PHP PDO Class Wrapper

[A Wrapper Class of PDO]

Version 1.2 (Beta)

Introduction:

PDO Class Wrapper is a wrapper class of PDO (PHP Data Object) library. As we know that in any web application, database makes an important role for developer to create a good dynamic web application. We can use different database drivers to make web more and more interactive and dynamic. But in any web project we also know that 'Security' is a big part and concern for developers. Every developer wants to keep user's data very safe. Hence, we use much built-in functionality in PHP to prevent unauthorized access for database e.g. mysql_real_escape_string (), addslashes () etc. But some time it's very difficult to manage big application with big chunk of code. So PHP improves MySQL to MySQLi (MySQL Improved). According to php.net The MySQLi extension has a number of benefits, the key enhancements over the MySQL extension being:Object-oriented interface

- Support for Prepared Statements
- Support for Multiple Statements
- Support for Transactions
- Enhanced debugging capabilities
- Embedded server support

php.net

What is the PDO MYSQL driver?

The PDO MYSQL driver is not an API as such, at least from the PHP programmer's perspective. In fact the PDO MYSQL driver sits in the layer below PDO itself and provides MySQL-specific functionality. The programmer still calls the PDO API, but PDO uses the PDO MYSQL driver to carry out communication with the MySQL server.

The PDO MYSQL driver is one of several available PDO drivers. Other PDO drivers available include those for the Firebird and PostgreSQL database servers.

The PDO MYSQL driver is implemented using the PHP extension framework. Its source code is located in the directory ext/pdo_mysql. It does not expose an API to the PHP programmer.

php.net

Comparison of MySQL API options for PHP			
	PHP's MySQLi Extension	**PDO (Using PDO MySQL Driver and MySQL Native Driver)**	**PHP's MySQL Extension**
PHP version introduced	5.0	5.0	Prior to 3.0
Included with PHP 5.x	yes	yes	Yes
MySQL development status	Active development	Active development as of PHP 5.3	Maintenance only
Recommended by MySQL for new projects	Yes - preferred option	Yes	No
API supports Charsets	Yes	Yes	No
API supports server-side Prepared Statements	Yes	Yes	No
API supports client-side Prepared Statements	No	Yes	No
API supports Stored Procedures	Yes	Yes	No
API supports Multiple Statements	Yes	Most	No
Supports all MySQL 4.1+ functionality	Yes	Most	No

About PDO Class Wrapper:

PDO Class Wrapper is a wrapper class of PDO (PHP Data Object) library. It has many useful built in functions to manage your web application database code very shorter. Also you will find some helpful method to fix your bug with very ease.

Net Tut+

Advantage of using PDO:

Many PHP programmers learned how to access databases by using either the MySQL or MySQLi extensions. As of PHP 5.1, there's a better way. PHP Data Objects (PDO) provides methods for prepared statements and working with objects that will make you far more productive!

PDO Class Wrapper Features:

PDO Class Wrapper has very classic methods like any database class library:

- MySQL query — pdoQuery()
- MySQL select query — select ()
- MySQL insert query — insert ()
- MySQL insert batch — insertBatch()
- MySQL update query — update()
- MySQL delete query — delete()
- MySQL truncate table — truncate()
- MySQL drop table — drop()
- MySQL describe table — describe()
- MySQL count records — count()
- Show/debug executed query — showQuery()
- Get last insert id — getLastInsertId()
- Get all last insert id — getAllLastInsertId()
- Get MySQL results — results()
- Get MySQL result — result()
- Get status of executed query — affectedRows()
- MySQL begin transactions — start()
- MySQL commit the transaction — end()
- MySQL rollback the transaction — back()
- Debugger PDO Error — setErrorLog()

How to Connect PDO Class:

Example: [A]

```
$dbConfig = array
(
 "host"=>"localhost", "dbname"=>mydb, "username"=>'root', "password"=>''
);
$db = new PdoWrapper($dbConfig);
```

Example: [B]

```
$dbConfig = array
(
 "host"=>"localhost", "dbname"=>mydb, "username"=>'root', "password"=>''
);
$db = PdoWrapper::getPDO($dbConfig);
```

PDO Class Wrapper Methods Explanations:

pdoQuery():

Method name and parameter

pdoQuery (string $sSql, array $aBindWhereParam)

Explanations:

This method is use for simple MySQL query; you can execute your MySQL query with parameterized parameter or as simple query.

