
MySQL™ Connect

ORACLE

Enhancing Productivity with MySQL 5.6 New Features

Arnaud ADANT
MySQL Principal Support Engineer

Safe Harbour Statement

THE FOLLOWING IS INTENDED TO OUTLINE OUR GENERAL PRODUCT DIRECTION. IT IS INTENDED FOR INFORMATION PURPOSES ONLY, AND MAY NOT BE INCORPORATED INTO ANY CONTRACT. IT IS NOT A COMMITMENT TO DELIVER ANY MATERIAL, CODE, OR FUNCTIONALITY, AND SHOULD NOT BE RELIED UPON IN MAKING PURCHASING DECISIONS. THE DEVELOPMENT, RELEASE, AND TIMING OF ANY FEATURES OR FUNCTIONALITY DESCRIBED FOR ORACLE'S PRODUCTS REMAINS AT THE SOLE DISCRETION OF ORACLE.

Program Agenda (part 1)

- Introduction
- InnoDB Memcached Plugin
- InnoDB Full Text Indexes
- Major Optimizer Enhancements
- Enhanced Explain and Tracing

Program Agenda (part 2)

- Persistent statistics
- Managing tablespaces and partitions
- Online Schema changes
- Performance_schema in practice
- Q & A

Introduction : Who I am

Arnaud ADANT

- http://blog.aadant.com
- 10 year+ Development
- MySQL Support for 3 years
- MySQL Performance
- I love my job !

Introduction : About this tutorial

Productivity oriented

- **Examples** are worth a thousand words
- **Web App demo** using php
- The source code will be available on my blog : blog.aadant.com

InnoDB Memcached Plugin

- About Memcached
- What is it ?
- Benefits / Limitations
- How to configure ?
- How to use ?
- Demo

About memcached

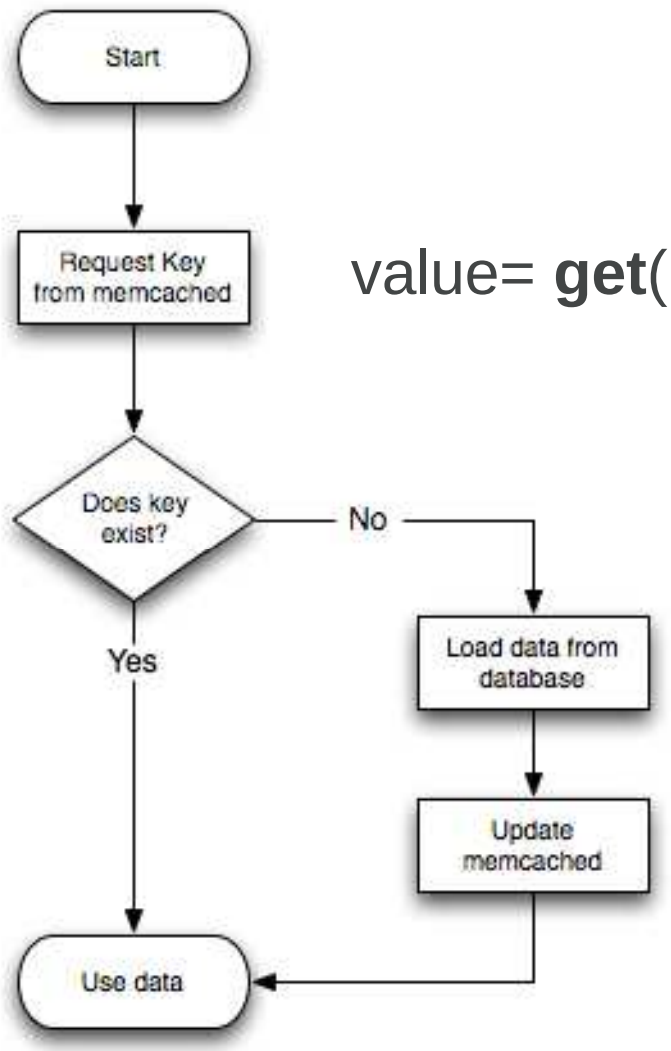
Popular caching framework

- In-memory caching
- Open source
- Widely used
- Invented by **Brian Aker**,
  - Former Director of Architecture at **MySQL AB**
- Wikipedia Flickr … Twitter … Youtube

Memcached principle

Popular caching framework

- value = **get**(key)
- if undefined value
  - load data
  - **set**(key, value)
- return the value



value= **get**(key)

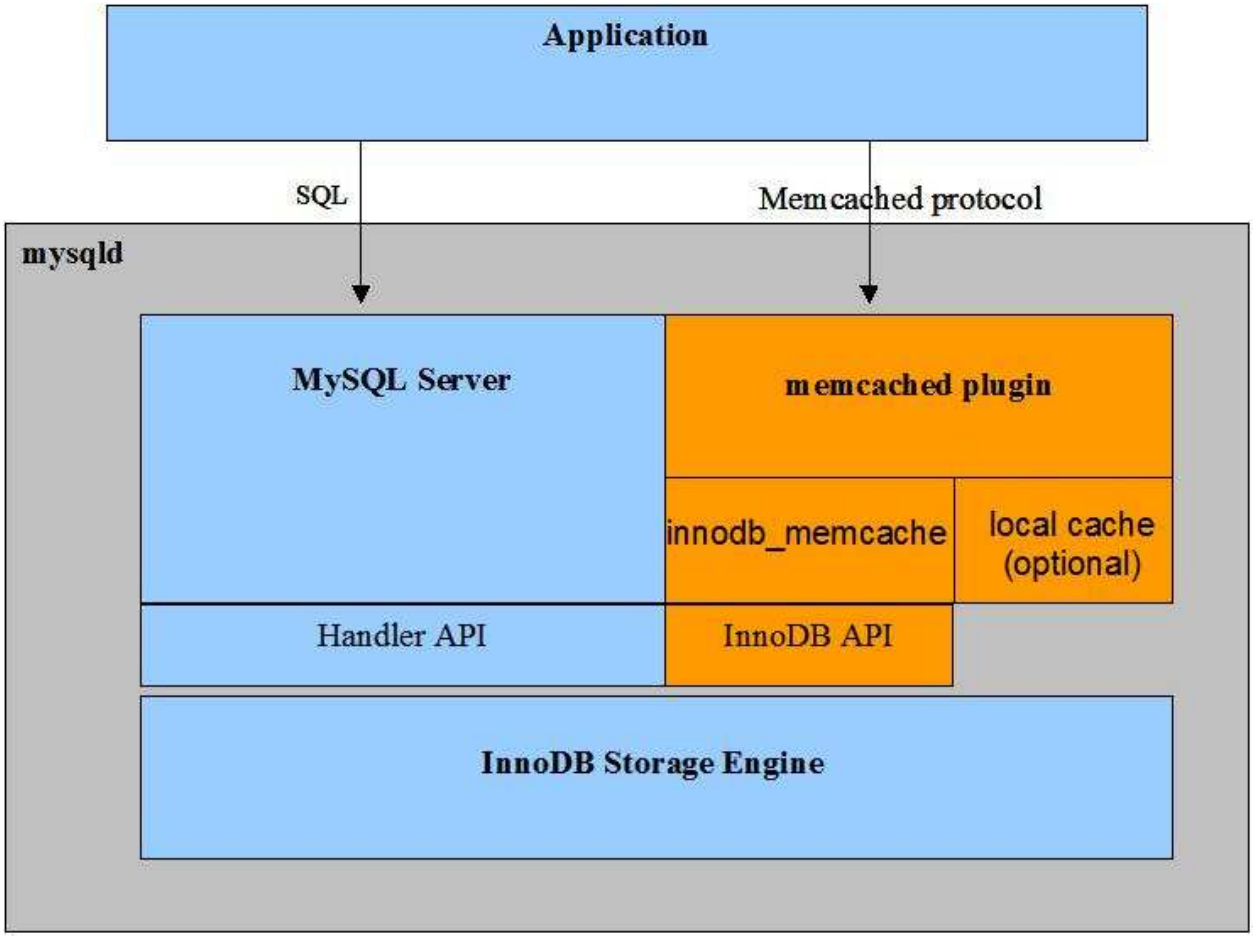
set(key,value)

InnoDB Memcached plugin : what is it ?

A new way to connect to MySQL !

- NoSQL !
- In 5.6 GA, production ready
- Implements the memcached protocol (memcapable)
- **Plugin** architecture on top of the **InnoDB** engine
- MySQL becomes
  - a **key - value store**
  - a **memcached** server

InnoDB Memcached plugin : architecture



port 3306

listener, parser, optimizer

port 11211

listener

cache(s)

direct access to InnoDB

Memcached to InnoDB translation

memcached	InnoDB DML implementation	SQL equivalent
get (k)	a read/fetch command	SELECT value FROM t WHERE key = k
set (k,v)	a search followed by an insertion or update (depending on whether or not a key exists)	INSERT INTO t(key,value) VALUES(v,k) ON DUPLICATE KEY UPDATE set key=v;
add (k, v)	a search followed by an insertion or update	INSERT INTO t(key, value) VALUES (v, k)
flush_all	truncate table	TRUNCATE TABLE t

InnoDB Memcached plugin : namespaces

memcached	InnoDB DML implementation
get ("@@container_name.key")	Read the value of key in the specified container
get ("@@container_name")	Select container_name as default container for the Transaction. No value returned
get ("key")	Read the value of key in the default container.

