php

- Downloads
- Documentation
- Get Involved
- Help

Search

PHPKonf: Istanbul PHP Conference 2017

Keyboard Shortcuts

?	This help
j	Next menu item
k	Previous menu item
g p	Previous man page
g n	Next man page
G	Scroll to bottom
g g	Scroll to top

PDOStatement::execute

(PHP 5 >= 5.1.0, PHP 7, PECL pdo >= 0.1.0)

PDOStatement::execute — Executes a prepared statement

Description ¶

public bool **PDOStatement::execute** ([array `$input_parameters`])

Execute the prepared statement. If the prepared statement included parameter markers, either:

- PDOStatement::bindParam() and/or PDOStatement::bindValue() has to be called to bind either variables or values (respectively) to the parameter markers. Bound variables pass their value as input and receive the output value, if any, of their associated parameter markers

- or an array of input-only parameter values has to be passed

Parameters ¶

`input_parameters`

An array of values with as many elements as there are bound parameters in the SQL statement being executed. All values are treated as **`PDO::PARAM_STR`**.

Multiple values cannot be bound to a single parameter; for example, it is not allowed to bind two values to a single named parameter in an IN() clause.

Binding more values than specified is not possible; if more keys exist in `input_parameters` than in the SQL specified in the PDO::prepare(), then the statement will fail and an error is emitted.

Return Values ¶

Returns **TRUE** on success or **FALSE** on failure.

Changelog ¶

Version	Description
5.2.0	The keys from `input_parameters` must match the ones declared in the SQL. Before PHP 5.2.0 this was silently ignored.

Examples ¶

Example #1 Execute a prepared statement with a bound variable and value

```
<?php
/* Execute a prepared statement by binding a variable and value */
$calories = 150;
$colour = 'gre';
$sth = $dbh->prepare('SELECT name, colour, calories
    FROM fruit
    WHERE calories < :calories AND colour LIKE :colour');
$sth->bindParam(':calories', $calories, PDO::PARAM_INT);
$sth->bindValue(':colour', "%{$colour}%");
$sth->execute();
?>
```

Example #2 Execute a prepared statement with an array of insert values (named parameters)

```
<?php
/* Execute a prepared statement by passing an array of insert values */
$calories = 150;
$colour = 'red';
$sth = $dbh->prepare('SELECT name, colour, calories
    FROM fruit
    WHERE calories < :calories AND colour = :colour');
$sth->execute(array(':calories' => $calories, ':colour' => $colour));
?>
```

Example #3 Execute a prepared statement with an array of insert values (placeholders)

```
<?php
/* Execute a prepared statement by passing an array of insert values */
$calories = 150;
$colour = 'red';
$sth = $dbh->prepare('SELECT name, colour, calories
    FROM fruit
    WHERE calories < ? AND colour = ?');
$sth->execute(array($calories, $colour));
?>
```

Example #4 Execute a prepared statement with question mark placeholders

```
<?php
/* Execute a prepared statement by binding PHP variables */
$calories = 150;
$colour = 'red';
$sth = $dbh->prepare('SELECT name, colour, calories
    FROM fruit
    WHERE calories < ? AND colour = ?');
$sth->bindParam(1, $calories, PDO::PARAM_INT);
$sth->bindParam(2, $colour, PDO::PARAM_STR, 12);
$sth->execute();
?>
```

Example #5 Execute a prepared statement using array for IN clause

```
<?php
/* Execute a prepared statement using an array of values for an IN clause */
$params = array(1, 21, 63, 171);
/* Create a string for the parameter placeholders filled to the number of params */
$place_holders = implode(',', array_fill(0, count($params), '?'));

/*
    This prepares the statement with enough unnamed placeholders for every value
    in our $params array. The values of the $params array are then bound to the
    placeholders in the prepared statement when the statement is executed.
    This is not the same thing as using PDOStatement::bindParam() since this
    requires a reference to the variable. PDOStatement::execute() only binds
    by value instead.
*/
$sth = $dbh->prepare("SELECT id, name FROM contacts WHERE id IN ($place_holders)");
$sth->execute($params);
?>
```

Notes ¶

Note:

Some drivers require to close cursor before executing next statement.

