# PHP MySQL Connect

Since PHP 5.5, **mysql_connect()** extension is *deprecated*. Now it is recommended to use one of the 2 alternatives.

- **mysqli_connect()**
- **PDO::__construct()**

---

## PHP mysqli_connect()

PHP **mysqli_connect() function** is used to connect with MySQL database. It returns *resource* if connection is established or *null*.

**Syntax**

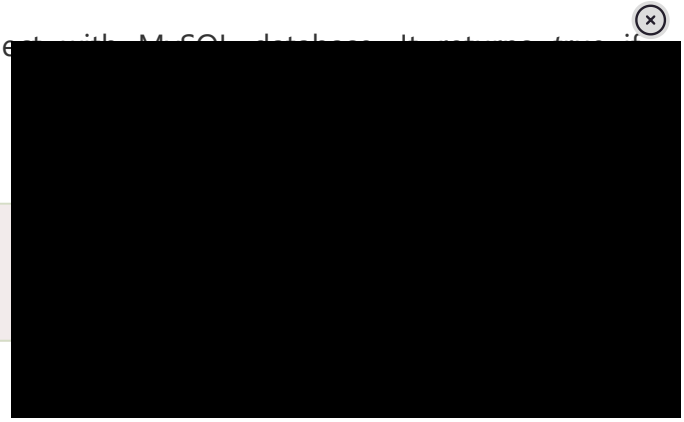
```
resource mysqli_connect (server, username, password)
```

## PHP mysqli_close()

PHP **mysqli_close() function** is used to disconnect with MySQL database. It returns *true* if connection is closed or *false*.

**Syntax**

```
bool mysqli_close(resource $resource_link)
```



# PHP MySQL Connect Example

**Example**

```php
<?php
$host = 'localhost:3306';
$user = '';
$pass = '';
$conn = mysqli_connect($host, $user, $pass);
if(! $conn )
{
  die('Could not connect: ' . mysqli_error());
}
echo 'Connected successfully';
mysqli_close($conn);
?>
```

Output:

```
Connected successfully
```

# PHP MySQL Create Database

Since PHP 4.3, **mysql_create_db()** function is *deprecated*. Now it is recommended to use one of the 2 alternatives.

- **mysqli_query()**
- **PDO::__query()**

## PHP MySQLi Create Database Example

**Example**

```
<?php
$host = 'localhost:3306';
$user = '';
$pass = '';
$conn = mysqli_connect($host, $user, $pass);
if(! $conn )
{
  die('Could not connect: ' . mysqli_connect_error());
}
echo 'Connected successfully<br/>';

$sql = 'CREATE Database mydb';
if(mysqli_query( $conn,$sql)){
  echo "Database mydb created successfully.";
}else{
echo "Sorry, database creation failed ".mysqli_error($conn);
```

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

Output:

```
Connected successfully
Database mydb created successfully.
```

← Prev

Next →

# PHP MySQL Create Table

PHP mysql_query() function is used to create table. Since PHP 5.5, **mysql_query()** function is *deprecated*. Now it is recommended to use one of the 2 alternatives.

- **mysqli_query()**
- **PDO::__query()**

## PHP MySQLi Create Table Example

**Example**

```
<?php
$host = 'localhost:3306';
$user = '';
$pass = '';
$dbname = 'test';

$conn = mysqli_connect($host, $user, $pass,$dbname);
if(!$conn){
  die('Could not connect: '.mysqli_connect_error());
}
echo 'Connected successfully<br/>';

$sql = "create table emp5(id INT AUTO_INCREMENT,name VARCHAR(20) NOT NULL,
emp_salary INT NOT NULL,primary key (id))";
if(mysqli_query($conn, $sql)){
 echo "Table emp5 created successfully";
```

```
}else{
echo "Could not create table: ". mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Output:

```
Connected successfully
Table emp5 created successfully
```

← Prev

Next →



## Feedback

- Send your Feedback to feedback@javatpoint.com

## Help Others, Please Share

## Learn Latest Tutorials

Splunk tutorial
Splunk

SPSS tutorial
SPSS

Swagger tutorial
Swagger

T-SQL tutorial
Transact-SQL

Tumblr tutorial
Tumblr

React tutorial
ReactJS

Regex tutorial
Regex

Reinforcement learning tutorial
Reinforcement Learning

# PHP MySQL Insert Record

PHP mysql_query() function is used to insert record in a table. Since PHP 5.5, **mysql_query()** function is *deprecated*. Now it is recommended to use one of the 2 alternatives.

- **mysqli_query()**
- **PDO::__query()**

## PHP MySQLi Insert Record Example

**Example**

```
<?php
$host = 'localhost:3306';
$user = '';
$pass = '';
$dbname = 'test';

$conn = mysqli_connect($host, $user, $pass,$dbname);
if(!$conn){
  die('Could not connect: '.mysqli_connect_error());
}
echo 'Connected successfully<br/>';

$sql = 'INSERT INTO emp4(name,salary) VALUES ("sonoo", 9000)';
if(mysqli_query($conn, $sql)){
 echo "Record inserted successfully";
}else{
```

```
echo "Could not insert record: ". mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Output:

```
Connected successfully
Record inserted successfully
```

← Prev

Next →

# PHP MySQL Get Last Inserted ID

‹ Previous Next ›

## 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)

Get your own PHP Server

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

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

```
$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());
}
```

```
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

❮ Previous Next ❯

## 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)

Get your own PHP Server

```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);
}

$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 {
```

```
?>
```

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);
?>
```

Example (PDO)

```php
<?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

❮ Previous Next ❯

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:

```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);

// 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:

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)

```
$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;
?>
```

# PHP MySQL Update Record

PHP mysql_query() function is used to update record in a table. Since PHP 5.5, **mysql_query()** function is *deprecated*. Now it is recommended to use one of the 2 alternatives.

- **mysqli_query()**
- **PDO::__query()**

## PHP MySQLi Update Record Example

**Example**

```
<?php
$host = 'localhost:3306';
$user = '';
$pass = '';
$dbname = 'test';

$conn = mysqli_connect($host, $user, $pass,$dbname);
if(!$conn){
  die('Could not connect: '.mysqli_connect_error());
}
echo 'Connected successfully<br/>';

$id=2;
$name="Rahul";
$salary=80000;
$sql = "update emp4 set name=\"$name\", salary=$salary where id=$id";
```

```
if(mysqli_query($conn, $sql)){
 echo "Record updated successfully";
}else{
echo "Could not update record: ". mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Output:

```
Connected successfully
Record updated successfully
```

← Prev

Next →



## Feedback

- Send your Feedback to feedback@javatpoint.com

## Help Others, Please Share

## Learn Latest Tutorials

Splunk tutorial
Splunk

SPSS tutorial
SPSS

Swagger tutorial
Swagger

T-SQL tutorial
Transact-SQL

Tumblr tutorial
Tumblr

React tutorial
ReactJS

Regex tutorial
Regex

Reinforcement learning tutorial

# PHP MySQL Delete Record

PHP mysql_query() function is used to delete record in a table. Since PHP 5.5, **mysql_query()** function is *deprecated*. Now it is recommended to use one of the 2 alternatives.

- **mysqli_query()**
- **PDO::__query()**

## PHP MySQLi Delete Record Example

**Example**

```
<?php
$host = 'localhost:3306';
$user = '';
$pass = '';
$dbname = 'test';

$conn = mysqli_connect($host, $user, $pass,$dbname);
if(!$conn){
  die('Could not connect: '.mysqli_connect_error());
}
echo 'Connected successfully<br/>';

$id=2;
$sql = "delete from emp4 where id=$id";
if(mysqli_query($conn, $sql)){
 echo "Record deleted successfully";
```