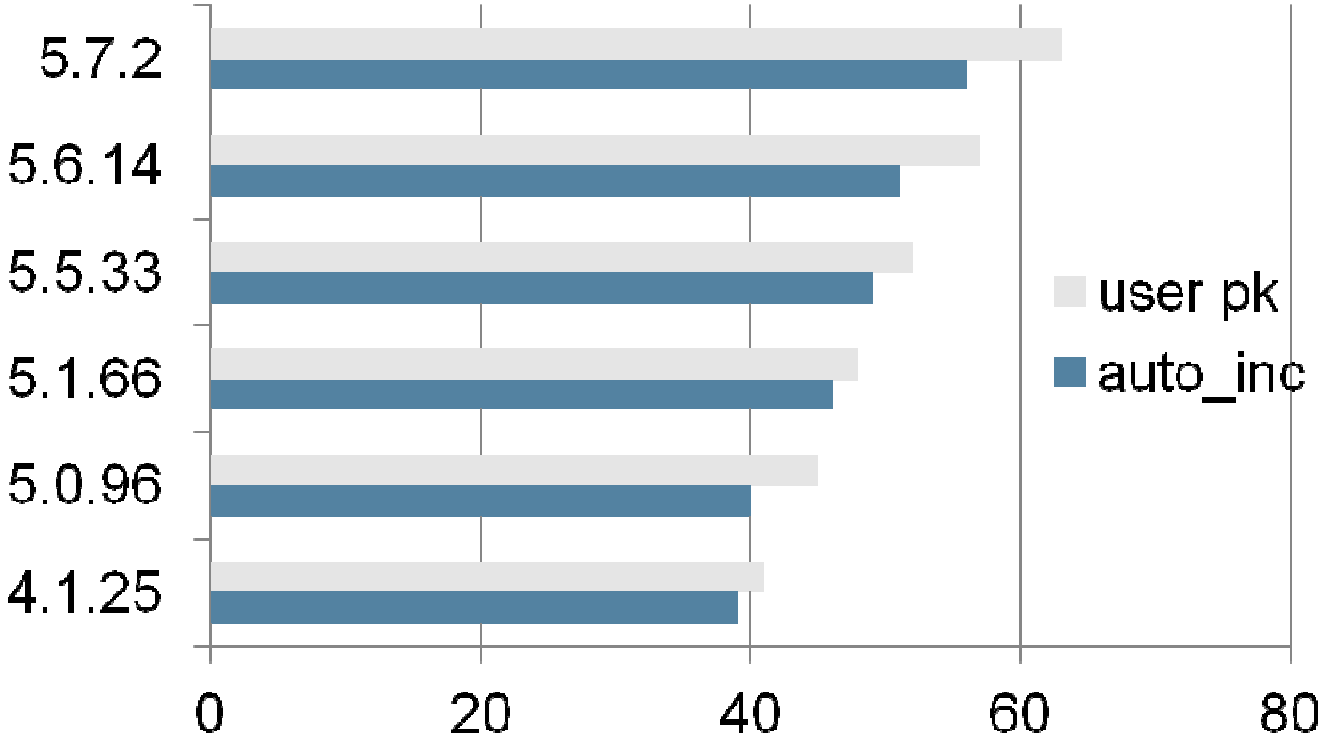
InnoDB Memcached plugin : introduction

Performance

- Single threaded performance decreases with releases
  - features
  - more complex code
  - overhead
  - Meta Data Locking from 5.5
- The memcached plugin by-passes
  - the SQL layer (optimizer, handler interface, …)
- Multi-threaded performance

Single threaded insert performance (SQL)

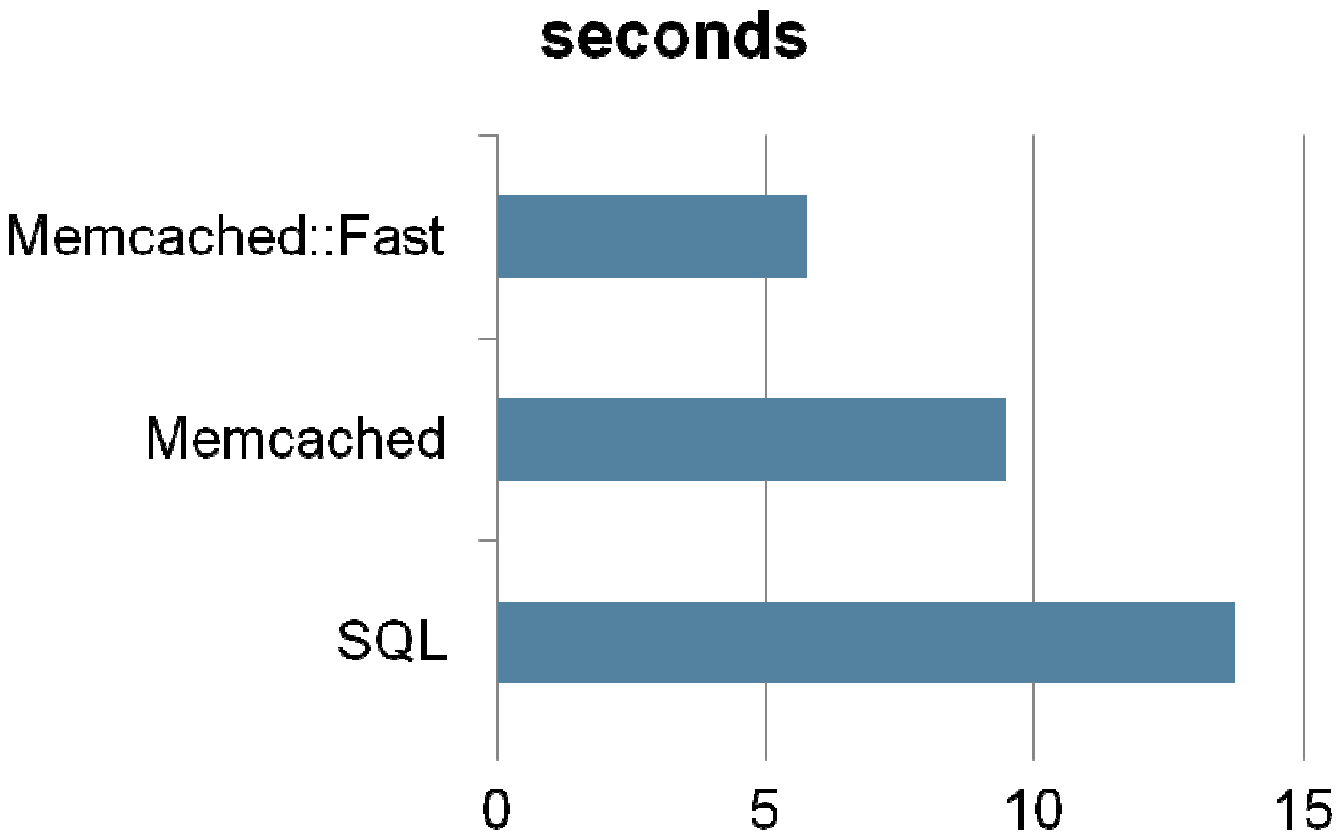
- 500k inserts
- autocommit = 1
- innodb_flush_log_at_trx_commit = 2



MySQL Connect ORACLE

Single threaded insert performance

- PERL
- 100k inserts
- Commit every 20k
- 3 x faster with Cache::Memcached::fast
- 5.6.14
- 5.7.2



PHP code using memcached (NoSQL)

PECL memcache

```
<?php
$cache = new Memcache;
$cache->connect('localhost',11211);
$cache->get('@@test');
$cache->flush(0);
$handle = @fopen("/home/aadant/list5.csv","r");
if ($handle) {
    while (($buffer = fgets($handle, 4096)) !== false){
        list($key,$value) = split("\t",$buffer);
        $cache->add($key,$value);
    }
    fclose($handle);
}?>
```

PHP code using mysqli (SQL)

```php
<?php
$mysqli = new mysqli("localhost", "root", "", "test_case",0,"/tmp/mysql.sock");
$mysqli->query("TRUNCATE TABLE t1");
$mysqli->query("set autocommit = 0");
$handle = @fopen("/home/aadant/list5.csv", "r");
if ($handle) {
    $i=0;
    while (($buffer = fgets($handle, 4096)) !== false) {
        list($key,$value) = split("\t",$buffer);
        $mysqli->query('insert into t1(k,value) values('".$key."','".$value."');");
        $i = $i + 1;
        if ($i % 20000 == 0){
            $mysqli->commit();
        }
    }
    fclose($handle);
}
$mysqli->commit();
$mysqli->close();
?>
```

InnoDB Memcached plugin : benefits

More productivity for the developers

- simple API (get, set, delete, flush_all, …)
- community available
- new connectors to MySQL (C, C++, python, php, perl, java …)
- multi-columns support
  - k1|k2|ki
  - v1|v2|vi
- compatible with MySQL replication
- high performance from 5.6.14

InnoDB Memcached plugin : benefits

Combined with other 5.6 features

- **Flexible persistent key value store**
  - Fast warm up
  - Even better with :
    - ***innodb_buffer_pool_load_at_startup***
    - ***innodb_buffer_pool_dump_at_shutdown***
    - ***innodb_buffer_pool_dump_now***
    - ***innodb_buffer_pool_load_now***

InnoDB Memcached API : limitations

Do not cover all use cases yet !