See Also ¶

- PDO::prepare() - Prepares a statement for execution and returns a statement object
- PDOStatement::bindParam() - Binds a parameter to the specified variable name
- PDOStatement::fetch() - Fetches the next row from a result set
- PDOStatement::fetchAll() - Returns an array containing all of the result set rows
- PDOStatement::fetchColumn() - Returns a single column from the next row of a result set

add a note

User Contributed Notes 25 notes

up
down
24

Rami jamleh ¶

3 years ago

```
simplified $placeholder form

<?php

$data = ['a'=>'foo','b'=>'bar'];

$keys = array_keys($data);
$fields = '`'.implode('`, `',$keys).'`';

#here is my way
$placeholder = substr(str_repeat('?,',count($keys),0,-1));

$pdo->prepare("INSERT INTO `baz`($fields) VALUES($placeholder)")->execute(array_values($data));
```

up
down
5

Whitebeard ¶

2 years ago

```
If you are having issues passing boolean values to be bound and are using a Postgres database... but
you do not want to use bindParam for *every* *single* *parameter*, try passing the strings 't' or 'f'
instead of boolean TRUE or FALSE.
```

up
down
3

Clair Shaw ¶

1 year ago

```
Rami's code snippet is a very good solution for using an associated array to insert a line.

However, there's a small, but hard to spot, syntax error which may take a few moments to figure out
what's gone wrong. So I thought I should provide the corrected line:

Instead of:
$placeholder = substr(str_repeat('?,',count($keys),0,-1));

The correct code you'll want to use is:
$placeholder = substr(str_repeat('?,',count($keys)),0,-1);

(Notice that substr has 3 parameters, and str_repeat has 2).
```

up
down
14

VolGas ¶

10 years ago

```
An array of insert values (named parameters) don't need the prefixed colon als key-value to work.

<?php
/* Execute a prepared statement by passing an array of insert values */
$calories = 150;
$colour = 'red';
$sth = $dbh->prepare('SELECT name, colour, calories
   FROM fruit
   WHERE calories < :calories AND colour = :colour');
```

```
// instead of:
//     $sth->execute(array(':calories' => $calories, ':colour' => $colour));
// this works fine, too:
$sth->execute(array('calories' => $calories, 'colour' => $colour));
?>
```

This allows to use "regular" assembled hash-tables (arrays).
That realy does make sense!

up
down
11

gx ¶

6 years ago

Note that you must
- EITHER pass all values to bind in an array to PDOStatement::execute()
- OR bind every value before with PDOStatement::bindValue(), then call PDOStatement::execute() with *no* parameter (not even "array()"!).
Passing an array (empty or not) to execute() will "erase" and replace any previous bindings (and can lead to, e.g. with MySQL, "SQLSTATE[HY000]: General error: 2031" (CR_PARAMS_NOT_BOUND) if you passed an empty array).

Thus the following function is incorrect in case the prepared statement has been "bound" before:

```
<?php
function customExecute(PDOStatement &$sth, $params = NULL) {
    return $sth->execute($params);
}
?>
```

and should therefore be replaced by something like:

```
<?php
function customExecute(PDOStatement &$sth, array $params = array()) {
    if (empty($params))
        return $sth->execute();
    return $sth->execute($params);
}
?>
```

Also note that PDOStatement::execute() doesn't require $input_parameters to be an array.

(of course, do not use it as is ^^).

up
down
5

Ray.Paseur sometimes uses Gmail ¶

11 months ago

"You cannot bind more values than specified; if more keys exist in input_parameters than in the SQL specified in the PDO::prepare(), then the statement will fail and an error is emitted." However fewer keys may not cause an error.

As long as the number of question marks in the query string variable matches the number of elements in the input_parameters, the query will be attempted.

```
This happens even if there is extraneous information after the end of the query string.  The
semicolon indicates the end of the query string; the rest of the variable is treated as a comment by
the SQL engine, but counted as part of the input_parameters by PHP.

Have a look at these two query strings.  The only difference is a typo in the second string, where a
semicolon accidentally replaces a comma.  This UPDATE query will run, will be applied to all rows,
and will silently damage the table.

<?php
/**
* Query is intended to UPDATE a subset of the rows based on the WHERE clause
*/
$sql  = "UPDATE my_table SET fname = ?, lname = ? WHERE id = ?";

/**
* Query UPDATEs all rows, ignoring everything after the semi-colon, including the WHERE clause!
*
* Expected (but not received):
*
*** Warning:
*** PDOStatement::execute():
*** SQLSTATE[HY093]:
*** Invalid parameter number: number of bound variables does not match number of tokens...
*
*/
// Typo here ------------------------ |
//                                    V
$sql  = "UPDATE my_table SET fname = ?; lname = ? WHERE id = ?"; // One token in effect
$pdos = $pdo->prepare($sql);
$pdos->execute( [ 'foo', 'bar', 3 ] );                           // Three input_parameters
?>

PHP 5.4.45, mysqlnd 5.0.10
```