```
}else{
echo "Could not deleted record: ". mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Output:

```
Connected successfully
Record deleted successfully
```

← Prev

Next →



## Feedback

- Send your Feedback to feedback@javatpoint.com

## Help Others, Please Share

## Learn Latest Tutorials

Splunk tutorial
Splunk

SPSS tutorial
SPSS

Swagger tutorial
Swagger

T-SQL tutorial
Transact-SQL

Tumblr tutorial
Tumblr

React tutorial
ReactJS

Regex tutorial
Regex

Reinforcement learning tutorial
Reinforcement Learning

# PHP MySQL Select Query

PHP mysql_query() function is used to execute select query. Since PHP 5.5, **mysql_query()** function is *deprecated*. Now it is recommended to use one of the 2 alternatives.

- **mysqli_query()**
- **PDO::__query()**

There are two other MySQLi functions used in select query.

- **mysqli_num_rows(mysqli_result $result)**: returns number of rows.
- **mysqli_fetch_assoc(mysqli_result $result)**: returns row as an associative array. Each key of the array represents the column name of the table. It return NULL if there are no more rows.

# PHP MySQLi Select Query Example

**Example**

```
<?php
$host = 'localhost:3306';
$user = '';
$pass = '';
$dbname = 'test';
$conn = mysqli_connect($host, $user, $pass,$dbname);
if(!$conn){
  die('Could not connect: '.mysqli_connect_error());
}
```

```
echo 'Connected successfully<br/>';

$sql = 'SELECT * FROM emp4';
$retval=mysqli_query($conn, $sql);

if(mysqli_num_rows($retval) > 0){
 while($row = mysqli_fetch_assoc($retval)){
    echo "EMP ID :{$row['id']}  <br> ".
        "EMP NAME : {$row['name']} <br> ".
        "EMP SALARY : {$row['salary']} <br> ".
        "--------------------------------<br>";
 } //end of while
}else{
echo "0 results";
}
mysqli_close($conn);
?>
```

Output:

```
Connected successfully
EMP ID :1
EMP NAME : ratan
EMP SALARY : 9000
--------------------------------
EMP ID :2
EMP NAME : karan
EMP SALARY : 40000
--------------------------------
EMP ID :3
EMP NAME : jai
EMP SALARY : 90000
--------------------------------
```

← Prev

Next →

# PHP MySQL Order By

PHP mysql_query() function is used to execute select query with order by clause. Since PHP 5.5, **mysql_query()** function is *deprecated*. Now it is recommended to use one of the 2 alternatives.

- **mysqli_query()**
- **PDO::__query()**

The order by clause is used to fetch data in ascending order or descending order on the basis of column.

Let's see the query to select data from emp4 table in ascending order on the basis of name column.

```
SELECT * FROM emp4 order by name
```

Let's see the query to select data from emp4 table in descending order on the basis of name column.

```
SELECT * FROM emp4 order by name desc
```

## PHP MySQLi Order by Example

**Example**

```
<?php
$host = 'localhost:3306';
$user = '';
$pass = '';
$dbname = 'test';
$conn = mysqli_connect($host, $user, $pass,$dbname);
if(!$conn){
  die('Could not connect: '.mysqli_connect_error());
}
echo 'Connected successfully<br/>';

$sql = 'SELECT * FROM emp4 order by name';
$retval=mysqli_query($conn, $sql);

if(mysqli_num_rows($retval) > 0){
 while($row = mysqli_fetch_assoc($retval)){
    echo "EMP ID :{$row['id']}  <br> ".
         "EMP NAME : {$row['name']} <br> ".
         "EMP SALARY : {$row['salary']} <br> ".
         "--------------------------------<br>";
 } //end of while
}else{
echo "0 results";
}
mysqli_close($conn);
?>
```

# PHP MySQL Limit Data Selections

❮ Previous Next ❯

## 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";
```

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.

❮ Previous Log in to track progress Next ❯