How to Use PHP Improved MySQLi extension (and Why You Should)

March 29th, 2013 239,927

PHP supports MySQL using a PHP extension. Thousands or millions projects have been written worldwide using PHP and MySQL. PHP team announced plans to deprecate **MySQL extension** in **mid 2011**. Old MySQL extension **officially deprecated** since PHP 5.5.0 in late 2012 and it will be removed in the future. The alternatives since PHP 5 and later are **MySQLi** ("i" stands from "improved") and **PDO** (PHP Data Objects).

Everyone can understand that is not so easy to immediately migrate old projects. However, the old extension **must not be used anymore in new development**.

Old extension didn't support Prepared Statements. Both MySQLi and PDO are object oriented and do support Prepared Statements (also support Transactions, Stored Procedures and more). Prepared Statements are very important for web application security, as they protect from SQL injection. Using Prepared Statements you do not have to escape strings before insert them in Database. Moreover, PDO offers support for many databases (not

About the author

only MySQL).

So, the question is "Which Should I Use? MySQLi or PDO?". A short answer would be "whatever you like". Personally, I prefer MySQLi. I could select PDO if "multiple database support" was included in project requirements (however, in this case **php ADODB** could also be an alternative).

Below I describe the common use of MySQLi in php development with **MySQL** database (it can be also used with **MariaDB**, an enhanced, drop-in replacement for MySQL).

Installation

MySQLi extension is automatically installed in most cases (Linux or Windows), when php5 mysql package is installed.

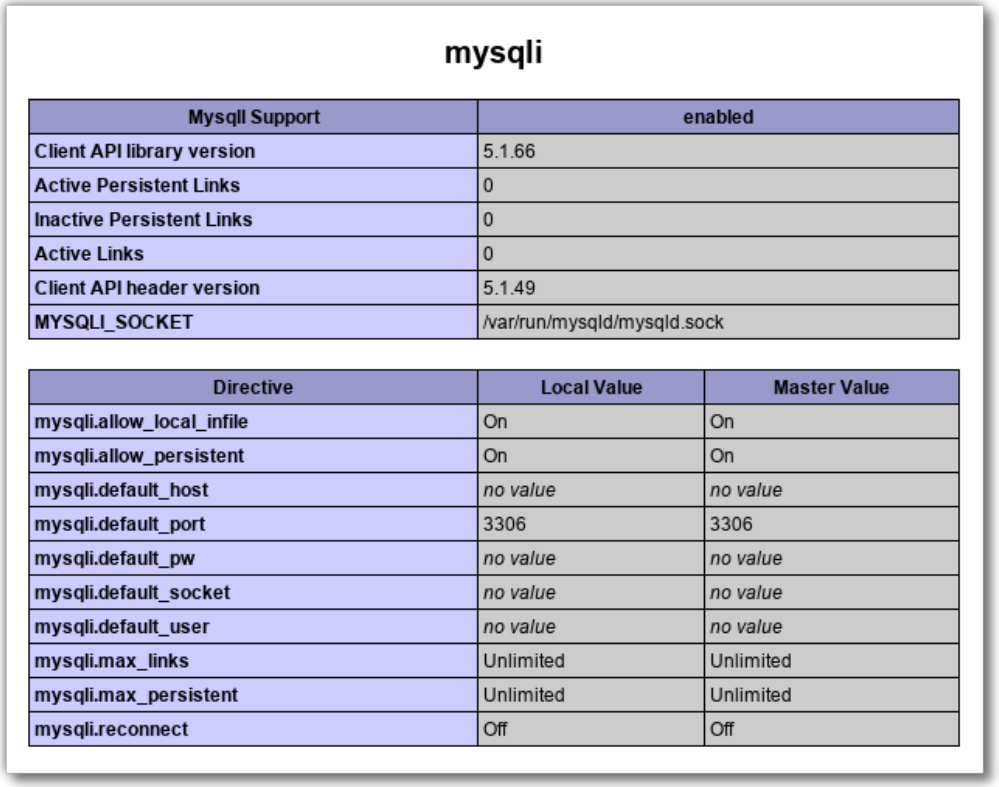
In a Debian (or Ubuntu) server, the following command is enough:

```
apt-get install php5-mysql
```

In a Centos (or Red Hat) server:

```
yum install php-mysql
```

Detailed instructions for any operation system are available **here**. The result in *phpinfo.php* must be something like:



mysqli

Mysqli Support	enabled
Client API library version	5.1.66
Active Persistent Links	0
Inactive Persistent Links	0
Active Links	0
Client API header version	5.1.49
MYSQLI_SOCKET	/var/run/mysqld/mysqld.sock

Directive	Local Value	Master Value
mysqli.allow_local_infile	On	On
mysqli.allow_persistent	On	On
mysqli.default_host	*no value*	*no value*
mysqli.default_port	3306	3306
mysqli.default_pw	*no value*	*no value*
mysqli.default_socket	*no value*	*no value*
mysqli.default_user	*no value*	*no value*
mysqli.max_links	Unlimited	Unlimited
mysqli.max_persistent	Unlimited	Unlimited
mysqli.reconnect	Off	Off

Connect

Define connection parameters:

```
1  $DBServer = 'server name or IP address'; // e.g 'local
2  $DBUser   = 'DB_USER';
3  $DBPass   = 'DB_PASSWORD';
4  $DBName   = 'DB_NAME';
```

Connection using the object oriented way (RECOMMENDED).

```php
$conn = new mysqli($DBServer, $DBUser, $DBPass, $DBNam

// check connection
if ($conn->connect_error) {
  trigger_error('Database connection failed: '  . $con
}
```

Connection using the procedural way (NOT RECOMMENDED).