up
down
9
Jean-Lou dot Dupont at jldupont dot com ¶
8 years ago

```
Hopefully this saves time for folks: one should use $count = $stmt->rowCount() after $stmt->execute()
in order to really determine if any an operation such as ' update ' or ' replace ' did succeed i.e.
changed some data.

Jean-Lou Dupont.
```

up
down
9
ElTorqiro ¶
5 years ago

```
When using a prepared statement to execute multiple inserts (such as in a loop etc), under sqlite the
performance is dramatically improved by wrapping the loop in a transaction.

I have an application that routinely inserts 30-50,000 records at a time.  Without the transaction it
was taking over 150 seconds, and with it only 3.
```

```
This may affect other implementations as well, and I am sure it is something that affects all
databases to some extent, but I can only test with PDO sqlite.

e.g.

<?php
$data = array(
  array('name' => 'John', 'age' => '25'),
  array('name' => 'Wendy', 'age' => '32')
);

try {
  $pdo = new PDO('sqlite:myfile.sqlite');
}

catch(PDOException $e) {
  die('Unable to open database connection');
}

$insertStatement = $pdo->prepare('insert into mytable (name, age) values (:name, :age)');

// start transaction
$pdo->beginTransaction();

foreach($data as &$row) {
  $insertStatement->execute($row);
}

// end transaction
$pdo->commit();

?>

[EDITED BY sobak: typofixes by Pere submitted on 12-Sep-2014 01:07]
```

up
down
9
mail at horn-online-media dot de ¶
4 years ago

```
hi,

just a qick note to get started without problems when using quotation: PDO does NOT replace given
variables if they are wrapped in quotationmarks, e.g.

<?php

$st = $db->prepare( '
    INSERT INTO fruits( name, colour )
    VALUES( :name, ":colour" )
';
$st->execute( array( ':name' => 'Apple', ':colour' => 'red' ) );

?>
```

```
results in in a new fruit like

-> Apple, :colour

without the colour beeing replaced by "red". so leave variables WITHOUT the quotation - PDO will do.
```

up
down
6
simon dot lehmann at gmx dot de ¶
9 years ago

```
It seems, that the quoting behaviour has changed somehow between versions, as my current project was
running fine on one setup, but throwing errors on another (both setups are very similar).

Setup 1: Ubuntu 6.10, PHP 5.1.6, MySQL 5.0.24a
Setup 2: Ubuntu 7.04, PHP 5.2.1, MySQL 5.0.38

The code fragment which caused problems (shortened):
<?php
$stmt = $pdo->prepare("SELECT col1, col2, col3 FROM tablename WHERE col4=? LIMIT ?");
$stmt->execute(array('Foo', 1));
?>

On the first Setup this executes without any problems, on the second setup it generates an Error:

SQLSTATE[42000]: Syntax error or access violation: 1064 You have an error in your SQL syntax; check
the manual that corresponds to your MySQL server version for the right syntax to use near ''1'' at
line 1

The problem is, that $stmt->execute() quotes the number passed to the second placeholder (resulting
in: ... LIMIT '1'), which is not allowed in MySQL (tested on both setups).

To prevent this, you have to use bindParam() or bindValue() and specify a data type.
```

up
down
5
albright atat anre dotdot net ¶
8 years ago

```
When passing an array of values to execute when your query contains question marks, note that the
array must be keyed numerically from zero. If it is not, run array_values() on it to force the array
to be re-keyed.

<?php
$anarray = array(42 => "foo", 101 => "bar");
$statement = $dbo->prepare("SELECT * FROM table WHERE col1 = ? AND col2 = ?");

//This will not work
$statement->execute($anarray);

//Do this to make it work
$statement->execute(array_values($anarray));
?>
```