- The key
  - **non-null varchar** only
- The value
  - a **char**, **varchar** or **blob**
- Windows is *not* supported
  - the **memcached** Daemon Plugin is only supported on Linux, Solaris, and OS X platforms.
- Less secure than SQL, same security as memcached

InnoDB Memcached : how to install ?

Bundled from 5.6

- 2 dynamic libraries in lib/plugin
  - **libmemcached.so** : the plugin / listener
  - **innodb_memcached.so** : the cache
- Available in community and enterprise builds
- tar.gz packages have no dependency
- RPMs need ***libevent-dev***

InnoDB Memcached : how to configure ? (1)

Can be configured dynamically

- Cleaning

```
drop database if exists innodb_memcache;
drop table test.demo_test;
```

- An initialization script is required (innodb_memcache db)

```
source share/innodb_memcached_config.sql;
```

- Uninstall (in case of re-configuration)

```
uninstall plugin daemon_memcached;
```

InnoDB Memcached : how to configure ? (2)

Design the container table

```
use test_case; drop table if exists t1;
CREATE TABLE t1 (
  k varchar(50),
  value varchar(1000),
  c3 int(11),
  c4 bigint(20) unsigned,
  c5 int(11),
  unique KEY k (k)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

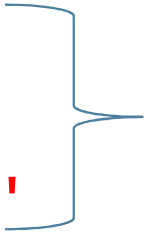
memcached protocol *mandatory* columns
flags, cas, exp

InnoDB Memcached : how to configure ? (3)

Container configuration

```
USE innodb_memcache; TRUNCATE TABLE containers;
INSERT INTO containers
SET    name = 'test',
       db_schema = 'test_case', db_table = 't1',
       key_columns = 'k',
       value_columns = 'value',
       unique_idx_name_on_key = 'k'
       flags = 'c3',
       cas_column = 'c4',
       expire_time_column = 'c5';
```

namespace



key, value, uk/pk

memcached columns

Innodb_memcache.containers

Table structure

```
mysql> desc containers;
+----------------------+-------------+------+-----+---------+-------+
| Field                | Type        | Null | Key | Default | Extra |
+----------------------+-------------+------+-----+---------+-------+
| name                 | varchar(50)  | NO   | PRI | NULL    |       |
| db_schema            | varchar(250) | NO   |     | NULL    |       |
| db_table             | varchar(250) | NO   |     | NULL    |       |
| key_columns          | varchar(250) | NO   |     | NULL    |       |
| value_columns        | varchar(250) | YES  |     | NULL    |       |
| flags                | varchar(250) | NO   |     | 0       |       |
| cas_column           | varchar(250) | YES  |     | NULL    |       |
| expire_time_column   | varchar(250) | YES  |     | NULL    |       |
| unique_idx_name_on_key | varchar(250) | NO   |     | NULL    |       |
+----------------------+-------------+------+-----+---------+-------+
```

InnoDB Memcached : how to configure ? (4)

Restart the memcached plugin

- Dynamic install

```
uninstall plugin daemon_memcached;
install plugin daemon_memcached soname "libmemcached.so";
```

- Configuration file

```
plugin-load=libmemcached.so
```

InnoDB Memcached : advanced options (1)

Configuration file option	Default value	Description
daemon_memcached_r_batch_size	1	Commit after N reads
daemon_memcached_w_batch_size	1	Commit after N writes
innodb_api_trx_level	0 = READ UNCOMMITTED	Transaction isolation (1, 2, 3, 4 to SERIALIZABLE)
innodb_api_enable_mdl	0 = off	MDL locking
innodb_api_disable_rowlock	0 = off	**Row level locking in ON**
innodb_api_enable_binlog	0 = off	Replication (ROW format)
daemon_memcached_option		Options passed to the memcached daemon

InnoDB Memcached : advanced options (2)

Security, config options, cache policies

- SASL security : using daemon_memcached_option=-S

```
mysql> select * from innodb_memcache.config_options;
+--------------------+-------+
| name               | value |
+--------------------+-------+
| separator          | |     |
| table_map_delimiter | .     |
+--------------------+-------+
2 rows in set (0.01 sec)

mysql> select * from innodb_memcache.cache_policies;
+--------------+-------------+-------------+---------------+--------------+
| policy_name  | get_policy  | set_policy  | delete_policy | flush_policy |
+--------------+-------------+-------------+---------------+--------------+
| cache_policy | innodb_only | innodb_only | innodb_only   | innodb_only  |
+--------------+-------------+-------------+---------------+--------------+
1 row in set (0.00 sec)
```

Multi-column separator
Namespace delimiter : get @@t1.some_key

Memcached expiry off : innodb_only
Traditional memcached : cache
Memcached expiry on : caching

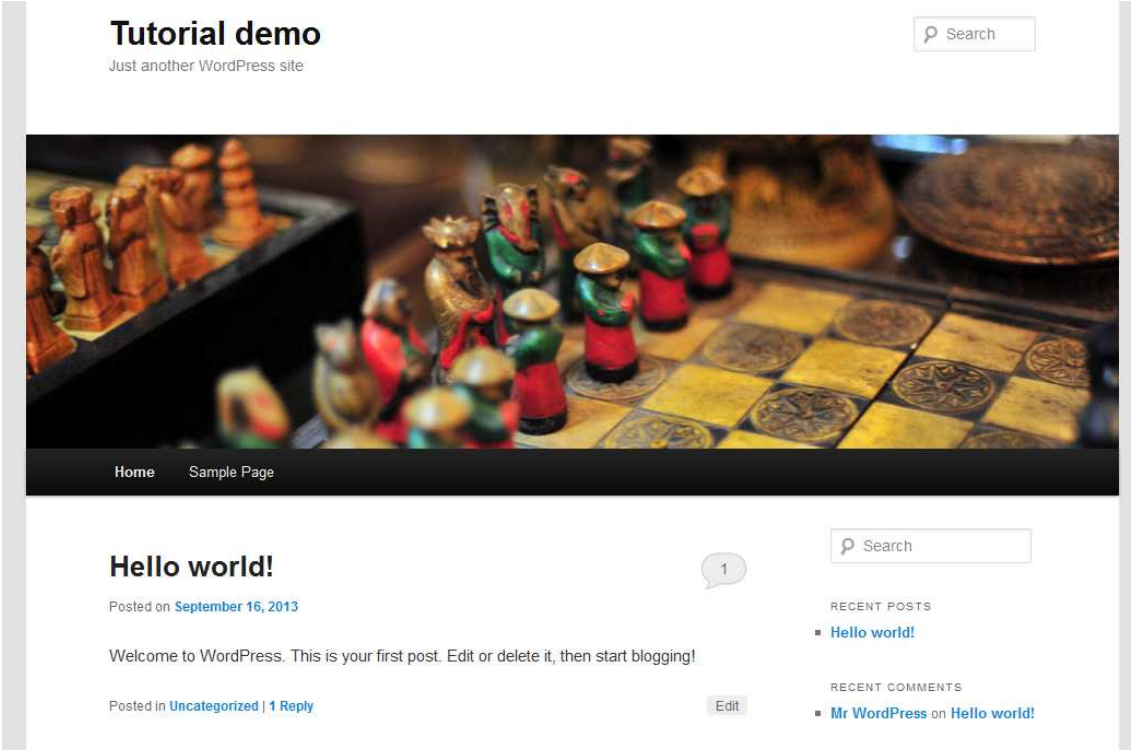
InnoDB Memcached : how to use it?

You can use it with any memcached clients

- telnet port 11211
- libmemcached executables (memcat, memcp, memrm, …)
- PHP
  - PECL memcache / memcached
- PERL
  - Cache::Memcached
  - Cache::Memcached::Fast
- Java
- C / C++, Python, Ruby, .NET ....

InnoDB Memcached : demo (0)

Default blog page = 7k



 |

MySQL Connect ORACLE

InnoDB Memcached : demo (1)

Adding a persistent memcached to WordPress

- Apache 2.x + PHP 5.3 + PECL memcached
- MySQL 5.6.14 advanced or community
- WordPress
  - wordpress-3.4-RC2.zip
- WordPress Memcached plugin : memcached.2.0.2

InnoDB Memcached : demo (2)

Configuring MySQL

