



# Winter Tooling

## ALSO INSIDE

### Leveling Up:

The Power of Interfaces

### Security Corner:

The Importance of Effective Validation

### Education Station:

Plates—the Templating Engine Designed with PHP in Mind

### Community Corner:

Q&A with Rafael Dohms

Build Data Analysis and Visualization Tools with PHP

Grinding Out Profiled (and Faster) PHP Code

Switch to MySQLnd Already!

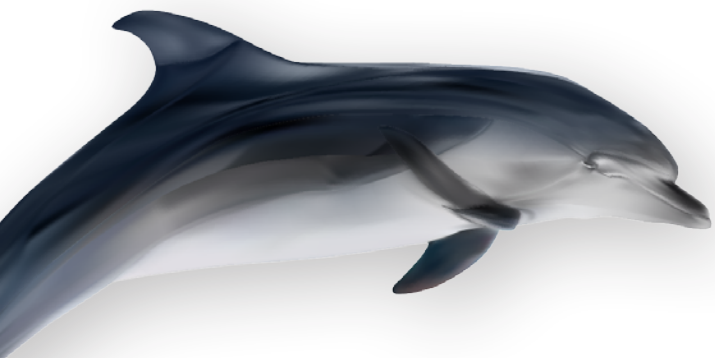
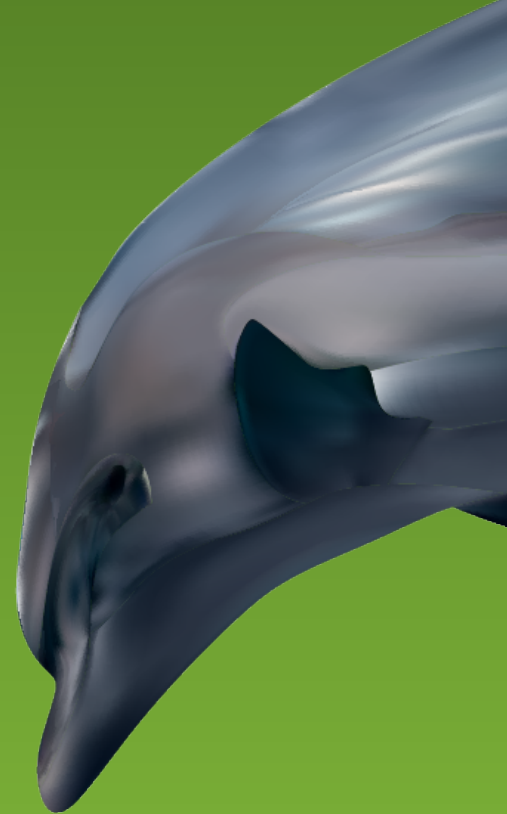
Clever, Secure, Team-Friendly Templating with FigDice

**FREE  
Article!**

# Switch to MySQLnd Already!

Jervin Real

MySQL has been the long-standing de facto backend data store for PHP applications, and I imagine it will be for a very long time. Traditionally, connecting from PHP to MySQL has been very stable and available out of the box on many platforms. New and legacy applications enjoy this stability; in many cases, this is the main reason many teams do not require bleeding-edge or at least recent stable release upgrades of their stack. However, there are a number of compelling reasons why applications would need an upgrade—in this case switching our php-mysql/php-mysqli connector from libmysql to mysqlnd.



Traditionally, we have two extensions, php-mysql and php-mysqli, which both expose C level functions as PHP functions from libmysql. If you look at the libmysql source code, the `mysql_*` functions are almost all exported into the same PHP functions on php-mysql, and this was the first and has been the default MySQL extension before PHP 5. The php-mysqli extension came with the release of PHP 5. The new extension had full support of newer features from MySQL 4.1.3 and support for OOP-style implementation.

