

memcached Functions For MySQL: Seamless caching for MySQL

Patrick Galbraith, Lycos Inc.

About the speaker

Patrick Galbraith

- Principal Software Engineer, Lycos
- 16 Years dabbling in Open Source
- Author of Developing Web Applications using..
- Federated Storage Engine, Memcached Functions for MySQL/UDFs, DBD::mysql...



What are the memcached Functions for MySQL?

- A suite of functions available to use with MySQL that allow you to store, retrieve and delete data, as well as most of the functions/operations that are available with libmemcached such as server connectivity to the client, server status, client behaviors and more.
- The power of the SQL engine which can be used to initiate caching or data retrieval using the result sets from query.
- You can combine the fetching of data from one or more tables with the fetching of data from memcached and be able to apply any SQL operations on that result set such as LIMIT, sorting and other conditional operations.

What are the memcached Functions for MySQL?

- Written in C using libmemcached and the MySQL UDF API
- Provide get, set, delete, replace, add, flush, stats, behavior setting and retrieval, as well as other functionality
- Can be used in stored procedures and triggers
- Allow one to interact with Memcached and MySQL independent of the language the application using them is written in, or for languages that don't have clients for memcached
- Open Source !!!

What is memcached?

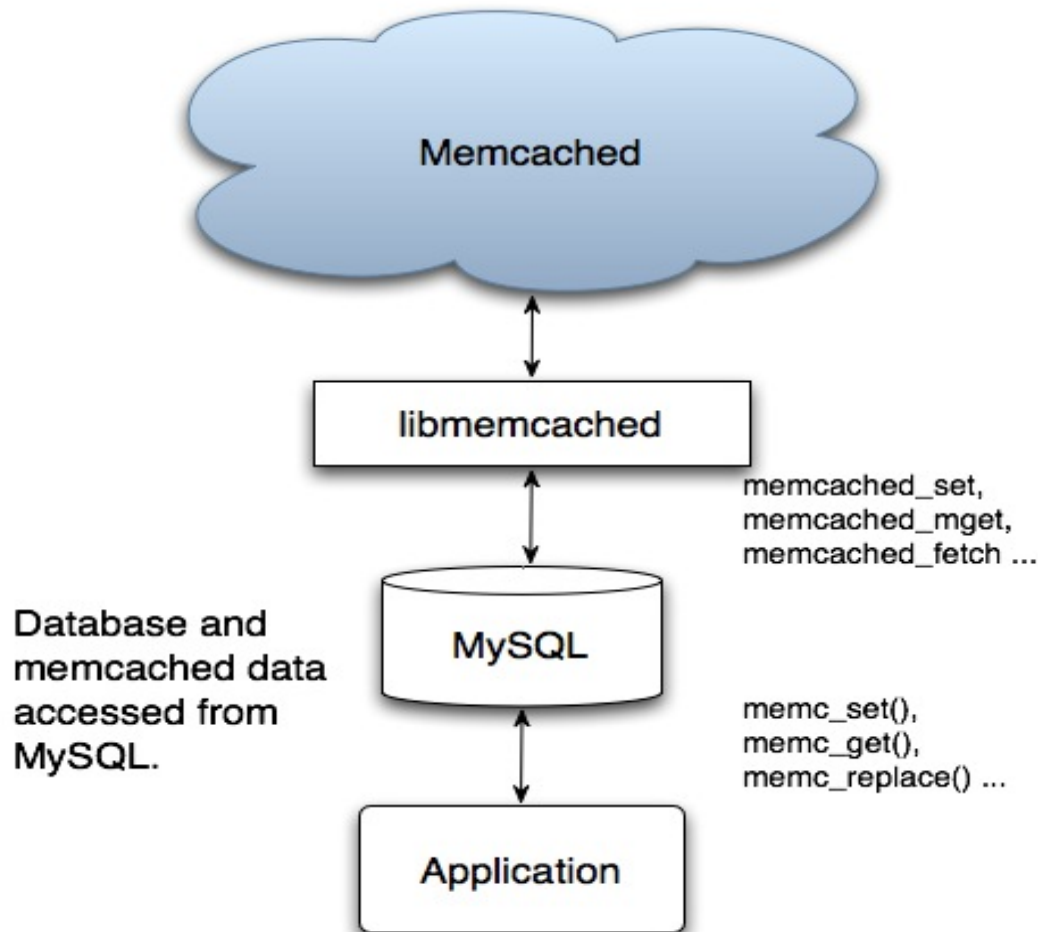
- a high-performance, distributed memory object caching system
- Simply a memory server that provides caching layer for applications to store data to alleviate database load, as well as providing a dictionary, or hash lookup table
- The data stored in memcached is not durable—meaning it's gone when the memcached server is shut down or restarted, as well as having no failover or authentication
- Is an LRU (Least Recently Used) cache, which means that stored data that is the oldest and least accessed will be replaced with newer items when memcached's memory capacity is reached.
- provides expiration timeouts for stored data

What is libmemcached?