```
drop database if exists innodb_memcache; drop table if exists test.demo_test;
source  share/innodb_memcached_config.sql; drop database if exists test_case;
create database test_case; use test_case; drop table if exists t1;
CREATE TABLE `t1` (
  `k` varchar(50) PRIMARY KEY,
  `value` BLOB DEFAULT NULL,
  `c3` int(11) DEFAULT NULL,
  `c4` bigint(20) unsigned DEFAULT NULL,
  `c5` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
use innodb_memcache; truncate table containers;
INSERT INTO containers VALUES
("test", "test_case", "t1","k","value", "c3",  "c4","c5", "primary");
```

Primary key
BLOB storage

InnoDB Memcached : demo (3)

Configuring MySQL

plugin-load=libmemcached.so

daemon_memcached_w_batch_size=20000

daemon_memcached_r_batch_size=20000

Configuring the WordPres memcached plugin

- install the plugin
- do not activate
- cd /var/www/html/wordpress/wp-content
- cp plugins/memcached/object-cache.php .
- more configuration
  - see plugins/memcached/readme.txt

InnoDB Memcached : demo (4)

How to test ?

- /etc/init.d/httpd restart
- Using Apache Benchmark
  - ab -n 10000 -c 10 http://127.0.0.1/wordpress/
  - compare global handlers and temporary tables

InnoDB Memcached : demo (5)

Results

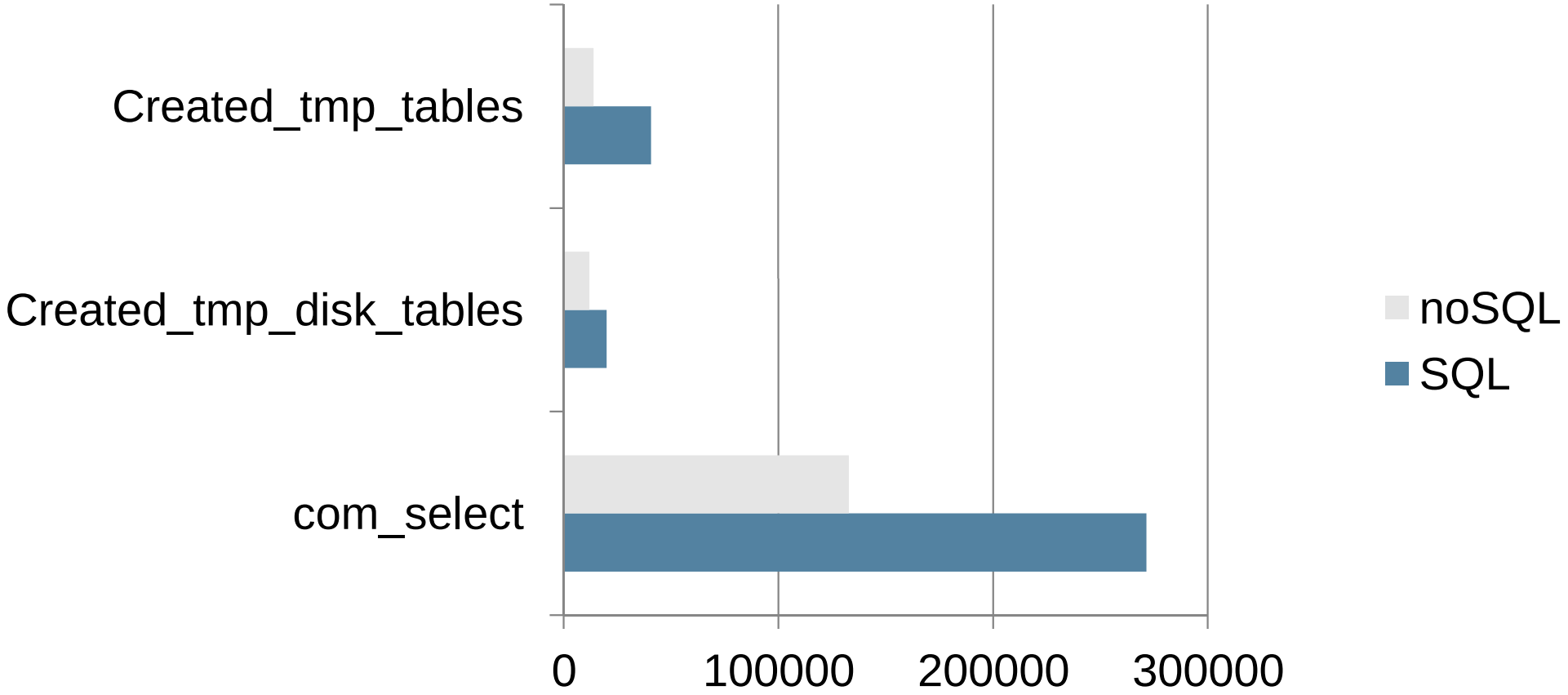
- memcached is not faster than plain SQL on the test machine :

  memcached : Requests per second: 19.56 [#/sec] (mean)
  SQL Requests per second: 19.81 [#/sec] (mean)

- however, the number of handlers and temporary tables on disk decreased
- will use the **performance_schema** to troubleshoot (P_S in practice)

InnoDB Memcached : demo (6)

Results ab -n 10000 -c 10 http://127.0.0.1/wordpress/



InnoDB Full-Text Indexes

- Introduction
- Benefits / limitations
- Indexing / Searching
- Sorting
- Advanced configuration
- Demo

Introduction

Full text indexes are useful for text processing

- Before 5.6, only supported on MyISAM
- FTS optimizes textual search
  - LIKE %search% requires a full table scan
  - very uneffective on big tables
- In 5.6, InnoDB full text indexes
  - Same features as MyISAM
  - Drop-in replacement
  - More scalable
  - Less reasons to keep MyISAM

InnoDB FTS : implementation (from J. Yang)

Inverted indexes

- Incoming text strings are tokenized into individual words
- Words are stored in one or more auxiliary tables.
- For each word, a list of Document IDs and word position pairs is stored

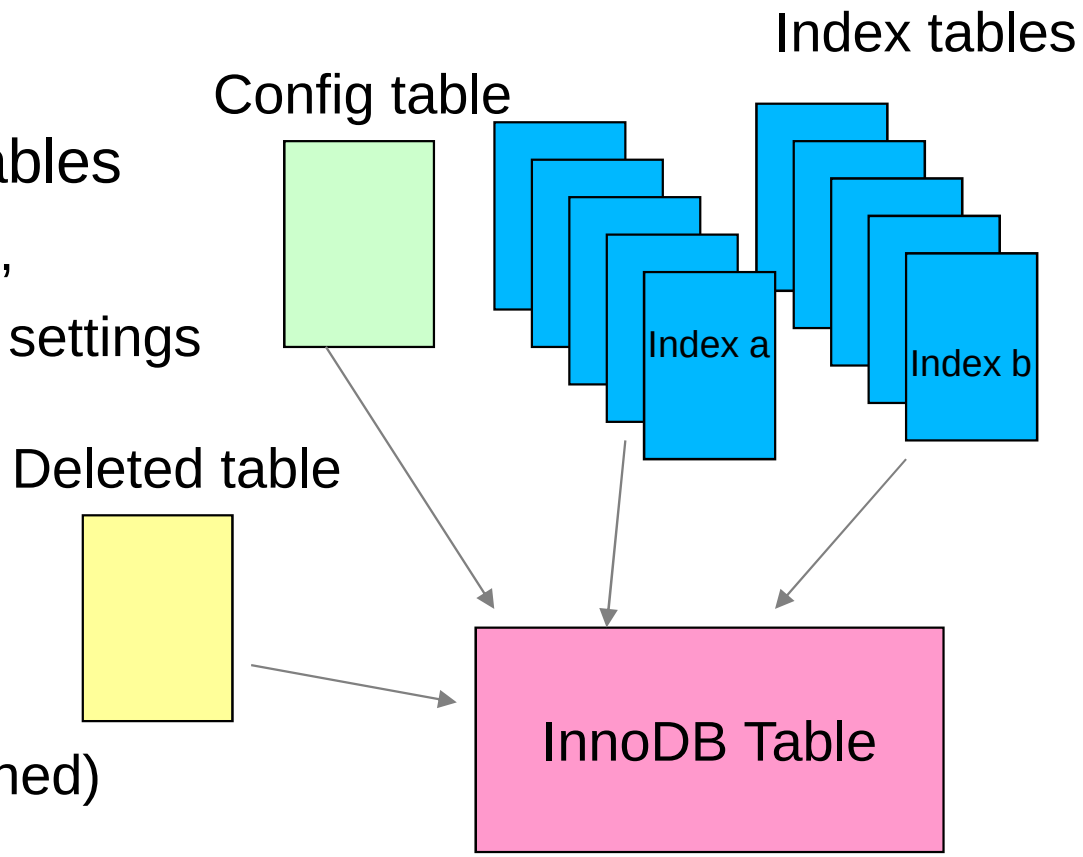
Word	DOC ID	Pos
and	2	16
database	2	7
fulltext	2	20
mysql	2	0
search	2	28

2	mysql database and fulltext search

InnoDB FTS : implementation (from J.Yang)

Auxiliary tables

- There two types of Auxiliary tables
  - Table specific auxiliary tables,
    - managing table wise FTS settings
    - config table
    - deleted table
- Index specific auxiliary tables
  - The actual index tables
  - Multiple Index tables (partitioned)



InnoDB FTS : benefits

Supersedes MyISAM implementation

- MyISAM implementation replacement (not drop-in)
- Full transactional support
- InnoDB scalabilty on cores
- Parallel indexing
- Better performance than MyISAM
- Additional features over MyISAM (proximity search)
- Easier to troubleshoot indexing
  - auxiliary tables

InnoDB FTS : limitations

- Basically the same as the MyISAM implementation
  - No custom tokenization (language dependent).
  - No index based custom sort
- Not supported for partitioned tables
- Can slow down the transaction commit (changes in auxiliary tables)
  - Note the the indexing is not done at commit time
- Increased memory usage
- Same character set for indexed columns

InnoDB FTS : indexing (from J.Yang)

2 ways to do

1.
CREATE TABLE *tbl_name*
(*ol_name column_definition*
FULLTEXT [INDEX|KEY] [*index_name*]
(*index_col_name*,...));

2.
CREATE FULLTEXT INDEX
index_name ON *tbl_name*
(*index_col_name*,...);

ALTER TABLE *tbl_name*
ADD FULLTEXT INDEX
index_name (*index_col_name*,...)

InnoDB FTS : searching

Natural language search

