- **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 reduces 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.

Example (Procedural)

```
<?php
```

```
$link = mysqli_connect("localhost", "root", "passphp", "myDB");
```

```
// Check connection
```

```
if($link === false){
```

```
    die("ERROR: Could not connect. " . mysqli_connect_error());
```

```
}
```

```
// Prepare an insert statement
```

```
$sql = "INSERT INTO MyGuests (firstname, lastname, email) VALUES (?, ?, ?)";
```

```
if($stmt = mysqli_prepare($link, $sql))//Prepare an SQL statement for execution
```

```
{
```

```
    // Bind variables to the prepared statement as parameters
```

```
    mysqli_stmt_bind_param($stmt, "sss", $firstname, $lastname, $email);
```



```
/* Set the parameters values and execute
the statement to insert a row */
$firstname = "Yamaguchi";
$lastname = "San";
$email = "yamaguchi@mail.com";
mysqli_stmt_execute($stmt);

echo "Records inserted successfully.";
} else{
    echo "ERROR: Could not prepare query: $sql. " . mysqli_error($link);
}

// Close statement
mysqli_stmt_close($stmt);

// Close connection
mysqli_close($link);
?>
```

mysqli_stmt_bind_param()

Binds variables to a prepared statement as parameters

Syntax:

```
bool mysqli_stmt_bind_param ( $stmt , $types , $var )
```

Parameters

- **Stmt:** A statement identifier returned by **mysqli_prepare()**
- **Types:** A string that contains one or more characters which specify the types for the corresponding bind variables

Character	Description
i	corresponding variable has type integer
d	corresponding variable has type double
s	corresponding variable has type string
b	corresponding variable is a blob and will be sent in packets

Var: The number of variables and length of string types must match the parameters in the statement.

Return Values : Returns TRUE on success or FALSE on failure.

PHP Select Data From MySQL

- The 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

- First, we set up an SQL query that selects the id, firstname and lastname columns from the MyGuests table. The next line of code runs the query and puts the resulting data into a variable called \$result.
- Then, the function **num_rows()** checks if there are more than zero rows returned.
- If there are more than zero rows returned, the function **fetch_assoc()** puts all the results into an associative array that we can loop through. The while() loop loops through the result set and outputs the data from the id, firstname and lastname columns.

Example (MySQLi Procedural)

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

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

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = mysqli_query($conn, $sql);

if (mysqli_num_rows($result) > 0) {
 // output data of each row
 while($row = mysqli_fetch_assoc($result)) {
 echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "
";
 }
} else {
 echo "o results";
}

mysqli_close($conn);
?>
```



# PHP Delete Data From MySQL

- The DELETE statement is used to delete records from a table:
- **DELETE FROM table\_name  
WHERE some\_column = some\_value**
- Lets look at 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 |

# Example (MySQLi Procedural)

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

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

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

if (mysqli_query($conn, $sql)) {
    echo "Record deleted successfully";
} else {
    echo "Error deleting record: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```


PHP Update Data in MySQL

- Data can be updated into MySQL tables by executing SQL UPDATE statement through PHP function **mysql_query**.
- **UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value**
- **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!

•	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

Example (MySQLi Procedural)

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

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

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

if (mysqli_query($conn, $sql)) {
 echo "Record updated successfully";
} else {
 echo "Error updating record: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```



# PHP Limit Data Selections From MySQL

- 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";
```

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";
```

# Open a Connection to MySQL

- Example (MySQLi Object-Oriented)

```
<?php
```

```
$servername = "localhost";
```

```
$username = "root";
```

```
$password = "root";
```

```
// Create connection
```

```
$conn = new mysqli($servername, $username,
$password);
```

```
// Check connection
```

```
if ($conn->connect_error) {
 die("Connection failed: " . $conn->connect_error);
}
```

```
echo "Connected successfully";
```

```
?>
```



# Create Database Example (MySQLi Object-Oriented)

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

// Create connection
$conn = new mysqli($servername, $username, $password);
// Check connection
if ($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();
?>
```

Create Table Example(MySQLi Object Oriented)

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

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



- // 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  
)";

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

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

# Insert into TableExample (MySQLi Object-oriented)

- ```
<?php
$servername = "localhost";
$username = "root";
$password = "root";
$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) {
    echo "New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}

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


Multiple Insertions Example (MySQLi Object-oriented)

- ```
<?php
$servername = "localhost";
$username = "root";
$password = "root";
$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 . "
" . $conn->error;
}

$conn->close();
?>
```
- Note: Note that each SQL statement must be separated by a semicolon.

# Example Selection from table (MySQLi Object-oriented)

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

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

$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 "o results";
}
$conn->close();
?>
```


Deleting records Example (MySQLi Object-oriented)

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

- <?php

```
$servername = "localhost";  
$username = "root";  
$password = "root";  
$dbname = "myDB";
```

```
// Create connection
```

```
$conn = new mysqli($servername, $username, $password, $dbname);
```

```
// Check connection
```

```
if ($conn->connect_error) {  
    die("Connection failed: " . $conn->connect_error);  
}
```

```
// 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();
```

```
?>
```

Updating Records Example (MySQLi Object-oriented)

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

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

  $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();
?>
```

•

Example (MySQLi with Prepared Statements)


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