- faster memcached client library written in C.
- faster, more efficient, thread-safe, full-featured C library that has a significant performance gain over existing client libraries.
- more control to affect how the client functions by being able to set the client behavior (`memcached_behavior_set()`) with numerous behavior settings such as hashing algorithm or whether the client is blocking or non-blocking,
- CAS (Check and Set) support, server host sorting, etc.

How do these UDFs work?

Using both MySQL UDF API and libmemcached



Installation

- Obtain source
- Configure, compile, and install code
- Install functions
 - SQL install script
 - Perl installation utility
- Post installation check

Obtain source

http://patg.net/downloads/memcached_functions_mysql-0.9.tar.gz

or

http://download.tangent.org/memcached_functions_mysql-0.8.tar.gz

Configure, compile and install the code

Run configure, make, make install

```
./configure --with-  
mysql=/usr/local/mysql/bin/mysql_config --  
libdir=/usr/local/mysql/lib/mysql/plugin/
```

```
make
```

```
make install
```

Install the functions

- Using SQL file:

```
mysql -uroot -ps3kr1t < sql/install_functions.sql
```

- Using the Perl installation utility:

```
./utils/install.pl --host=localhost --  
user=user --password=s3kr1t
```

(see ./utils/install.pl --help for usage)

Post installation check

Check the mysql schema func table:

```
mysql> select name, dl from mysql.func where  
name like 'memc%';
```

Using the UDFs

- Connection
- Verifying the connection
- Setting values
- Getting/fetching values
- Increment/Decrement
- Behavioral functions
- Statistical functions
- Utility/Version functions

Connection

```
mysql> select memc_servers_set('127.0.0.1:11211');
```

```
+-----+
| memc_servers_set('127.0.0.1:11211') |
+-----+
|                                     0 |
+-----+
```

Connection

```
mysql> select memc_servers_set('127.0.0.1:
11211,
127.0.0.1:112112,
127.0.0.1:112113,
127.0.0.1:112114,
10.20.0.246:11211');
```

Verifying the connection

```
mysql> select memc_server_count();  
+-----+  
| memc_server_count() |  
+-----+  
|                    5 |  
+-----+
```


CAS - check and switch

"cas is a check and set operation which means 'store this data but only if no one else has updated since I last fetched it.'"

```
select memc_behavior_set('MEMCACHED_BEHAVIOR_SUPPORT_CAS', 1);
```

Prepend, append

```
select memc_set('foo1', ' this value ');
```

```
select memc_prepend('foo1', 'before');
```

```
select memc_append('foo1', 'after');
```

Getting/fetching values

```
mysql> select memc_get('foo1');  
+-----+  
| memc_get('foo1') |  
+-----+  
| before this value after |  
+-----+
```

Incrementing values

```
select memc_set('counter', 0);  
select memc_increment('counter', 1);  
select memc_increment('counter', 1);  
select memc_increment('counter', 20);
```

```
select memc_decrement('counter');  
select memc_decrement('counter', 5);
```

...

Behavioral functions

```
mysql> select memc_list_behaviors()\G
***** 1. row *****
memc_list_behaviors():
MEMCACHED_SERVER_BEHAVIORS
MEMCACHED_BEHAVIOR_SUPPORT_CAS
MEMCACHED_BEHAVIOR_NO_BLOCK
MEMCACHED_BEHAVIOR_TCP_NODELAY
MEMCACHED_BEHAVIOR_HASH
MEMCACHED_BEHAVIOR_CACHE_LOOKUPS
MEMCACHED_BEHAVIOR_SOCKET_SEND_SIZE
MEMCACHED_BEHAVIOR_SOCKET_RECV_SIZE
MEMCACHED_BEHAVIOR_BUFFER_REQUESTS
MEMCACHED_BEHAVIOR_KETAMA
MEMCACHED_BEHAVIOR_POLL_TIMEOUT
MEMCACHED_BEHAVIOR_RETRY_TIMEOUT
MEMCACHED_BEHAVIOR_DISTRIBUTION
MEMCACHED_BEHAVIOR_BUFFER_REQUESTS
...
```