```
SELECT * FROM articles
    WHERE MATCH (title,body)AGAINST
  ('run mysqld as root' IN NATURAL LANGUAGE MODE);

SELECT * FROM articles
    WHERE MATCH (title,body)AGAINST
  ('dbms stand for' IN NATURAL LANGUAGE MODE WITH
 QUERY EXPANSION);
```

InnoDB FTS : searching

Boolean search

```
SELECT * FROM articles WHERE MATCH (title,body)
   AGAINST ('+MySQL -YourSQL' IN BOOLEAN MODE);

SELECT * FROM articles WHERE MATCH (title,body)
 AGAINST('"following comparison" @8' IN BOOLEAN
 MODE);
```

Boolean operators (from J.Yang)

- “+” A leading plus sign indicates that this word *must* be present in each row that is returned
- “-” A leading minus sign indicates that this word must *not* be present in any of the rows that are returned
- “> <“ These two operators are used to change a word's contribution to the relevance value that is assigned to a row
- “~” A leading tilde acts as a negation operator, causing the word's contribution to the row's relevance to be negative
- “*” Wildcard search operator

InnoDB FTS : sorting

Automatic sorting by relevance desc

- rows sorted with the highest relevance first (IDF algorithm)
- relevance
  - nonnegative floating-point numbers.
- zero relevance means no similarity.
- >< operators alter relevance
  - +apple +(>turnover <strudel)'
- ~ suppress the effect on relevance
- custom sort via ORDER BY (slower)

InnoDB FTS : advanced configuration

Relevant options only

Configuration file option	Default value	Description
Innodb_optimize_fulltext_only	**0**	OPTIMIZE TABLE only optimize the full text index
innodb_ft_num_word_optimize	**20000**	Number of words per OPTIMIZE TABLE
innodb_ft_min_token_size	3 (MyISAM 4)	Maximum token size
innodb_ft_aux_table		Dynamic variable to access to the index tables via INFORMATION_SCHEMA
innodb_ft_sort_pll_degree	2	Number of threads at index creation
Innodb_ft_cache_size	32M	Holds the tokenized document in memory
innodb_sort_buffer_size	1M	Used at index creation for sort
innodb_ft_enable_stopword	1	Whether using stopwords
innodb_ft_*_stopword_table		Db_name/table_name of the stop word table

InnoDB FTS index : maintenance

Use optimize table

- Updates and deletes feed the « deleted » auxiliary table
- Physically removed by OPTIMIZE
- Run OPTIMIZE TABLE after changes
- Or better:

```
SET innodb_optimize_fulltext_only = 1;
SET innodb_ft_num_word_optimize = 100000;
OPTIMIZE TABLE <table_name>;
OPTIMIZE TABLE <table_name>;
...
```

Demo (1)

Adding a full text index search to WordPress

- Same installation as previously
- WordPress comes with a SQL search, not full text
- Create a full text index on wp_posts

```
CREATE FULLTEXT INDEX in_fts
ON wp_posts(post_title, post_content);
```

Demo (2)

Edit /var/www/html/wordpress/wp-includes/query


```
/*foreach( (array) $q['search_terms'] as $term ) {
    $term = esc_sql( like_escape( $term ) );
    $search .= "{$searchand}(($wpdb->posts.post_title LIKE
    '{$n}{$term}{$n}') OR ($wpdb->posts.post_content LIKE
    '{$n}{$term}{$n}'))";
    $searchand = ' AND ';
  }*/
$search .= "{$searchand}(( MATCH($wpdb->posts.post_title,$wpdb-
  >posts.post_content) against('".$q['s']."') ))";
```

Major Optimizer enhancements

- Index Condition Pushdown (ICP)
- Multi-Range Reads (MRR)
- Batched Key Access (BKA)
- Subqueries
- Order by limit optimization
- Varchar maximum length
- InnoDB extended secondary keys

Optimizer switches (1)

5.6

```
mysql> select @@optimizer_switch\G
@@optimizer_switch:
index_merge=on,
index_merge_union=on,
index_merge_sort_union=on,
index_merge_intersection=on,
engine_condition_pushdown=on,
index_condition_pushdown=on,
mrr=on,
mrr_cost_based=on,
```

ICP

MRR

Optimizer switches (2)

5.6

```
block_nested_loop=on,
batched_key_access=off,
```

BNL, BKA

```
materialization=on,
semijoin=on,loosescan=on,firstmatch=on,
subquery_materialization_cost_based=on,
```

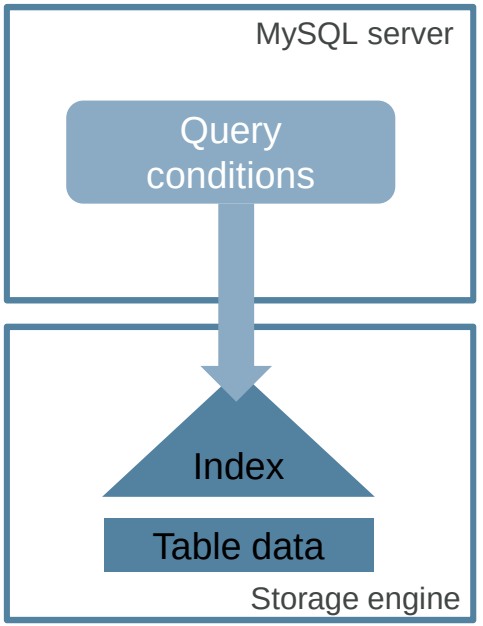
Subqueries

```
use_index_extensions=on
```

Index extension using
InnoDB PK in sec indexes

Index Condition Pushdown (from O. Grovlen)

- Pushes conditions that can be evaluated on the index down to storage engine
  - Works only on indexed columns
- Goal: evaluate conditions without having to access the actual record
  - Reduces number of disk/block accesses
  - Reduces CPU usage
- Both conditions on a single index and conditions on earlier tables in a JOIN can be pushed down



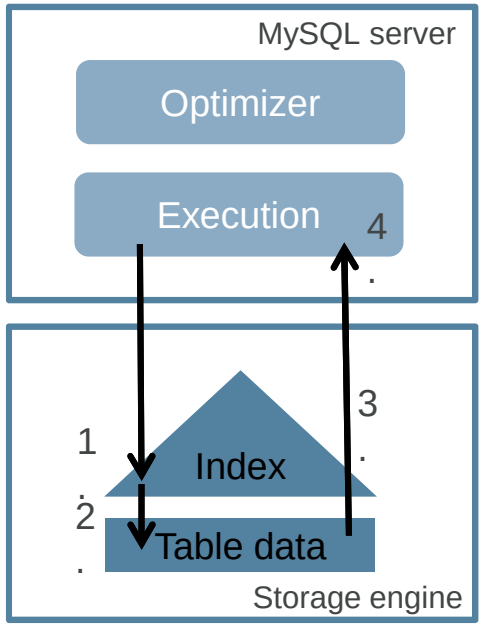
Index Condition Pushdown (from O. Grovlen)

How it works

Without ICP:

Storage Engine:
1. Reads index
2. Reads record
3. Returns record

Server:
4. Evaluates condition



With ICP:

Storage Engine:
1. Reads index and **evaluates pushed index condition**
2. Reads record
3. Returns record

Server:
4. Evaluates **rest of** condition

Index Condition Pushdown Example