up
down
4

Tony Casparro ¶

6 years ago

```
We know that you can't see the final raw SQL before its parsed by the DB, but if you want to simulate
the final result, this may help.

<?php
public function showQuery($query, $params)
    {
        $keys = array();
        $values = array();

        # build a regular expression for each parameter
        foreach ($params as $key=>$value)
        {
            if (is_string($key))
            {
                $keys[] = '/:'.$key.'/';
            }
            else
            {
                $keys[] = '/[?]/';
            }

            if(is_numeric($value))
            {
                $values[] = intval($value);
            }
            else
            {
                $values[] = '"'.$value .'"';
            }
        }

        $query = preg_replace($keys, $values, $query, 1, $count);
        return $query;
    }
?>
```

up
down
3

anon at anon dot com ¶

4 years ago

```
If your MySQL table has 500,000+ rows and your script is failing because you have hit PHP's memory
limit, set the following attribute.

<?php $this->pdo->setAttribute(PDO::MYSQL_ATTR_USE_BUFFERED_QUERY, false); ?>

This should make the error go away again and return memory usage back to normal.
```

up
down
2

nils andre with my googelian maily accou ¶

4 years ago

```
I realized that I ran into serious trouble when debugging my PHP scripts from the command line, and
despite of going to fetchAll and so, I always got the error

SQLSTATE[HY000]: General error: 2014 Cannot execute queries while other unbuffered queries are
active.

I realized that I had a double init command:

PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8; SET CHARACTER SET utf8;"

The first one is the better choice and removing the latter, the error is gone.
```

up
down
2
T-Rex ¶
4 years ago

```
When you try to make a query with a date, then take the whole date and not just a number.

This Query will work fine, if you try it like this:
SELECT * FROM table WHERE date = 0

But if you try it with prepared you have to take the whole date format.
<?php
$sth = $dbh->prepare('SELECT * FROM table WHERE date = :date');
$sth->execute( $arArray );

//--- Wrong:
$arArray = array(":date",0);

//--- Right:
$arArray = array(":date","0000-00-00 00:00:00");
?>

There must be something with the mysql driver.

best regards
T-Rex
```

up
down
3
russel at sunraystudios dot com ¶
10 years ago

```
I've used it and it returns booleans=>
$passed = $stmt->execute();
if($passed){
echo "passed";
} else {
echo "failed";
}

If the statement failed it would print failed.  You would want to use errorInfo() to get more info,
but it does seem to work for me.
```

up
down

1
richard at securebucket dot com ¶
5 years ago
```
Note:  Parameters don't work with a dash in the name like ":asd-asd" you can do a quick
str_replace("-","_",$parameter) to fix the issue.
```
up
down
1
Robin Millette ¶
6 years ago
```
If you're going to derive PDOStatement to extend the execute() method, you must define the signature
with a default NULL argument, not an empty array.

In otherwords:
<?php
class MyPDOStatement extends PDOStatement {
  // ...

  // don't use this form!
  // function execute($input_parameters = array()) {
  // use this instead:
  function execute($input_parameters = null) {
      // ...
      return parent::execute($input_parameters);
  }
}

?>

As a sidenote, that's why I always set default parameter to NULL and take care of handling the actual
correct default parameters in the body of the method or function. Thus, when you have to call the
function with all the parameters, you know to always pass NULL for defaults.
```
up
down
0
takingsides at gmail dot com ¶
2 years ago
```
Debugging prepared statements can be a pain sometimes when you need to copy a query and run it in the
DB directly.  The function below is an example of how to compile your own query (of course it would
need some tweaking and may not work in all scenarios).

<?php

$sql = "
    SELECT t1.*
    FROM table1 AS t1
    INNER JOIN table2 AS t2 ON (
        t2.code = t1.code
        AND t1.field1 = ?
        AND t1.field2 = ?
        AND t1.field3 = ?
    )
";
```

```
$stmt = $pdo->prepare($sql);
$params = [ 'A', 'B', 'C' ];
$stmt->execute($params);

// Output the compiled query
debug($sql, $params);

function debug($statement, array $params = [])
{
    $statement = preg_replace_callback(
        '/[?]/',
        function ($k) use ($params) {
            static $i = 0;
            return sprintf("'%s'", $params[$i++]);
        },
        $statement
    );

    echo '<pre>Query Debug:<br>', $statement, '</pre>';
}
?>

This would output something like:

SELECT t1.*
FROM table1 AS t1
INNER JOIN table2 AS t2 ON (
    t2.part_code = t1.code
    AND t1.field1 = 'A'
    AND t1.field2 = 'B'
    AND t1.field3 = 'C'
)
```