Distribution types

This function allows you see the different means of distributing values to servers for use with memc_behavior_set

```
mysql> select memc_list_distribution_types()\G
***** 1. row *****
```

```
memc_list_distribution_types():
MEMACHED_DISTRIBUTION_MODULA
MEMACHED_DISTRIBUTION_CONSISTENT
MEMACHED_DISTRIBUTION_KETAMA
```

```
mysql> select memc_servers_behavior_get
('MEMACHED_BEHAVIOR_DISTRIBUTION');
```

```
+-----+
| memc_servers_behavior_get('MEMACHED_BEHAVIOR_DISTRIBUTION') |
+-----+
| MEMACHED_DISTRIBUTION_MODULA |
+-----+
```

Hashing types

```
mysql> select memc_list_hash_types()\G
***** 1. row *****
memc_list_hash_types():
MEMCACHED_HASH_DEFAULT
MEMCACHED_HASH_MD5
MEMCACHED_HASH_CRC
MEMCACHED_HASH_FNV1_64
MEMCACHED_HASH_FNV1A_64
MEMCACHED_HASH_FNV1_32
MEMCACHED_HASH_FNV1A_32
MEMCACHED_HASH_JENKINS
MEMCACHED_HASH_HSI
```

Hashing types

```
mysql> select memc_servers_behavior_get('MEMCACHED_BEHAVIOR_HASH');
```

```
+-----+
| memc_servers_behavior_get('MEMCACHED_BEHAVIOR_HASH') |
+-----+
| MEMCACHED_HASH_DEFAULT                               |
+-----+
```

```
1 row in set (0.00 sec)
```

```
mysql> select memc_servers_behavior_set('MEMCACHED_BEHAVIOR_HASH',
'MEMCACHED_HASH_DEFAULT');
```

Statistical functions

```
select memc_stats('127.0.0.1:11214');
```

```
Server: 127.0.0.1 (11214)  
  pid: 15470  
  uptime: 1634  
  time: 1240205306  
  version: 1.2.6  
  pointer_size: 64  
  rusage_user: 0.15997  
  rusage_system: 0.23996  
  curr_items: 8  
  total_items: 8  
  bytes: 640  
  curr_connections: 4  
  total_connections: 27  
  connection_structures: 5  
  ...
```

Statistical functions

```
mysql> select memc_stat_get_keys();
```

```
mysql> select memc_stat_get_value('127.0.0.1:11214',  
'total_items');
```

```
+-----+  
| memc_stat_get_value('127.0.0.1:11214', 'total_items') |  
+-----+  
| 8 |  
+-----+
```

Version info

```
mysql> select memc_udf_version();
```

```
+-----+  
| memc_udf_version() |  
+-----+  
| 0.9                |  
+-----+
```

```
mysql> select memc_server_version('127.0.0.1:11212')
```

```
mysql> select memc_libmemcached_version();
```

```
+-----+  
| memc_libmemcached_version() |  
+-----+  
| 0.27                        |  
+-----+
```

Application

- * Triggers (see `sql/trigger_fun.sql`)
- * Stored procedures
- * You can use these with DBI programs, storing Perl objects using storable
- * Aggregation in select
- * Anything you can think of!

Serialization (set)

```
sub memc_set {
    my ($this, $key, $value) = @_ ;
    my $sth = $this->{dbh}->prepare_cached
('SELECT memc_set(?, ?)');

    my $storable_value = ref $value ?
        nfreeze($value) : $value;
    $sth->execute($key, $storable_value);
    my $stored = $sth->fetchrow_arrayref();
    # returns 0 on success, greater than 0 error
    return $stored->[0];
}
```

Serialization (get)

```
sub memc_get {
    my ($this, $key) = @_;
    my $de_serialized;
    my $sth = $this->{dbh}->prepare_cached('SELECT memc_get
(?) ');

    $sth->execute($key);
    my $stored = $sth->fetchrow_arrayref();
    # will be 1 or 0

    if (defined $stored->[0]) {
        eval { $de_serialized = thaw($stored->[0]) };
        return $@ ? $stored->[0] : $de_serialized;
    }
    else {
        return undef;
    }
}
```

Using with other UDFs (Gearman!)

```
DELIMITER |
DROP TRIGGER IF EXISTS urls_stored_insert |
CREATE TRIGGER urls_stored_insert
BEFORE INSERT ON urls_stored
FOR EACH ROW BEGIN
    SET @count = memc_increment('urls_stored');
    SET @index_count = memc_increment('index_counter');
    IF @count > 5 THEN
        SET @gd= gman_servers_set('127.0.0.1:4730');
        SET @gd= gman_do_background('url_process', NEW.url);
        SET @reset = memc_set('urls_stored', 0);
    END IF;
    IF @index_count > 20 THEN
        SET @gd= gman_servers_set('127.0.0.1:4730');
        SET @gd = gman_do_background('indexer', NEW.url);
        SET @reset = memc_set('index_counter', 0);
    END IF;
END |
```

Gearman example