```
CREATE TABLE person (
  name VARCHAR(..),
  height INTEGER,
  postalcode INTEGER,
  age INTEGER,
  INDEX (postalcode, age)
);
```

```
SELECT name FROM person
WHERE postalcode BETWEEN 90000 AND 95500 AND age BETWEEN 21 AND 22 AND
height>180;
```

Evaluated by server

Pushed to storage engine
Evaluated on index entries

Index Condition Pushdown Demo (1)

```
drop database if exists test_icp;
create database test_icp;
use test_icp;
CREATE TABLE person (
    name VARCHAR(50),
    height INTEGER,
    postalcode INTEGER,
    age INTEGER,
    INDEX (postalcode , age)
)  ENGINE=InnoDB;
```

Index on (postalcode, age)

```
insert into person(name, height, postalcode, age)
values( floor(rand()*100000),  greatest(floor(rand()*235),100),floor(rand()*95000), floor(rand()*125));
insert into person(name, height, postalcode, age)
values( floor(rand()*100000),  greatest(floor(rand()*235),100), floor(rand()*95000), floor(rand()*125));
replace into person(name, height, postalcode, age)
select floor(rand()*100000),  greatest(floor(rand()*235),100), floor(rand()*95000), floor(rand()*125) from
person p1, person p2, person p4, person p5, person p6, person p7, person p8, person p9, person p10,
person p11, person p12, person p13, person p14, person p15, person p16, person p17, person p18;
```

```
SELECT count(*) FROM person;
```

131074

```
SELECT count(*) FROM person WHERE postalcode BETWEEN 90000 AND 95000;
```

6911

```
SELECT count(*) FROM person WHERE postalcode BETWEEN 90000 AND 95000 AND age BETWEEN 21 AND 22;
```

97

Index Condition Pushdown Demo (2)

Testing ICP on / off

5.6

```
SET SESSION optimizer_switch = 'index_condition_pushdown=on';
FLUSH STATUS;

SELECT name FROM person WHERE
postalcode BETWEEN 90000 AND 95000 AND age BETWEEN 21 AND 22 AND height > 180;

SHOW SESSION STATUS LIKE 'Handler_read%';
EXPLAIN SELECT name FROM person WHERE
postalcode BETWEEN 90000 AND 95000 AND age BETWEEN 21 AND 22 AND height > 180;
```

5.5 like

```
SET SESSION optimizer_switch = 'index_condition_pushdown=off';
FLUSH STATUS;

SELECT name FROM person WHERE
postalcode BETWEEN 90000 AND 95000 AND age BETWEEN 21 AND 22 AND height > 180;

SHOW SESSION STATUS LIKE 'Handler_read%';
EXPLAIN SELECT name FROM person WHERE
postalcode BETWEEN 90000 AND 95000 AND age BETWEEN 21 AND 22 AND height > 180;
```

Index Condition Pushdown Demo (3)

Results for 5.6

```
mysql> SHOW SESSION STATUS LIKE 'Handler_read%';
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| Handler_read_first    | 0     |
| Handler_read_key      | 1     |
| Handler_read_last     | 0     |
| Handler_read_next     | 97    |
| Handler_read_prev     | 0     |
| Handler_read_rnd      | 0     |
| Handler_read_rnd_next | 0     |
+-----------------------+-------+
7 rows in set (0.00 sec)
```

Using *index condition*,
97 rows examined in the index

```
mysql> EXPLAIN SELECT name FROM person WHERE
    -> postalcode BETWEEN 90000 AND 95000 AND age BETWEEN 21 AND 22 AND height > 180;
+----+-------------+--------+-------+---------------+------------+---------+------+-------+-----------------------------------+
| id | select_type | table  | type  | possible_keys | key        | key_len | ref  | rows  | Extra                             |
+----+-------------+--------+-------+---------------+------------+---------+------+-------+-----------------------------------+
|  1 | SIMPLE      | person | range | postalcode    | postalcode | 10      | NULL | 13990 | Using index condition; Using where |
+----+-------------+--------+-------+---------------+------------+---------+------+-------+-----------------------------------+
1 row in set (0.00 sec)
```

Index Condition Pushdown Demo (4)

Results for ICP = off, 5.5 like

```
mysql> SHOW SESSION STATUS LIKE 'Handler_read%';
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| Handler_read_first    | 0     |
| Handler_read_key      | 1     |
| Handler_read_last     | 0     |
| Handler_read_next     | 6911  |
| Handler_read_prev     | 0     |
| Handler_read_rnd      | 0     |
| Handler_read_rnd_next | 0     |
+-----------------------+-------+
7 rows in set (0.00 sec)
```

Using where, 6911 rows examined