up
down
-2
mail at tinodidriksen dot com ¶
5 months ago

```
The example shows this to generate the needed number of question marks, which is amazingly wasteful:
$place_holders = implode(',', array_fill(0, count($params), '?'));

Instead, just do:
$place_holders = '?'.str_repeat(',?', count($params)-1);
```

up
down
0
Ant P. ¶
8 years ago

```
As of 5.2.6 you still can't use this function's $input_parameters to pass a boolean to PostgreSQL. To
do that, you'll have to call bindParam() with explicit types for each parameter in the query.
```

up
down
-1
dbrucas ¶
10 years ago

```
If you don't want to turn on exception raising, then try this:

    //$dbErr = $dbHandler->errorInfo(); OR
    $dbErr = $dbStatement->errorInfo();
    if ( $dbErr[0] != '00000' ) {
        print_r($dbHandler->errorInfo());
        die( "<div class='redbg xlarge'>FAILED:  $msg</div><br />".$foot);
    // or handle the error your way...
            }
    echo "SUCCESS:  $msg<br />";
... continue if succesful
```

up
down
-5
Daniel ¶
9 years ago

```
You could also use switch the order of t1 and t2 to get user_id from t1 (tested on postgresql):

SELECT
   t2.*,
   t1.user_id, t1.user_name
FROM table1 t1
LEFT JOIN table2 t2 ON t2.user_id = t1.user_id
WHERE t1.user_id = 2
```

up
down
-2
pere dot pasqual at gmail dot com ¶
1 year ago

```
It's been 7 years since simon dot lehmann at gmx dot comment, but today I found myself having
problems with a prepared statement involving an INSERT, PDO odbc driver for Microsoft Access and PHP
5.4.7. The prepared statement was done using the prepare + execute method, throwing an ugly
"SQLExecDirect[-3500] at ext\\pdo_odbc\\odbc_driver.c:247" error
and a
42000 ("Syntax error or access violation") SQLSTATE.

He suspects what the problem is and points to a possible solution: using bindParam() or bindValue()
and specify a data type.

Well, that seems to be right identifying the source of the problem, but there is a simpler solution
that worked for me, simpler and that allows you to continue using pdo::prepare() with ? as parameters
and pdo::execute():
the only thing you have to do is, if not done before, a cast of the binded parameters to its specific
type (the type that the database is expecting) before putting them in the array you pass to
pdo::execute($array).

The following code fails, throwing the error above:

<?php
$name = "John";
$length = "1";
$price = "1.78";
$SQL = "INSERT INTO table (name, length, price) VALUES (?,?,?)";
$arra = array($name, $length, $price);
```

```
$sth = $msq->prepare($SQL);
$sth->execute($arra);
?>

This one works for me like a charm:

<?php
$name = "John";
$length = (int)"1"; // the database is expecting this type
$price = (float)"1.78"; // the database is expecting this type
$SQL = "INSERT INTO table (name, length, price) VALUES (?,?,?)";
$arra = array($name, $length, $price);
$sth = $msq->prepare($SQL);
$sth->execute($arra);
?>
```

up
down
-10
narcis at narcisradu dot com ¶
10 years ago

```
For a query like this:

SELECT
   t1.user_id, t1.user_name,
   t2.*
FROM table1 t1
LEFT JOIN table2 t2 ON t2.user_id = t1.user_id
WHERE t1.user_id = 2

If I don't have an entry in table2 for user_id=2, the user_id in  result will be empty.

SELECT
   t1.user_id, t1.user_name,
   t2.user_pet, t2.user_color, t2.user_sign
FROM table1 t1
LEFT JOIN table2 t2 ON t2.user_id = t1.user_id
WHERE t1.user_id = 2

This query will return nonempty user_id.

So please be careful with wildcard select.
```

⊞ add a note

- PDOStatement
  - bindColumn
  - bindParam
  - bindValue
  - closeCursor
  - columnCount
  - debugDumpParams
  - errorCode
  - errorInfo
  - execute
  - fetch