PHP MySQLi Class Documentation

Release 1.0

Read the Docs

Sep 16, 2017

Contents

Note: This is not an official documentation. Official API documentation is available in the GitHub repo here.

CHAPTER 1

Installation

To utilize this class, first import MysqliDb.php into your project, and require it.

```
require_once ('MysqliDb.php');
```

Installation with composer

It is also possible to install library via composer

```
composer require joshcam/mysqli-database-class:dev-master
```

CHAPTER 2

Initialization

Simple initialization with utf8 charset set by default:

```
$db = new MysqliDb ('host', 'username', 'password', 'databaseName');
```

Advanced initialization:

```
$db = new MysqliDb (Array (
        'host' => 'host',
        'username' => 'username',
        'password' => 'password',
        'db'=> 'databaseName',
        'port' => 3306,
        'prefix' => 'my_',
        'charset' => 'utf8'));
```

table prefix, port and database charset params are optional. If no charset should be set charset, set it to null

Also it is possible to reuse already connected mysqli object:

```
$mysqli = new mysqli ('host', 'username', 'password', 'databaseName'); $db = new MysqliDb ($mysqli);
```

If no table prefix were set during object creation its possible to set it later with a separate call:

```
$db->setPrefix ('my_');
```

If you need to get already created mysqliDb object from another class or function use

```
function init () {  // db staying private here
        $db = new MysqliDb ('host', 'username', 'password', 'databaseName');
```

```
function myfunc () {  // obtain db object created in init ()
    $db = MysqliDb::getInstance();
    ...
}
```

CHAPTER 3

Insert Query

Simple Example

$data = Array (“login” => “admin”, “firstName” => “John”, “lastName” => ‘Doe’

); $id = $db->insert (‘users’, $data); if($id)

echo ‘user was created. Id=’ . $id;

Insert with functions use

$data = Array (‘login’ => ‘admin’, ‘active’ => true, ‘firstName’ => ‘John’, ‘lastName’ => ‘Doe’, ‘password’ => $db->func(‘SHA1(?)’,Array (“secretpassword+salt”)), // password = SHA1(‘secretpassword+salt’) ‘createdAt’ => $db->now(), // createdAt = NOW() ‘expires’ => $db->now(‘+1Y’) // expires = NOW() + interval 1 year // Supported intervals [s]econd, [m]inute, [h]hour, [d]day, [M]onth, [Y]ear

);

$id = $db->insert (‘users’, $data); if ($id)

echo ‘user was created. Id=’ . $id;

else echo ‘insert failed: ‘ . $db->getLastError();

Insert with on duplicate key update

$data = Array (“login” => “admin”, “firstName” => “John”, “lastName” => ‘Doe’, “createdAt” => $db->now(), “updatedAt” => $db->now(),

); $updateColumns = Array (“updatedAt”); $lastInsertId = “id”; $db->onDuplicate($updateColumns, $lastInsertId); $id = $db->insert (‘users’, $data);

Replace Query

Replace() method implements same API as insert();

CHAPTER 4

Update Query

$data = Array (‘firstName’ => ‘Bobby’, ‘lastName’ => ‘Tables’, ‘editCount’ => $db->inc(2), // editCount = editCount + 2; ‘active’ => $db->not() // active = !active;

); $db->where (‘id’, 1); if ($db->update (‘users’, $data))

echo $db->count . ‘ records were updated’;

else echo ‘update failed: ‘ . $db->getLastError();

update() also support limit parameter:

$db->update (‘users’, $data, 10); // Gives: UPDATE users SET ... LIMIT 10

CHAPTER 5

Select Query

After any select/get function calls amount or returned rows is stored in $count variable

$users = $db->get('users'); //contains an Array of all users $users = $db->get('users', 10); //contains an Array 10 users

or select with custom columns set. Functions also could be used

$cols = Array ("id", "name", "email"); $users = $db->get ("users", null, $cols); if ($db->count > 0)

**foreach ($users as $user) {** print_r ($user);

}

or select just one row

$db->where ("id", 1); $user = $db->getOne ("users"); echo $user['id'];

$stats = $db->getOne ("users", "sum(id), count(*) as cnt"); echo "total ".$stats['cnt']. "users found";

or select one column value or function result

$count = $db->getValue ("users", "count(*)"); echo "{$count} users found";

select one column value or function result from multiple rows:

$logins = $db->getValue ("users", "login", null); // select login from users $logins = $db->getValue ("users", "login", 5); // select login from users limit 5 foreach ($logins as $login)

echo $login;

CHAPTER 6

Pagination

Use paginate() instead of get() to fetch paginated result

$page = 1; // set page limit to 2 results per page. 20 by default $db->pageLimit = 2; $products = $db->arraybuilder()->paginate("products", $page); echo "showing $page out of " . $db->totalPages;

Result transformation / map

Instead of getting an pure array of results its possible to get result in an associative array with a needed key. If only 2 fields to fetch will be set in get(), method will return result in array($k => $v) and array ($k => array ($v, $v)) in rest of the cases.