Along the same lines, we have PDO\_MYSQL for MySQL-backed PHP Data Objects. In this case, PDO is the PHP extension that allows an



## DisplayInfo()

### Requirements:

- MySQL Native Driver—<http://php.net/book.mysqlnd>

### Related URLs:

- mysql\_ removed in PHP 7—<http://phpa.me/ref-deprecated-php7>
- Mysql 4.1 features—<http://downloads.mysql.com/docs/refman-4.1-en.a4.pdf>
- PHP 5 Usage—<http://phpa.me/php-usage-nov2015>
- mysqlnd\_ms—<http://php.net/book.mysqlnd-ms>
- Simple MySQL Master HA with mysqlnd\_ms—<https://www.percona.com/blog/?p=24407>
- High Availability with mysqlnd\_ms on Percona XtraDB Cluster—<https://www.percona.com/blog/?p=24413>
- mysqlnd\_uh: Installing a proxy—<http://phpa.me/mysqlnd-proxy-quickstart>
- mysqlnd\_memcache—<http://php.net/book.mysqlnd-memcache>
- PHP: Memory savings with mysqlnd —<http://phpa.me/oracle-mysqlnd-memory>
- PHP mysqlnd memory optimizations: from 49MB to 2MB—<http://blog.ulf-wendel.de/?p=2989>
- Differences between mysqlnd and libmysql—<http://phpa.me/mysqlnd-libmysql>

application to use a standard API to access different backend data stores without significant changes in the API. PDO\_MYSQL acts as an intermediate layer between libmysql and PDO. The difference between PDO\_MYSQL and php-mysql/mysqli is that the former does not expose any functions by itself; underneath, it has its own set of C level functions that adheres to the PDO interface.

As a last piece of history, php-mysql will be removed from PHP 7 (see Related URLs). For some this may be bad news, this is an excellent opportunity to explore mysqlnd as well, if you are in that situation. We've mentioned that php-mysqli was born out of growing needs, and because these needs have been fulfilled

for a long time now, it only makes sense to reduce code debt. There are many reason, but to name a few, php-mysql:

- Does not support MySQL 4.1.3, at least not all its features (see Related URLs).
- Code was hackish and becoming difficult to maintain for newer systems.
- Has no support for prepared statements, parameter binding, and batched statements.
- Does not support multiple character sets, which is increasingly being used.
- MySQL 4.1.3+ has better binary protocol support you should not ignore
- ... and more

## MySQL Native Driver

MySQL Native Driver (mysqlnd) is a newer connection driver for the PHP extensions php-mysql/php-mysqli and PDO. When we say connection driver, it means the low-level layer that handles the communications between the MySQL server and our PHP extensions. The foremost benefit is that there are no changes on how your application should be communicating with MySQL!

By one measurer, PHP 5.3 still accounts for a majority of the PHP-enabled sites on the planet (see Related URLs), either because of some show-stopping code they cannot upgrade or they are stuck with systems where upgrading wholly is not an immediate option. Whether or not your business is part of this statistic, consider switching to mysqlnd soon.

## Installation

If you are running PHP 5.4+ on most systems, such as RHEL and Debian-based systems, chances are you are already running with mysqlnd. You can use php-cli to check if this is the case.

The command:

```
php -i | grep 'Client API version => mysqlnd'
```

should output something like:

```
Client API version => mysqlnd 5.0.11-dev - 20120503 -  
$Id: 3c688b6bbc30d36af3ac34fdd4b7b5b787fe5555 \
```

If “mysqlnd” does not show up anywhere in the output, don’t fret, it should not be that hard to install. If you are already on 5.4+ but somehow explicitly chose not to use mysqlnd from your package manager, you can easily replace it with similar commands like below:

```
# Debian/Ubuntu  
apt-get install php5-mysqlnd  
# RHEL/CentOS  
yum install php-mysqlnd
```

Otherwise, on 5.3, you either have to compile from source or find repositories such as Remi-Collet, Atomic or Webtatic that have packages for 5.3.

Compiling from source is not that difficult either—again, because there are no additional dependencies with libmysql, at least, you will need to specify the following with your configure command.:

```
./configure --with-mysql=mysqlnd --with-mysqli=mysqlnd --with-pdo-mysql=mysqlnd
```

On Windows, starting from 5.3, mysqlnd is already bundled and used by default if you are using the official PHP packages. So, all good there.

