# Contents

1 Installation 3

2 Initialization 5
  2.1 Advanced initialization: 5

3 Insert Query 7
  3.1 Simple Example 7
  3.2 Insert with on duplicate key update 7
  3.3 Replace Query 8

4 Update Query 9

5 Select Query 11

6 Pagination 13
  6.1 Result transformation / map 13
  6.2 Defining a return type 13

7 Delete Query 15

8 Running raw SQL queries 17
  8.1 More advanced examples: 17

9 Query Keywords 19

10 Ordering method 21

11 Grouping method 23

12 Properties sharing 25

13 JOIN method 27

14 Subqueries 29

15 EXISTS / NOT EXISTS condition 31

16 Has method 33
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>Helper methods</td>
<td>35</td>
</tr>
<tr>
<td>18</td>
<td>Transaction helpers</td>
<td>37</td>
</tr>
<tr>
<td>19</td>
<td>Query execution time benchmarking</td>
<td>39</td>
</tr>
<tr>
<td>20</td>
<td>Indices and tables</td>
<td>41</td>
</tr>
</tbody>
</table>
Note: This is not an official documentation. Official API documentation is available in the GitHub repo here.
Installation

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

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

Installation with composer

It is also possible to install library via composer

```bash
composer require joshcam/mysqli-database-class:dev-master
```
Chapter 1. Installation
Initialize Simple initialization with utf8 charset set by default:

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

**Advanced initialization:**

```php
$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:

```php
$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:

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

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

```php
function init () { // db staying private here
    $db = new MysqliDb ('host', 'username', 'password', 'databaseName');
```
function myfunc() {  // obtain db object created in init()
    $db = MysqliDb::getInstance();
    ...
}
Simple Example

```php
$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]our,
[d]ay, [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

```php
$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();
$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

```php
$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

```php
(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

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

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

or select one column value or function result

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

select one column value or function result from multiple rows:

```php
$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;
```
Use paginate() instead of get() to fetch paginated result

```php
$page = 1; // set page limit to 2 results per page. 20 by default $db->paginateLimit = 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.

```php
$user = $db->map ("login")->ObjectBuilder()->getOne ("users", "login, id"); Array ( [user1] => 1 )
```

```php
```

**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 $u = $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';
Running raw SQL queries

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

More advanced examples:

```php
$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); $sql = "(
    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 ?

$results = $db->rawQuery ($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 and 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 11. Grouping method
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
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);
Subqueries

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

```php
$sq = $db->subQuery(); $sq->get ("users");
```

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

```php
$sq = $db->subQuery("sq"); $sq->get("users");
```

Subquery in selects:

```php
$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:

```php
$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:

```php
$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;
```
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')
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.

```php
$db->where("user", $user); $db->where("password", md5($password)); if($db->has("users")) {
    return "You are logged";
} else {
    return "Wrong user/password";
}
```
Reconnect in case mysql connection died:

```php
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.

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

Check if table exists:

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

`mysqli_real_escape_string()` wrapper:

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

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

```php
$db->startTransaction(); ... if (!$db->insert ('myTable', $insertData)) {
    //Error while saving, cancel new record $db->rollback();
} else { //OK $db->commit();
}
```
To track query execution time `setTrace()` function should be called.

```php
$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**

CHAPTER 20

Indices and tables

• genindex
• modindex
• search