Example:

```
$sql = 'select * from customers limit 5;';
$data = $pdo->pdoQuery($sql)->results();
```

Raw Query:

```
SELECT * FROM customers LIMIT 5;
```

```
$sql = "select * from customers where (customernumber = '0000' OR customern
umber = '45121') ;";
$data = $pdo->pdoQuery($sql)->results();
```

Raw Query:

```
SELECT * FROM customers WHERE (customernumber = 103 OR customernumber = 119) ;
```

```
$sql = "select * from customers where (customernumber = '0000' OR customern
umber = '45121') ;";
$data = $db->pdoQuery($sql)->results();
```

Raw Query:

```
SELECT * FROM customers WHERE (customernumber = '0000' OR customernumber = '45121') ;
```

```
$sql = "select p.checknumber, p.amount, p.paymentdate, c.customernumber, c.
customerName, c.contactLastName, c.contactFirstName, c.phone, c.addressLine
1, c.addressLine2, c.city, c.state, c.postalCode, c.country from payments a
s p inner join customers as c on p.customernumber = c.customernumber order
by p.amount desc limit 2;";
$data = $pdo->pdoQuery($sql)->results();
```

Raw Query:

```
SELECT p.checknumber, p.amount, p.paymentdate, c.customernumber, c.customername, c.contactlastname,
c.contactfirstname, c.phone, c.addressline1, c.addressline2, c.city, c.state, c.postalcode,
c.country FROM payments AS p INNER JOIN customers AS c ON p.customernumber =
c.customernumber ORDER BY p.amount DESC LIMIT 2;
```

select():

Method name and parameter

select (string $sTable , array $aColumn, array $aWhere, string $sOther)

Explanations:

The select method is made for get table data from just pass table name in method, if you omit column then you will get all fields of requested table else you can pass table field by an array. If you want to pass a where clause then you can use third parameter of select method and by pass fourth parameter you can send other filters.

Example:

Get all table fields from table without passing 2nd parameter.

```
$select = $pdo->select('customers');
$data = $select->results();
```

Raw Query:

```
SELECT * FROM `customers` ;
```

Or

You can use one line code to get a result array

```
$data = $pdo->select('employees')->results();
```

Raw Query:

```
SELECT * FROM `employees` ;
```

Get only selected fields from table

```
$data = $db-
>select('employees', array('employeeNumber','lastName','firstName'))-
>results();
```

Raw Query:

```
SELECT employeenumber, lastname, firstname FROM `employees` ;
```

Or

```
$fieldsArray = array('employeeNumber','lastName','firstName');
$data = $db->select('employees', $fieldsArray)->results();
```

Raw Query:

```
SELECT employeenumber, lastname, firstname FROM `employees` ;
```

```
$selectFields = array('employeeNumber','lastName','firstName');
$whereConditions = array('lastname'=>'bow');
$data = $db-
>select('employees', $selectFields, $whereConditions, 'ORDER BY employeeNum
ber DESC LIMIT 5')->results();
```

Raw Query:

```
SELECT employeenumber, lastname, firstname FROM `employees` WHERE lastname =
"bow" ORDER BY employeenumber DESC LIMIT 5;
```

Custom Where Clause with Select Method:

You can set your own custom where clause

```
$whereConditions = array('lastname ='=>'bow', 'or jobtitle ='=> 'Sales Rep'
, 'and isactive ='=>1, 'and officecode ='=> 1 );
$data = $db->select('employees','',$whereConditions)->results();
```

Raw Query:

```
SELECT * FROM `employees` WHERE lastname = "bow" OR jobtitle = "sales rep" AND isactive =
1 AND officecode = 1 ;
```