## How Can MySQLnd Help Your Application?

### Extend with Plugins

There are a number of plugins that come with mysqlnd developed by Oracle. They can be optimized at C level and have direct access to the mysqlnd extension API. A number of plugins are readily for performance monitoring, connection multiplexing, load balancing, and even allowing user-level plugins, i.e., PHP code to be executed. These plugins require another discussion at a different time. However, this opens up other possibilities for legacy applications; here are a few examples:

You can scale your applications without requiring additional hardware. Traditional implementations can be done via the PHP code or introducing another layer like HAProxy within the same application server or a dedicated one. A good use case is implementing high availability with **mysqlnd\_ms** via asynchronous or synchronous MySQL replication, see Related URLs for more details.

With a plugin, you can introduce additional layers of abstraction transparently to support new features. Sometimes, even legacy applications will be forced to support new features or functionality, however, it will not be easy to add them by touching thousands of lines of possibly untested code. Implementing these as hooks allows businesses to isolate implementation and testing for maximum compatibility. **mysqlnd\_uh**, <http://php.net/book.mysqlnd-uh> was created to make this possible. For example, by acting as proxy when executing queries.

The **mysqlnd\_memcache** plugin can translate simple SELECT queries into queries directly to InnoDB Memcache storage. This allows for key-value store-like capabilities that may not require significant code changes. For example, intercept INSERT and SELECT queries to persistent sessions table with the help of **mysqlnd\_uh**.

There are a few more plugins initially written, however, most of them are proof of concept and testing and experimentation is left to the reader.

## Potential Performance Improvement

The connector is now closely developed as a PHP native component; this means that it is integrated well with the core code giving PHP direct control over all structures from the wire to the view. One big improvement is no more double buffering of the results between PHP extension and libmysqlclient.

Previously, for simple fetch results, libmysqlclient had to buffer results to its buffers, then php-mysql again when results are read. When using mysqlnd, there is only one in-memory copy per row kept for the result set by default, reducing memory footprint, and allowing more PHP processes per server to execute. Additionally, depending on how you actually handle your result sets, you can also try MYSQLI\_STORE\_RESULT\_COPY\_DATA which might help in some situations. (Disclaimer, I tried this benchmark in the past and got similar results, but your situation may vary.) See Related URLs for more information.

Did I mention that if you do not get immediate performance improvement, the built-in diagnostics and statistics functions can help you achieve the same?

## Improved Diagnostics

MySQLnd offers a number of client and connection statistics <http://php.net/mysqlnd.stats> for on-the-fly optimization. Some of the things you can actually do with `mysqli_get_client_stats()` and `mysqli_get_connection_stats()` include:

You can identify whether a query is slow on the MySQL server or only some of the application servers with the `no_index_used`, `bad_index_used`, and `slow_queries` counters. Here is an example with a `SELECT * FROM table` query, which did a full table scan.

stat	client	connection
<code>no_index_used</code>	1	2
<code>bad_index_used</code>	0	0
<code>slow_queries</code>	0	0

Identify areas where you can save memory usage from large result sets or freeable memory. The `mem_*` stats are internal counters for memory allocation and deallocations under different conditions and can help detect potential memory leaks with your code or the extension itself. The `copy_on_write_*` stats show how many instances where results are referenced directly from internal mysqlnd buffers and when these buffers needed to be duplicated because changes were made to the results. In this case, some parts of the code may benefit from



enabling `mysqlnd.fetch_data_copy` (see [http://php.net/mysqlnd.config#ini.mysqlnd.fetch\\_data\\_copy](http://php.net/mysqlnd.config#ini.mysqlnd.fetch_data_copy)).

<code>copy_on_write_saved</code>	0
<code>copy_on_write_performed</code>	0
<code>mem_emalloc_count</code>	884
<code>mem_emalloc_amount</code>	34736
<code>mem_ecalloc_count</code>	498
<code>mem_ecalloc_amount</code>	140912
<code>mem_erealloc_count</code>	0
<code>mem_erealloc_amount</code>	0
<code>mem_efree_count</code>	1678
<code>mem_efree_amount</code>	182062
<code>mem_malloc_count</code>	300
<code>mem_malloc_amount</code>	413902
<code>mem_calloc_count</code>	600
<code>mem_calloc_amount</code>	187200
<code>mem_realloc_count</code>	0
<code>mem_realloc_amount</code>	0
<code>mem_free_count</code>	490
<code>mem_free_amount</code>	7154
<code>mem_strndup_count</code>	98
<code>mem_strndup_count</code>	104
<code>mem_estndup_count</code>	198
<code>mem_strdup_count</code>	208
<code>proto_text_fetched_blob</code>	512
<code>proto_binary_fetched_blob</code>	0
<code>bytes_received_real_data_normal</code>	536870912
<code>bytes_received_real_data_ps</code>	0




