

# **mysqlnd**

## **A new æra**

**Stuttgart**

**March, 23<sup>rd</sup> 2007**

**Andrey Hristov <[andrey@mysql.com](mailto:andrey@mysql.com)>**

## What is mysqlnd?

- Library and not a new extension
- (Almost) drop-in replacement for libmysql to be used for the PHP extensions needing connectivity to MySQL.
- Is not directly exposed to the users. ext/mysqli has been modified to be able to use libmysql or mysqlnd with a compile-time switch.
- PHP version independent, builds with PHP 5 and PHP 6. Probably will build with PHP 4 too.

## What are the pros?

- BSD licensed, so no more need for FOSS exceptions from MySQL AB.
- New versions, incl. bugfix ones, to be quite more often than the server releases.
- It takes quite more time for all the users to start to use a fixed libmysql, which implies installation of new server version, than just installing the latest version of mysqlnd.
- Tightly coupled with PHP and Zend Engine and doing magic to get more juice out of your CPU(s).
- mysqlnd can live with libmysql in the same process, shares no code, i.e. you have ext/mysql compiled in. There will be no port of ext/mysql to use mysqlnd, as ext/mysql is officially supported only for bugfixes.

## What are the cons?

- Can't think of any :-), but feel free to tell me if you find such. I will add them here.

## Abracadabra

- mysqlnd obeys to the PHP memory limit!
- mysqlnd uses PHP streams. They are robust.
- mysqlnd knows about connection persistency, and mysqli implements Pconns, the ext/mysql way.
- mysqlnd knows Unicode (compiled-out for PHP 5).
- mysqlnd implements read-only variables. Currently limited for normal queries and Unicode OFF
- mysqlnd does zval caching. ZE does something like this, but we bring it to an extreme.
- mysqlnd collects statistics, about few dozens about what has happened. Per process. In threaded environment, thread-safe and again per process. Available through two functions of just phpinfo(). :)

## Read-only variables (find the differences)

- mysqlnd implements a cool new feature, wanted in PHP 6 but still unimplemented.
- mysqlnd does less operations to perform the same work.
- What mysqli/libmysql does:
  - mysqli sends a query
  - result set fetched into libmysql buffers
  - mysqli allocates zvals, then new buffers
  - mysqli copies from libmysql to own buffers
  - mysqli calls `mysql_free_result()` and deallocates libmysql buffers.
- What mysqli/mysqlnd does:
  - mysqli sends a query
  - result set is fetched row by row, every row is different buffer
  - mysqlnd creates a result set of zvals, pointing to the buffers
  - mysqli calls `mysqlnd_free_result()` which could be lightweight

## Read-only variables (2)

### The differences

- `mysql/libmysql` does:
  - one extra allocation for `mysql` buffers
  - one extra data copy
  - one extra `zval` allocation, which can be saved with the `zval` cache
- What is suboptimal?
  - memory allocators, even in single-threaded applications are synchronised, thus slow.
  - `mysqlnd` allocates bigger chunks, thus minimizing heap fragmentation. Therefore, less work for the allocator.
  - `mysqlnd` saves tries to reuse the `zvals` to save CPU cycles and memory fragmentation.

# Read-only variables (3)

## How it works?

- A result set consists of
  - row buffers
  - zval buffer, 2D array
  - additional things
- Every zval points to memory inside the row buffers. The zval has `refcount == 1`, because it is owned by the result set.
- When fetch is performed, which is lightning fast, `refcount` becomes 2 or 3 (in case of NUM & ASSOC) and a hash including the zvals is returned. The hash is initialized with the correct number of fields, thus there will be no rehashing when the row grows, compared to `mysql`/`libmysql`.
- When the user frees the row or some of its elements the `refcount` decreases, but never reaches 0.
- When `mysql_free_result` is called, the result set is scanned and for zvals with `refcount > 1`, separation is performed (`zval_copy_ctor()`). So the zval won't point anymore to a row buffer. Then we free the row buffers.

## The zval cache

- On module start-up the zval cache is initialized. There is a ini setting for its size. Specific number of zvals are pre-allocated on a contiguous block of memory.
- When mysqlnd needs a zval, it asks the cache, if the cache has a free entry, it returns a pointer to one. Otherwise, creates a normal zval.
- When a result set is freed, the zvals are destructed through the cache. If a zval is not from the cache normal zval destruction procedure takes place. Of course we need to separate values, if refcount > 1.
- If the zval is from the cache, no efree() is called, which means no critical sections inside the allocator. The zval is ready to be used again.
- If a zval from the cache had refcount > 1, it won't be returned for re-usage till the end of the script. Limitation, which can be lifted in further versions by using garbage collection on `mysqlnd_free_result()`.

## The zval cache (2)

### More internals

- The cache consists of
  - Pre-allocated zval block
  - Free list stack
  - Reference counter, because the zval is passed from the driver to connections and result sets for usage.
  - Other variables for operation and statistics
- Because complexity of sequential scan of the block for finding free zval is  $O(n)$ , the access time will grow with the size. Thus, an optimisation called free-list-stack was introduced.
- Free-list-stack holds pointers to free zvals, thus *GET* operation has  $O(1)$  complexity.
- *PUT* is also is also leight-weight because the zval has to be added to the free-list.
- The stack is organised backward. New elements get lower memory addresses. Thus if there is a batch of *GET* operations the CPU will cache the line and but the first *GET* will hit the DATA L1/L2 cache.

## The zval cache (3)

### Problems

- Recall that on `mysql_free_result()` the result set has to be scanned and zvals with `refcount > 1` have to be separated. However, as there is (are) user variable(s) pointing to a zval cache entry, we cannot perform a *PUT* operation. Thus, we are stuck with this till the end of the script.
- This means that actually with `mysqlnd` it's better if the user doesn't call very quickly `mysql_free_result()`.
- This limitation can be lifted by adding additional stack with potentials to be free on next `mysql_free_result()`. This is a kind of a garbage collection which may kick in always on a probability basis.

# Statistics

mysql

Mysql Support		enabled
Client API library version	mysqlnd 5.1.0 - 070201 - \$Revision: 51	
Client statistics		
bytes_sent	0	
bytes_received	0	
packets_sent	0	
packets_received	0	
protocol_overhead_in	0	
protocol_overhead_out	0	
selects	0	
dml_ddls	0	
buffered_sets	0	
unbuffered_sets	0	
ps_buffered_sets	0	
ps_unbuffered_sets	0	
rows_fetched_from_server	0	
rows_fetched_from_client	0	
rows_skipped	0	
copy_on_write_saved	0	
copy_on_write_performed	0	
command_buffer_too_small	0	
connect_success	0	
connect_failure	0	
connection_reused	0	
Persistent cache		enabled
put_hits	0	
put_misses	0	
get_hits	0	
get_misses	0	
size	2000	
free_items	2000	
references	1	

Directive	Local Value	Master Value
mysql.cache_size	On	On
mysql.default_host	no value	no value
mysql.default_port	3306	3306
mysql.default_pw	no value	no value
mysql.default_socket	no value	no value
mysql.default_user	no value	no value
mysql.max_links	Unlimited	Unlimited
mysql.reconnect	Off	Off

## The future

- TTL Query Cache
- PS handle cache

## Who is behind this?

- Georg Richter – Management and development
- Andrey Hristov – Architecture and development
- Ulf Wendel – QA, the dirty work which brings sweet and tasteful fruit.

## Questions ?

- Next release : March 26th