OR

```
$whereConditions = array('lastname ='=>'bow', 'or jobtitle ='=> 'Sales Rep'
, 'and isactive ='=>1, 'and officecode ='=> 1 );
$data = $db-
>select('employees',array('employeenumber','lastname','jobtitle'),$whereCon
ditions)->results();
```

Raw Query:

```
SELECT employeenumber, lastname, jobtitle FROM `employees` WHERE lastname = "bow" OR jobtitle = "sales
rep" AND isactive = 1 AND officecode = 1 ;
```

insert():

Method name and parameter
insert(string $sTable, array $aData)

Explanations:
By insert method you can insert record into selected table. Just pass data as an array with fields as array key and the array data will insert in to table. Insert method automatically convert your array data in to SQL injection safe data.

Example:
```
$dataArray = array('first_name'=>'Sid','last_name'=>'Mike','age'=>45);
$data = $db->insert('test',$dataArray)->getLastInsertId();
```

Raw Query:
```
INSERT INTO `test` (first_name,last_name,age) VALUES ("sid","mike",45);
```

insertBatch():

Method name and parameter
insertBatch(string $sTable, array $aData, boolean $safeModeInsert)

Explanations:
You can use this method for inserting multiple array data in same table. You have to just send full array data and rest of thing insertBatch will handle. You can send third parameter as false if you don't want to insert parameterize insert or send true if want to secure insertions.

insertBatch works with MySQL transactions so you don't need to worry about failure data. It will be rollback if anything goes wrong.

Example:

```
$dataArray[] = array('first_name'=>'Sid','last_name'=>'Mike','age'=>45);
$dataArray[] = array('first_name'=>'Scott','last_name'=>'Dimon','age'=>78);
$dataArray[] = array('first_name'=>'Meena','last_name'=>'Verma','age'=>23);

$data = $db->insertBatch('test',$dataArray, true)->getAllLastInsertId();
```

Raw Query:

```
INSERT INTO `test` (first_name, last_name, age) VALUES ("sid", "mike", 45);
INSERT INTO `test` (first_name, last_name, age) VALUES ("scott", "dimon", 78);
INSERT INTO `test` (first_name, last_name, age) VALUES ("meena", "verma", 23);
```

update():

Method name and parameter

update(string $sTable, array $aData, array $aWhere, string $sOther)

Explanations:

Update method is use for update a table with array data. You can send array data as update data in table.

Example:

```
$dataArray = array('first_name'=>'Sangeeta','last_name'=>'Mishra','age'=>35
);
$aWhere = array('id'=>23);
$data = $db->update('test', $dataArray, $aWhere->affectedRows();
```

Raw Query:

```
UPDATE `test` SET first_name = "sangeeta", last_name = "mishra", age = 35 WHERE id = 23 ;
```

Or

```
$dataArray = array('first_name'=>'Sonia','last_name'=>'Shukla','age'=>23);
$aWhere = array('age'=>35, 'last_name'=>'Mishra');
$data = $db->update('test', $dataArray, $aWhere)->affectedRows();
```

Raw Query:

```
UPDATE `test` SET first_name = "sonia", last_name = "shukla", age = 23 WHERE age = 35 AND last_name =
"mishra" ;
```

delete():

Method name and parameter

delete(string $sTable, array $aWhere, string $sOther)

Explanations:

You can delete records from table by send table name and your where clause array.

Example:

```
$aWhere = array('age'=>35);
$data = $db->delete('test', $aWhere)->affectedRows();
```

Raw Query:

```
DELETE FROM `test` WHERE age = 35 ;
```

```
$aWhere = array('age'=>45, 'first_name'=> 'Sonu');
$data = $db->delete('test', $aWhere)->affectedRows();
```

Raw Query:

```
DELETE FROM `test` WHERE age = 45 AND first_name = "sonu" ;
```

truncate():

Method name and parameter

truncate(string $sTable)

Explanations:

You can truncate table by just pass table name.