$user = $db->map ('login')->ObjectBuilder()->getOne ('users', 'login, id'); Array (

[user1] => 1

)

$user = $db->map ('login')->ObjectBuilder()->getOne ('users', 'id,login,createdAt'); Array (

[user1] => stdClass Object

([id] => 1 [login] => user1 [createdAt] => 2015-10-22 22:27:53

)

)

Defining a return type

MysqliDb can return result in 3 different formats: Array of Array, Array of Objects and a Json string. To select a return type use ArrayBuilder(), ObjectBuilder() and JsonBuilder() methods. Note that ArrayBuilder() is a default return type

// Array return type $= $db->getOne("users"); echo $u['login']; // Object return type $u = $db->ObjectBuilder()->getOne("users"); echo $u->login; // Json return type $json = $db->JsonBuilder()->getOne("users");

CHAPTER 7

Delete Query

$db->where(‘id’, 1); if($db->delete(‘users’)) echo ‘successfully deleted’;

CHAPTER 8

Running raw SQL queries

```
$users = $db->rawQuery('SELECT * from users where id >= ?', Array (10)); foreach ($users as $user) {
    print_r ($user);
}
```

To avoid long if checks there are couple helper functions to work with raw query select results:

Get 1 row of results:

```
$user = $db->rawQueryOne ('select * from users where id=?', Array(10)); echo $user['login']; // Object return type $user = $db->ObjectBuilder()->rawQueryOne ('select * from users where id=?', Array(10)); echo $user->login;
```

Get 1 column value as a string:

```
$password = $db->rawQueryValue ('select password from users where id=? limit 1', Array(10)); echo "Password is {$password}"; NOTE: for a rawQueryValue() to return string instead of an array 'limit 1' should be added to the end of the query.
```

Get 1 column value from multiple rows:

```
$logins = $db->rawQueryValue ('select login from users limit 10'); foreach ($logins as $login)
    echo $login;
```

More advanced examples:

```
$params = Array(1, 'admin'); $users = $db->rawQuery("SELECT id, firstName, lastName FROM users WHERE id = ? AND login = ?", $params); print_r($users); // contains Array of returned rows

// will handle any SQL query $params = Array(10, 1, 10, 11, 2, 10); $q = "(
    SELECT a FROM t1 WHERE a = ? AND B = ? ORDER BY a LIMIT ?
) UNION (
```

SELECT a FROM t2 WHERE a = ? AND B = ? ORDER BY a LIMIT ?

)”; $resutls = $db->rawQuery ($q, $params); print_r ($results); // contains Array of returned rows

CHAPTER 9

Query Keywords

To add LOW PRIORITY | DELAYED | HIGH PRIORITY | IGNORE and the rest of the mysql keywords to INSERT (), REPLACE (), GET (), UPDATE (), DELETE() method or FOR UPDATE | LOCK IN SHARE MODE into SELECT ():

```
$db->setQueryOption (‘LOW_PRIORITY’)->insert ($table, $param); // GIVES: INSERT LOW_PRIORITY INTO table ...

$db->setQueryOption (‘FOR UPDATE’)->get (‘users’); // GIVES: SELECT * FROM USERS FOR UPDATE;
```

Also you can use an array of keywords:

```
$db->setQueryOption (Array(‘LOW_PRIORITY’, ‘IGNORE’))->insert ($table,$param); // GIVES: INSERT LOW_PRIORITY IGNORE INTO table ...
```

Same way keywords could be used in SELECT queries as well:

```
$db->setQueryOption (‘SQL_NO_CACHE’); $db->get(“users”); // GIVES: SELECT SQL_NO_CACHE * FROM USERS;
```

Optionally you can use method chaining to call where multiple times without referencing your object over an over:

```
$results = $db  ->where(‘id’, 1) ->where(‘login’, ‘admin’) ->get(‘users’);
```

CHAPTER 10

Ordering method

$db->orderBy("id","asc"); $db->orderBy("login","Desc"); $db->orderBy("RAND ()"); $results = $db->get('users'); // Gives: SELECT * FROM users ORDER BY id ASC,login DESC, RAND ();

Order by values example:

$db->orderBy('userGroup', 'ASC', array('superuser', 'admin', 'users')); $db->get('users'); // Gives: SELECT * FROM users ORDER BY FIELD (userGroup, 'superuser', 'admin', 'users') ASC;

If you are using setPrefix () functionality and need to use table names in orderBy() method make sure that table names are escaped with ``.