```
mysql> EXPLAIN SELECT name FROM person WHERE
    -> postalcode BETWEEN 90000 AND 95000 AND age BETWEEN 21 AND 22 AND height > 180;
+----+-------------+--------+-------+---------------+------------+---------+------+-------+-------------+
| id | select_type | table  | type  | possible_keys | key        | key_len | ref  | rows  | Extra       |
+----+-------------+--------+-------+---------------+------------+---------+------+-------+-------------+
|  1 | SIMPLE      | person | range | postalcode    | postalcode | 10      | NULL | 13990 | Using where |
+----+-------------+--------+-------+---------------+------------+---------+------+-------+-------------+
1 row in set (0.00 sec)
```

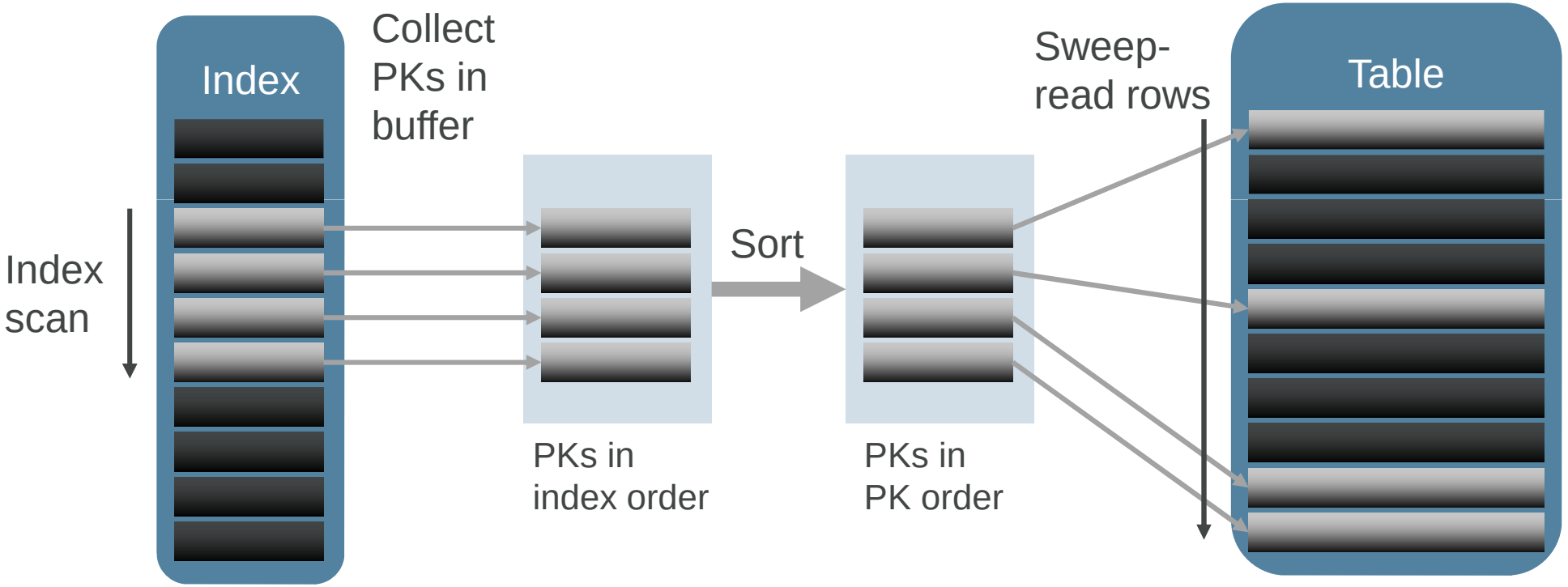
Multi-Range Reads

IO bound load only

- Used for MySQL Cluster as well
- MySQL Server uses DS-MRR (disk sweep MRR)
- Only useful for IO bound loads
  - index ranges
  - convert random key access to sequential access
  - using ***read_rnd_buffer_size***
  - Heuristics : table size larger than ***innodb_buffer_pool_size***

MySQL 5.6: Data Access with DS-MRR

InnoDB Example



Multi-Range Reads

Forcing MRR

- `set session optimizer_switch = 'mrr=on,mrr_cost_based=off,index_condition_pushdown=off';`

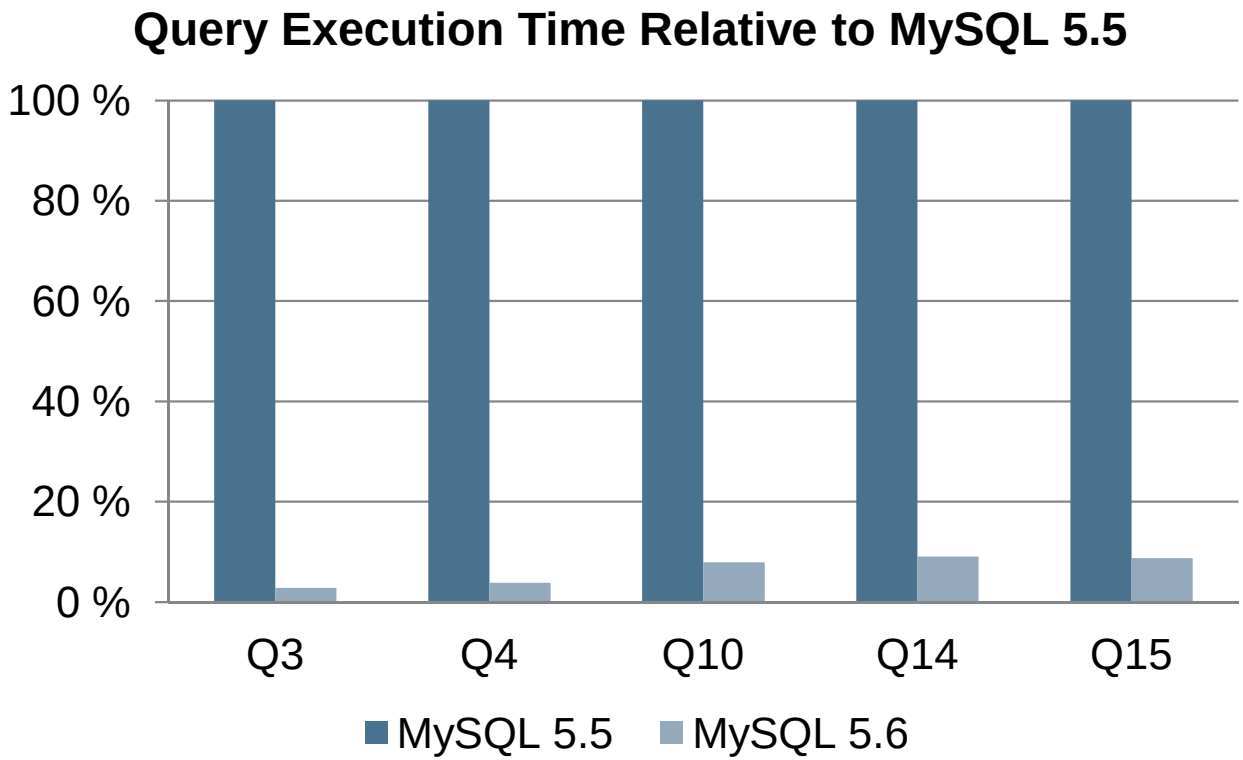
```
set session optimizer_switch = 'mrr=on,mrr_cost_based=off,index_condition_pushdown=off';
explain SELECT * FROM person   WHERE postalcode >= 90000 AND postalcode < 95000   AND age = 36;
+----+-------------+--------+-------+---------------+------------+---------+------+-------+-----------------------+
| id | select_type | table  | type  | possible_keys | key        | key_len | ref  | rows  | Extra                 |
+----+-------------+--------+-------+---------------+------------+---------+------+-------+-----------------------+
|  1 | SIMPLE      | person | range | postalcode    | postalcode | 10      | NULL | 13986 | Using where; Using MRR |
+----+-------------+--------+-------+---------------+------------+---------+------+-------+-----------------------+
```

MySQL 5.5 vs MySQL 5.6: DBT-3 Queries using DS-MRR

DBT-3, Scale 10 (23 GB)

innodb_buffer_pool_size= 1 GB (disk-bound)

read_rnd_buffer_size = 4 MB



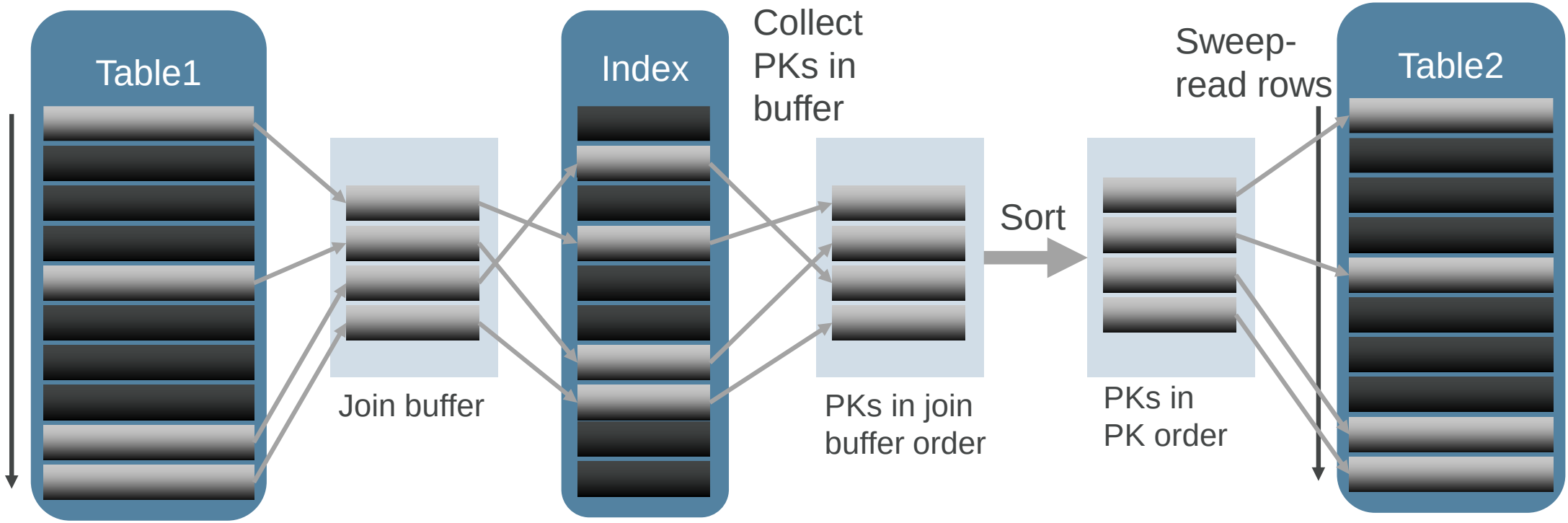
Batched Key Access

IO bound load only

- Basically MRR for joins with an index
- Using the ***join_buffer_size***
- Only useful for IO bound loads
- Sorts the rows in the join_buffer
  - MRR interface to storage engine

MySQL 5.6: Batched Key Access (BKA)

DS-MRR Applied to Join Buffering



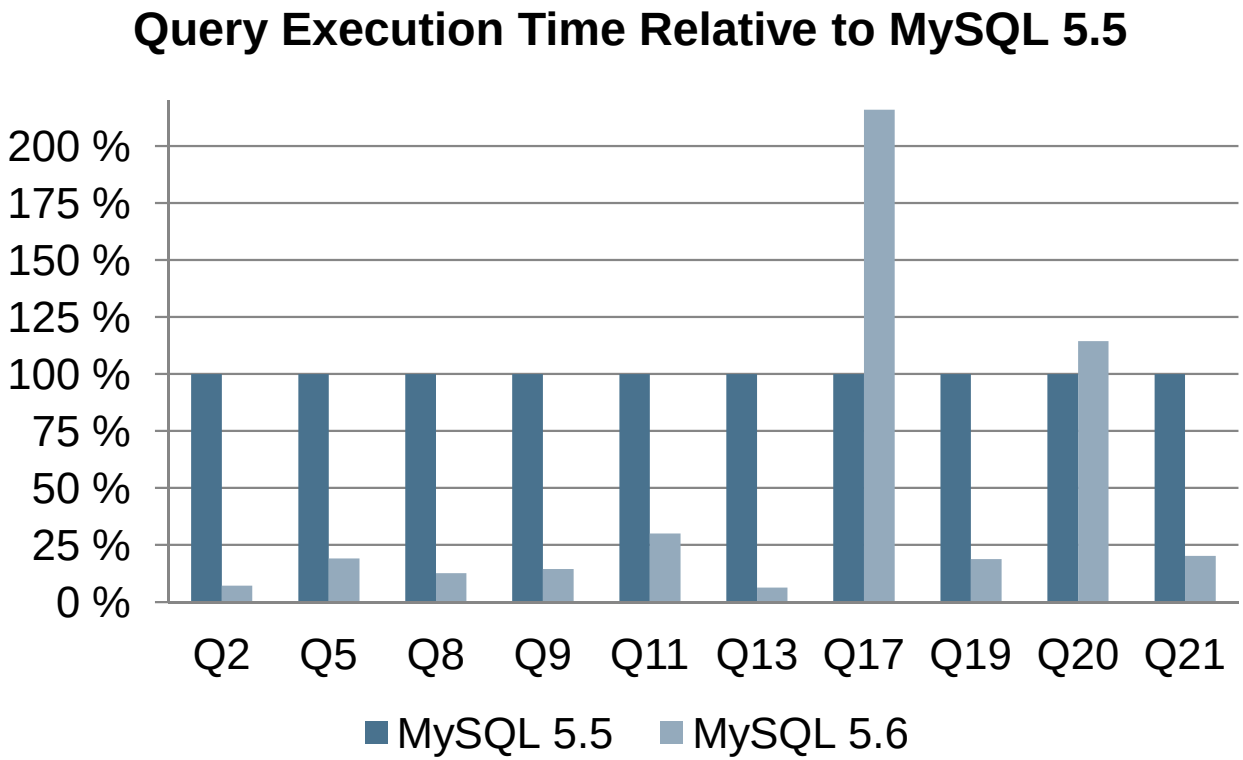
MySQL 5.5 vs MySQL 5.6: Queries using BKA

DBT-3, Scale 10 (23 GB)

innodb_buffer_pool_size= 1 GB (disk-bound)

join_buffer_size = 4 MB

optimizer_switch = 'batched_key_access=on, mrr_cost_based=off'



Batched Key Access

Forcing BKA

- `SET optimizer_switch='mrr=on,mrr_cost_based=off, batched_key_access=on';`
- `SELECT * FROM city c join person p WHERE p.postalcode >= 90000 AND p.postalcode < 95000 and c.postalcode = p.postalcode`
- Extra: Using join buffer (Batched Key Access)

Subqueries

A major improvement !

- IN queries were a big issue in MySQL before 5.6
  - A major cause of query rewrite