Example:

```
$data = $db->truncate('test');
```

Raw Query:

```
TRUNCATE TABLE `test`;
```

drop():

Method name and parameter
drop(string $sTable)

Explanations:
You can drop table by just pass table name.

Example:
```
$data = $db->drop('test');
```

Raw Query:
```
DROP TABLE `test`;
```

describe():

Method name and parameter
describe(string $sTable)

Explanations:
You can get a table field name and data type.

Example:
```
$data = $db->describe('test');
```

Raw Query:
```
DESC `test`;
```

count():

Method name and parameter
count(string $sTable)

Explanations:
This function will return the number of total rows in a table.

Example:
```
$data = $db->count('test');
```

Raw Query:
```
SELECT COUNT(*) AS numrows FROM `test`;
```

showQuery():

Method name and parameter
count(Boolean $logfile)

Explanations:
By this function you can get executed query. It will show raw query on your screen. If you want to logfile to save query then you can send 2nd param as true. By default it's false.

Example:
```
$db->showQuery();
```

Raw Query Example:
```
SELECT COUNT(*) AS numrows FROM `test`;
```

getLastInsertId():

Method name and parameter
getLastInsertId ()

Explanations:
Get a newly inserted id by insert function.

Example:
```
$lid = $db->getLastInsertId();
```

Return:
Number/Integer

getAllLastInsertId():

Method name and parameter
getAllLastInsertId ()

Explanations:
Get all newly inserted id by insertBatch function.

Example:
```
$lid = $db->getAllLastInsertId();
```

Return:
Array

results():

Method name and parameter
results (string $type)

Explanations:
Get array result data by executed SELECT or Select Query. You can get result in three formats Array, XML and JSON. Just pass 1st param as 'array' or 'xml' or 'json'. By default it will return array.

Example:
```
$data = $db->results();
```

Return:
Array | XML | JSON

result():

Method name and parameter
results (integer $iRow)

Explanations:
Get result from an array data by request index or false.

Example:
```
$data = $db->result(1);
```

Return:
Array | false

affectedRows():

Method name and parameter
affectedRows ()

Explanations:
Get number of affected rows by update, delete and select etc. statement or false.

Example:
```
$data = $db->affectedRows();
```

Return:
integer | false

start():

Method name and parameter
start ()

Explanations:
Start the MySQL transaction.

Example:
```
$db->start();
```

end():

Method name and parameter
end ()

Explanations:
Commit the MySQL transaction.

Example:
```
$db-> end();
```

back():

Method name and parameter
back ()

Explanations:
Rollback the MySQL transaction.

Example:
```
$db-> back();
```

setErrorLog():

Method name and parameter
setErrorLog (boolean $mode)

Explanations:
setErrorLog, method works for show/hide PDO error. If you send true then all errors will show on screen or if you send false then all errors will store in log file in same location.
Example:
```
$db-> setErrorLog(true);
```

Example Connection Page:

```php
<?php

    // include PDO Class Wrapper
    include_once 'class/class.pdowrapper.php';

    // set connection data
    $dbConfig = array
    (
     "host"=>"localhost", "dbname"=>'sampledb', "username"=>'root', "password"=>''
    );

    // get instance of PDO Class Wrapper
    $db = PdoWrapper::getPDO($dbConfig);

    // set error log mode true to show all error on screen
    $db->setErrorLog(true);

    /* simple update example */

    // update array data
    $dataArray = array('first_name'=>'Sangeeta','last_name'=>'Mishra','age'=>35);

    // where condition array
    $aWhere = array('id'=>23);

    // call update function
    $q = $p->update('test', $dataArray, $aWhere)->showQuery()->affectedRows();

    ?>
```

Output:

```
UPDATE `test` SET first_name = "sangeeta", last_name = "mishra", age = 35 WHERE id = 23 ;

1
```

Cheers!!
Priyadarshan Salkar | priyadarshan.salkar@lbi.co.in
Bhaskar Rabha |bhaskar.rabha@lbi.co.in
Neeraj Singh | neeraj.singh@lbi.co.in

[Document End]