$db->setPrefix ("t_"); $db->orderBy ("users.id","asc"); $results = $db->get ('users'); // WRONG: That will give: SELECT * FROM t_users ORDER BY users.id ASC;

$db->setPrefix ("t_"); $db->orderBy ("*users*.id", "asc"); $results = $db->get ('users'); // CORRECT: That will give: SELECT * FROM t_users ORDER BY t_users.id ASC;

CHAPTER 11

Grouping method

$db->groupBy (“name”); $results = $db->get (‘users’); // Gives: SELECT * FROM users GROUP BY name;

Join table products with table users with LEFT JOIN by tenantID

CHAPTER 12

Properties sharing

Its is also possible to copy properties

```
$db->where (“agentId”, 10); $db->where (“active”, true);

$customers = $db->copy (); $res = $customers->get (“customers”, Array (10, 10)); // SELECT * FROM customers where agentId = 10 and active = 1 limit 10, 10

$cnt = $db->getValue (“customers”, “count(id)”); echo “total records found: ” . $cnt; // SELECT count(id) FROM users where agentId = 10 and active = 1
```

CHAPTER 13

JOIN method

$db->join(“users u”, “p.tenantID=u.tenantID”, “LEFT”); $db->where(“u.id”, 6); $products = $db->get (“products p”, null, “u.name, p.productName”); print_r ($products);

CHAPTER 14

Subqueries

Subquery init

Subquery init without an alias to use in inserts/updates/where Eg. (select * from users)

```
$sq = $db->subQuery(); $sq->get (“users”);
```

A subquery with an alias specified to use in JOINs . Eg. (select * from users) sq

```
$sq = $db->subQuery(“sq”); $sq->get (“users”);
```

Subquery in selects:

```
$ids = $db->subQuery (); $ids->where (“qty”, 2, “>”); $ids->get (“products”, null, “userId”);

$db->where (“id”, $ids, ‘in’); $res = $db->get (“users”); // Gives SELECT * FROM users WHERE id IN (SELECT userId FROM products WHERE qty > 2)
```

Subquery in inserts:

```
$userIdQ = $db->subQuery (); $userIdQ->where (“id”, 6); $userIdQ->getOne (“users”, “name”),

$data = Array (  “productName” => “test product”, “userId” => $userIdQ, “lastUpdated” => $db->now()

); $id = $db->insert (“products”, $data); // Gives INSERT INTO PRODUCTS (productName, userId, lastUpdated) values (“test product”, (SELECT name FROM users WHERE id = 6), NOW());
```

Subquery in joins:

```
$usersQ = $db->subQuery (“u”); $usersQ->where (“active”, 1); $usersQ->get (“users”);

$db->join($usersQ, “p.userId=u.id”, “LEFT”); $products = $db->get (“products p”, null, “u.login, p.productName”); print_r ($products); // SELECT u.login, p.productName FROM products p LEFT JOIN (SELECT * FROM t_users WHERE active = 1) u on p.userId=u.id;
```

CHAPTER 15

EXISTS / NOT EXISTS condition

$sub = $db->subQuery(); $sub->where(“company”, ‘testCompany’); $sub->get (“users”, null, ‘userId’);

$db->where (null, $sub, ‘exists’); $products = $db->get (“products”); // Gives SELECT * FROM products WHERE EXISTS (select userId from users where company=’testCompany’)

CHAPTER 16

Has method

A convenient function that returns TRUE if exists at least an element that satisfy the where condition specified calling the “where” method before this one.

```
$db->where(“user”, $user); $db->where(“password”, md5($password)); if($db->has(“users”)) {
    return “You are logged”;
} else {  return “Wrong user/password”;
}
```

CHAPTER 17

Helper methods

Reconnect in case mysql connection died:

```
if (!$db->ping())  $db->connect()
```

Get last executed SQL query: Please note that function returns SQL query only for debugging purposes as its execution most likely will fail due missing quotes around char variables.

```
$db->get('users'); echo "Last executed query was ". $db->getLastQuery();
```

Check if table exists:

```
if ($db->tableExists ('users'))  echo "hooray";
```

mysqli_real_escape_string() wrapper:

```
$escaped = $db->escape ("' and 1=1");
```

CHAPTER 18

Transaction helpers

Please keep in mind that transactions are working on innoDB tables. Rollback transaction if insert fails:

```
$db->startTransaction(); ... if (!$db->insert ('myTable', $insertData)) {
    //Error while saving, cancel new record $db->rollback();
} else {  //OK $db->commit();
}
```

CHAPTER 19

Query exectution time benchmarking

To track query execution time setTrace() function should be called.

```
$db->setTrace (true); // As a second parameter it is possible to define prefix of the path which should be striped from filename // $db->setTrace (true, $_SERVER['SERVER_ROOT']); $db->get("users"); $db->get("test"); print_r ($db->trace);
```

```
[0] => Array

    ( [0] => SELECT * FROM t_users ORDER BY id ASC [1] => 0.0010669231414795
      [2] => MysqliDb->get() >> file "/avb/work/PHP-MySQLi-Database-Class/tests.php"
      line #151

    )

[1] => Array

    ( [0] => SELECT * FROM t_test [1] => 0.00069189071655273 [2] => MysqliDb->get()
      >> file "/avb/work/PHP-MySQLi-Database-Class/tests.php" line #152

    )
```

CHAPTER 20

Indices and tables

- genindex
- modindex
- search