Pinpoint the source of connection errors. Previously, doing this server side required detective work. Knowing when connect or reconnect failure happens, or when unexpected connection termination occurs, are essential. When these statistics are non-zero, you can help isolate and easily report from the code when they occur. This is particularly helpful when you have multiple distinct applications connecting to the same MySQL server.

<code>connect_failure</code>	0	0
<code>implicit_close</code>	0	0
<code>disconnect_close</code>	0	0
<code>in_middle_of_command_close</code>	0	0
<code>init_command_failed_count</code>	0	0

You can measure how effective your persistent connections are—a very high number of persistent connections per application server might cause the MySQL server to use an unnecessary amount of memory for its per thread buffers or `max_connections`, in which case you are likely better off optimizing from the server level and disabling persistent connection. On the other hand, if your statistics show a handful of connections with high rate of reuse, then things are going right.

<code>connect_success</code>	2
<code>connect_failure</code>	0
<code>connection_reused</code>	98
<code>reconnect</code>	0
<code>pconnect_success</code>	2
<code>active_connections</code>	2
<code>active_persistent_connections</code>	2
<code>explicit_close</code>	0
<code>implicit_close</code>	0
<code>disconnect_close</code>	0
<code>com_ping</code>	0
<code>com_change_user</code>	98



With a little review, you can predict container/server scalability. With the combined metrics on memory, queries, and network, you can gauge whether a server may be nearing capacity both for the MySQL server and application server. While you may have to create your own instrumentation, there are tools like NewRelic's custom metrics with its PHP agent that can help you get started with this.

Consider a legacy application that has been held back numerous times for upgrade or enhancements. Allowing these applications to collect performance and diagnostic metrics over time can help you decide based on numbers and not solely based on executive priorities. If not as a whole, focus on critical parts incrementally until you are caught up with the whole stack.

### Reducing Clutter a Bit More

Because mysqlnd does not depend on libmysql anymore, we take away the lingering dependencies with the latter. It would've been easier if the PHP source code could bundle libmysql in the same package, but alas, this was not the case because of different licensing models.

Now, this means that if you do not need the MySQL client libraries installed, for example, in a container or

a virtual machine, you are reducing your application footprint. By reducing dependency, this minimizes friction in packaging your application development environment to developers and testers.

### Watch Out

While we've highlighted a number of good things that come with the package, there are a few things to watch out for.

Be careful with `memory_limit`. Because mysqlnd now follows this setting, the usual reaction is to increase it when you are dealing with larger result sets. However, this may be counter-intuitive at times, as there may be other parts of the application that would otherwise consume the same memory you honestly intended for MySQL results. Of course, there can also be bugs <https://bugs.php.net/bug.php?id=68544> in this space as well.

The `old_password` MySQL configuration is not supported—if you have MySQL accounts in the `mysql.user` table that are still using the old password hashing format, this can break your application, since those credentials won't work when your application tries to connect to the database server. I mean, come on, how hard is it to create new MySQL accounts and update your application's configuration files?



*We are passionate about making  
the web a better place.*

---

Come work with us!  
[automattic.com/work-with-us](https://automattic.com/work-with-us)

---

If you rely on a local `my.cnf` for your connections, unfortunately, they do not work yet. An alternative to this is setting your credentials via global or local `php.ini` (or `.user.ini`, `.htaccess`, `-c` option to CLI) by specifying `mysqli.default_` configuration options <http://php.net/mysqli.configuration>.

A few other incompatibilities are officially documented at <http://php.net/mysqlnd.incompatibilities>. See Related URLs for the full comparison of `mysqlnd` and `libmysql`.