MySQLi also offers a procedural API, except the object-oriented API. Procedural API might be easier for newcomers to understand, as it is similar with the old PHP MySQL extension API. Here is an example:

```php
$conn = mysqli_connect($DBServer, $DBUser, $DBPass, $D

// check connection
if (mysqli_connect_errno()) {
  trigger_error('Database connection failed: '  . mysq
}
```

(I will not quote further examples using the proceduaral API, as I recommend to use the object-oriented API. Of course it is available in PHP **MySQLi documentation**). A useful summary is also **available**.

Select

Use the following syntax:

```php
$sql='SELECT col1, col2, col3 FROM table1 WHERE condit

$rs=$conn->query($sql);

```

```
if($rs === false) {
  trigger_error('Wrong SQL: ' . $sql . ' Error: ' . $c
} else {
  $rows_returned = $rs->num_rows;
}
```

Iterate over recordset

Using column names - recommended

```
$rs->data_seek(0);
while($row = $rs->fetch_assoc()){
    echo $row['col1'] . '<br>';
}
```

Using column index

```
$rs->data_seek(0);
while($row = $rs->fetch_row()){
    echo $row[0] . '<br>';
}
```

Store all values to array

```
$rs=$conn->query($sql);

if($rs === false) {
  trigger_error('Wrong SQL: ' . $sql . ' Error: ' . $
} else {
  $arr = $rs->fetch_all(MYSQLI_ASSOC);
}
foreach($arr as $row) {
  echo $row['co1'];
}
```

Using *MYSQLI_ASSOC* an associated array is returned, *MYSQLI_NUM* an enumerated one and *MYSQLI_BOTH* both of them.

WARNING: **fetch_all** is available only with **MySQL Native Driver**.

Store row values to array

The following example will return an array with first row (using `$rs->data_seek(n);` we can get any row).

```
$rs=$conn->query($sql);

if($rs === false) {
  trigger_error('Wrong SQL: ' . $sql . ' Error: ' . $c
} else {
  $rs->data_seek(0);
  $arr = $rs->fetch_array(MYSQLI_ASSOC);
}
```

Record count

```
$rows_returned = $rs->num_rows;
```

Move inside recordset

```
$rs->data_seek(10);
```

Free memory

Optional:

```
$rs->free();
```

Insert

Use the following syntax:

`real_escape_string` is used to escape special characters `NUL (ASCII 0), \n, \r, \, ', ", and Control-Z` in string values before insert to Database (mainly to prevent **SQL injection**).

WARNING: `real_escape_string` does not add quotes, you have to do it manually.

```
$v1="'" . $conn->real_escape_string('col1_value') . "

$sql="INSERT INTO tbl (col1_varchar, col2_number) VAL

if($conn->query($sql) === false) {
  trigger_error('Wrong SQL: ' . $sql . ' Error: ' . $
} else {
  $last_inserted_id = $conn->insert_id;
  $affected_rows = $conn->affected_rows;
}
```

Update

Use the following syntax:

```
$v1="'" . $conn->real_escape_string('col1_value') . "'

$sql="UPDATE tbl SET col1_varchar=$v1, col2_number=1 W

if($conn->query($sql) === false) {
  trigger_error('Wrong SQL: ' . $sql . ' Error: ' . $c
} else {
  $affected_rows = $conn->affected_rows;
}
```

Delete

Use the following syntax:

```
$sql="DELETE FROM tbl WHERE id>10";

if($conn->query($sql) === false) {
  trigger_error('Wrong SQL: ' . $sql . ' Error: ' . $c
} else {
  $affected_rows = $conn->affected_rows;
}
```

Transactions

Use the following syntax:

```
try {
  /* switch autocommit status to FALSE. Actually, it
  $conn->autocommit(FALSE);

  $res = $conn->query($sql1);
  if($res === false) {
    throw new Exception('Wrong SQL: ' . $sql . ' Errc
  }

  $res = $conn->query($sql2);
  if($res === false) {
    throw new Exception('Wrong SQL: ' . $sql . ' Errc
  }

  $res = $conn->query($sql3);
  if($res === false) {
    throw new Exception('Wrong SQL: ' . $sql . ' Errc
  }
```

```
19
20    $conn->commit();
21    echo 'Transaction completed successfully!';
22
23  } catch (Exception $e) {
24
25    echo 'Transaction failed: ' . $e->getMessage();
26    $conn->rollback();
27  }
28
29  /* switch back autocommit status */
30  $conn->autocommit(TRUE);
```

According to **http://www.php.net/manual/en/mysqli.commit.php#89976**, calling `$conn->commit()` will NOT automatically set *autocommit()* back to 'true'. That means that any queries following `$conn->commit()` will be rolled back when your script exits, if *autocommit()* will be not switched back to TRUE.