Measure as much as you can, as I've seen cases where large queries can slow down. For example, in this small test, <https://gist.github.com/dotmanila/ede81ea108f5dc3990ac>, I get obviously less-efficient times when not using protocol compression with `mysqlnd`.

## LISTING 1

```

01. <?php
02. //mysql> show create table t \G
03. //***** 1. row *****
04. //      Table: t
05. //Create Table: CREATE TABLE `t` (
06. //  `t` longtext
07. //) ENGINE=InnoDB DEFAULT CHARSET=latin1
08. //1 row in set (0.00 sec)
09.
10. //mysql> insert into t (t) values (repeat('a',1048576));
11. //Query OK, 1 row affected (0.14 sec)
12.
13. //https://github.com/jacobbednarz/php-bench/blob/master/benchmark.php
14. require_once 'benchmark.php';
15.
16. function t() {
17.     $mysqli = mysqli_init();
18.     if (!$mysqli) {
19.         echo 'mysqli_init failed';
20.         exit;
21.     }
22.
23.     if (!$mysqli->real_connect('localhost', 'myUser',
24.                             'secret', 'bench', 3306, NULL,
25.                             MYSQLI_CLIENT_COMPRESS)) {
26.         trigger_error(
27.             'Connect Error (' . mysqli_connect_errno() . ') '
28.             . mysqli_connect_error(),
29.             E_USER_ERROR
30.         );
31.     }
32.
33.     $r = $mysqli->query('select * from t');
34.     while($o = $r->fetch_object()) {}
35.     $r->free();
36.     $mysqli->close();
37. }
38.
39. $b = new Benchmark();
40. $b->iterations = 1000;
41. $b->report('t', 't');
42. $b->bench();

```



**Without compression mysqlnd:**

```
[root@php-mysqlnd ~]# php php-compress.php
Using 1000 iterations
```

IDENTIFIER	EXECUTION TIME
t	217.83938813ms

**Without compression libmysql:**

```
[root@php-libmysql ~]# php php-compress.php
Using 1000 iterations
```

IDENTIFIER	EXECUTION TIME
t	28.64038396ms

**With compression mysqlnd:**

```
[root@php-mysqlnd ~]# php php-compress.php
Using 1000 iterations
```

IDENTIFIER	EXECUTION TIME
t	22.08642602ms

**With compression libmysql:**

```
[root@php-libmysql ~]# php php-compress.php
Using 1000 iterations
```

IDENTIFIER	EXECUTION TIME
t	23.59013605ms

## Conclusion

It's very obvious what potential mysqlnd brings to the table and how easy it is to switch to. Since the majority of the changes are transparent to the extension API, extending your application capabilities with MySQL storage, diagnosing and improving performance from your application, and bringing limited scaling capabilities become immediately possible. Learn more about mysqlnd from the manual, <http://php.net/book.mysqlnd>, and happy hacking!



As Senior Consultant, **JERVIN** partners with Percona's customers on building reliable and highly performant MySQL infrastructures while also doing other fun stuff like watching cat videos on the internet. Jervin joined Percona in April 2010.

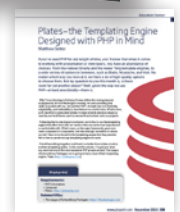
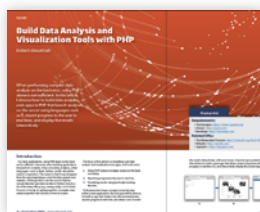
Starting as a PHP programmer, Jervin quickly got involved with the LAMP stack. He has worked on several high-traffic sites and a number of specialized web applications; i.e., mobile content distribution. Before joining Percona, Jervin also worked with several hosting companies, providing care for customer hosted services and data on both Linux and Windows.

**Twitter:** @dotmanila

# Want more articles like this one?

Keep your skills current and stay on top of the latest PHP news and best practices by reading each new issue of php[architect], jam-packed with articles.

Learn more every month about frameworks, security, ecommerce, databases, scalability, migration, API integration, devops, cloud services, business development, content management systems, and the PHP community.



magazine

books

conferences

training

[www.phparch.com](http://www.phparch.com)

**Get the complete issue  
for only \$6!**

We also offer digital and print+digital subscriptions starting at \$49/year.