WARNING: some MySQL statements **cause an implicit commit**, so the cannot be used inside a transaction. For example, you cannot rollback MySQL CREATE TABLE or TRUNCATE TABLE inside a transcaction. A useful comparison is available **here**.

Quoting and escaping strings

You have probably noticed that every string value is escaped before inserted to database as special characters may break SQL and, moreover, to prevent **SQL injection**.

```
1  $safe_string = $conn->real_escape_string($string);
```

Example: `bla"bla\bla` will be converted to `bla\"bla\\bla`.

`real_escape_string` does not add quotes, you have to do it manually.

However, these are not required, if you use Prepared statements (see below).

Prepared statements

What are Prepared Statements and why they are important?

Prepared Statement objects are used with an SQL statement which, typically but not necessary, takes parameters (using the symbol ? in our case or using other placeholders in different DBMS, e.g. $1, $2 etc in PostgreSQL).

After an SQL Statement has been prepared, the DBMS does not have to recompile it and prepare an execution plan. The Database engine simply runs (executes) the statement. This is can optimize performance. Performance advantage is remarkable when a single session is being used to execute a large number of similar statements.

These parameters inside a prepared statement don't need to be escaped and quoted. Driver takes care of this. So, using of Prepared Statements eliminates the possibility of **SQL injection**.

If you're not familiar with the use of Prepared Statements, you

should do it, as it is very important for web applications security.

Connect to database as decribed above.

Select queries

```
$sql='SELECT lastname, email FROM customers WHERE id
$id_greater_than = 5;
$firstname = 'John';

/* Prepare statement */
$stmt = $conn->prepare($sql);
if($stmt === false) {
  trigger_error('Wrong SQL: ' . $sql . ' Error: ' . $
}

/* Bind parameters. TYpes: s = string, i = integer, d
$stmt->bind_param('is',$id_greater_than,$firstname);

/* Execute statement */
$stmt->execute();
```

Iterate over results

```
$stmt->bind_result($lastname, $email);
while ($stmt->fetch()) {
  echo $lastname . ', ' . $email . '<br>';
}
```

Store all values to array

```
$rs=$stmt->get_result();
$arr = $rs->fetch_all(MYSQLI_ASSOC);
```

WARNING: **get_result** is available only with **MySQL Native Driver**.

Close statement

```php
$stmt->close();
```

Insert queries

```php
$sql='INSERT INTO customers (firstname, lastname) VAL
$firstname = 'John';
$lastname = 'Doe';

/* Prepare statement */
$stmt = $conn->prepare($sql);
if($stmt === false) {
  trigger_error('Wrong SQL: ' . $sql . ' Error: ' . $
}

/* Bind parameters. TYpes: s = string, i = integer, d
$stmt->bind_param('ss',$firstname,$lastname);

/* Execute statement */
$stmt->execute();

echo $stmt->insert_id;
echo $stmt->affected_rows;

$stmt->close();
```

Update queries

```php
$sql='UPDATE customers SET firstname = ?, lastname =
$firstname = 'John';
$lastname = 'Doe';
$id_greater_than = 5;

/* Prepare statement */
$stmt = $conn->prepare($sql);
if($stmt === false) {
  trigger_error('Wrong SQL: ' . $sql . ' Error: ' . $
}

/* Bind parameters. TYpes: s = string, i = integer, d
```

```
$stmt->bind_param('ssi',$firstname,$lastname,$id_grea

/* Execute statement */
$stmt->execute();

echo $stmt->affected_rows;

$stmt->close();
```

Delete queries

```
$sql='DELETE FROM customers WHERE id > ?';
$id_greater_than = 5;

/* Prepare statement */
$stmt = $conn->prepare($sql);
if($stmt === false) {
  trigger_error('Wrong SQL: ' . $sql . ' Error: ' . $
}

/* Bind parameters. TYpes: s = string, i = integer, d
$stmt->bind_param('i',$id_greater_than);

/* Execute statement */
$stmt->execute();

echo $stmt->affected_rows;

$stmt->close();
```

Disconnect

Optional:

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

Related Posts

You may also interested in

- **How to Write Code for Any Database with PHP ADODB**

We welcome your feedback. Share your experience with us. Leave us a comment.

Your comments are welcomed!

This site actively encourages commenting on any post. Comments are not pre-moderated, but this community does not tolerate direct or indirect attacks, name-calling or insults. Please, read **terms of use** and Comment Policy at **privacy policy**.

Ghostery 已拦截 Disqus 的评论。

comments powered by Disqus

← How to Collaborate on Github Open Source Projects

Is it time to Remove MySQL in favor of MariaDB in Production Servers? →