```
SELECT * FROM t1 WHERE t1.a IN
(SELECT t2.b FROM t2 WHERE where_condition)
```

- From 5.6, the optimizer can decide to use a :
  - Semi-join (duplicate removal)
  - Materialization
  - EXISTS strategy
- 5.6 optimizes subqueries in the FROM Clause (Derived Tables)

```
SELECT count(*)FROM (SELECT * FROM t) z;
```

Order by limit optimization

5.6 can use an i**n-memory** priority queue

- `SELECT col1, ... FROM t1 ... ORDER BY name LIMIT 10;`
- 2 possibilities in 5.6 :
  - Use filesort (IO on disk, CPU)
  - Use a ***priority queue*** in the ***sort_buffer_size*** that compares the row to the « last »value in the queue.
    - Sort the queue if the row is in the queue.
- The optimizer chooses the best option
- Works from SQL_CALC_FOUND_ROWS

Varchar maximum length

Using variable length MyISAM temporary tables

```
CREATE TABLE t1 (
  id int,
  col1 varchar(10),
  col2 varchar(2048)
);
/* insert 4M rows */
SELECT col1 FROM t1 GROUP BY col2, id LIMIT 1;
```

InnoDB index extension

The optimizer now considers the PK in secondary indexes

```
SET optimizer_switch='use_index_extensions=on'; (default)
CREATE TABLE t1 (
  i1 INT NOT NULL DEFAULT 0,
  i2 INT NOT NULL DEFAULT 0,
  d DATE DEFAULT NULL,
  PRIMARY KEY (i1, i2),
  INDEX k_d (d)   ← contains (d, i1, i2)
) ENGINE = InnoDB;
```

Enhanced explain and tracing

- Explain for update, insert, delete statements
- Explain format = JSON
- Optimizer tracing

EXPLAIN for update, insert, delete

Was a long awaited feature request !

- Before MySQL 5.6, EXPLAIN
  - Only available for SELECT
  - Rewrite was needed
  - `SHOW STATUS LIKE 'Handler%'`
- From 5.6, available for :
  - INSERT
  - UPDATE
  - DELETE
  - REPLACE

Explain for data modifiers

Example

```
EXPLAIN DELETE FROM person WHERE postalcode '10000‘\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: person
         type: range
possible_keys: postalcode
          key: postalcode
      key_len: 5
          ref: const
         rows: 1
        Extra: Using where
```

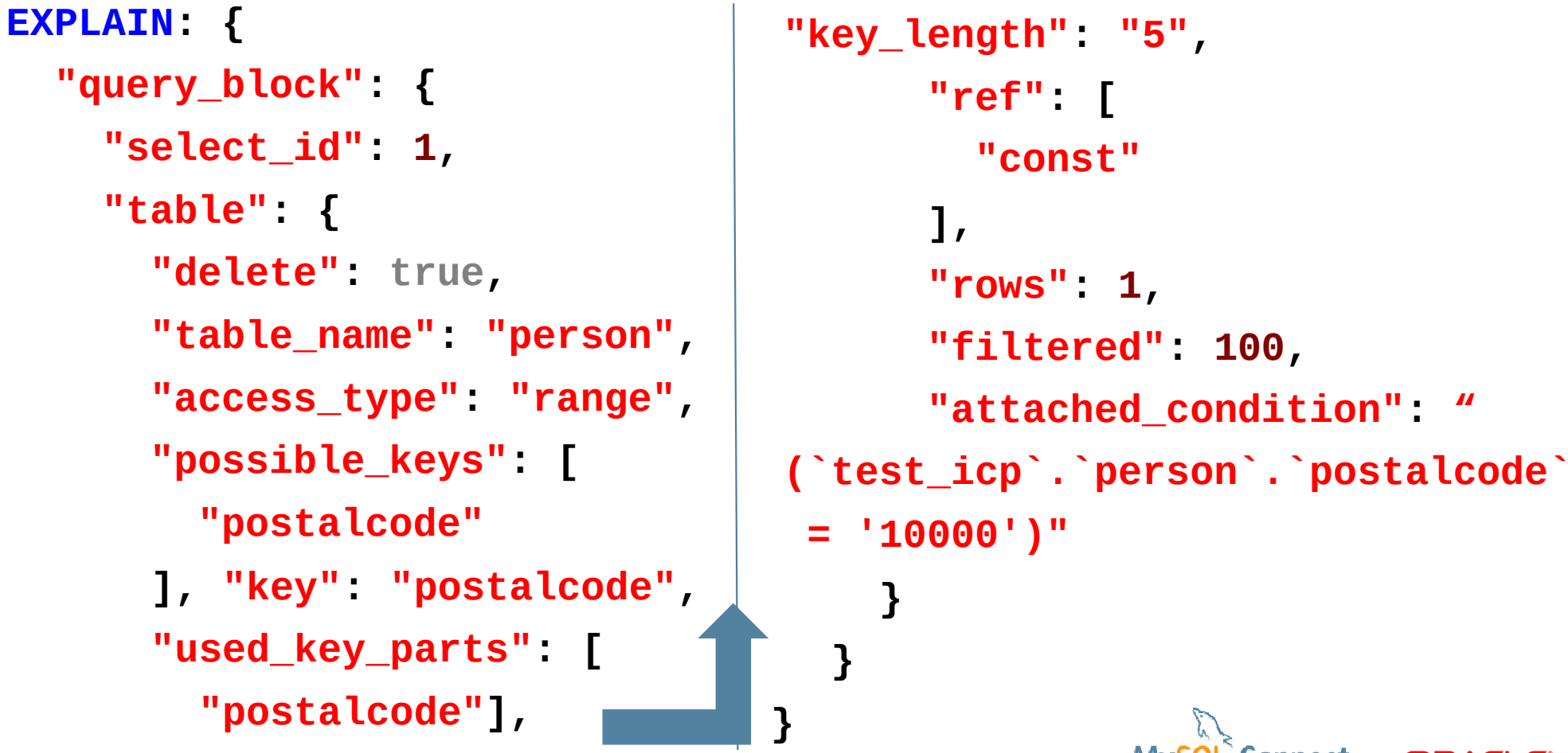
EXPLAIN format = JSON

Provides more information in a developer friendly format

- EXPLAIN has a lot of variations
  - EXPLAIN EXTENDED; SHOW WARNINGS\G
  - EXPLAIN PARTITIONS
- EXPLAIN format tabular
- EXPLAIN format=JSON
  - Has it all
  - In structured format
  - Extensible format
  - Verbose

EXPLAIN format = JSON

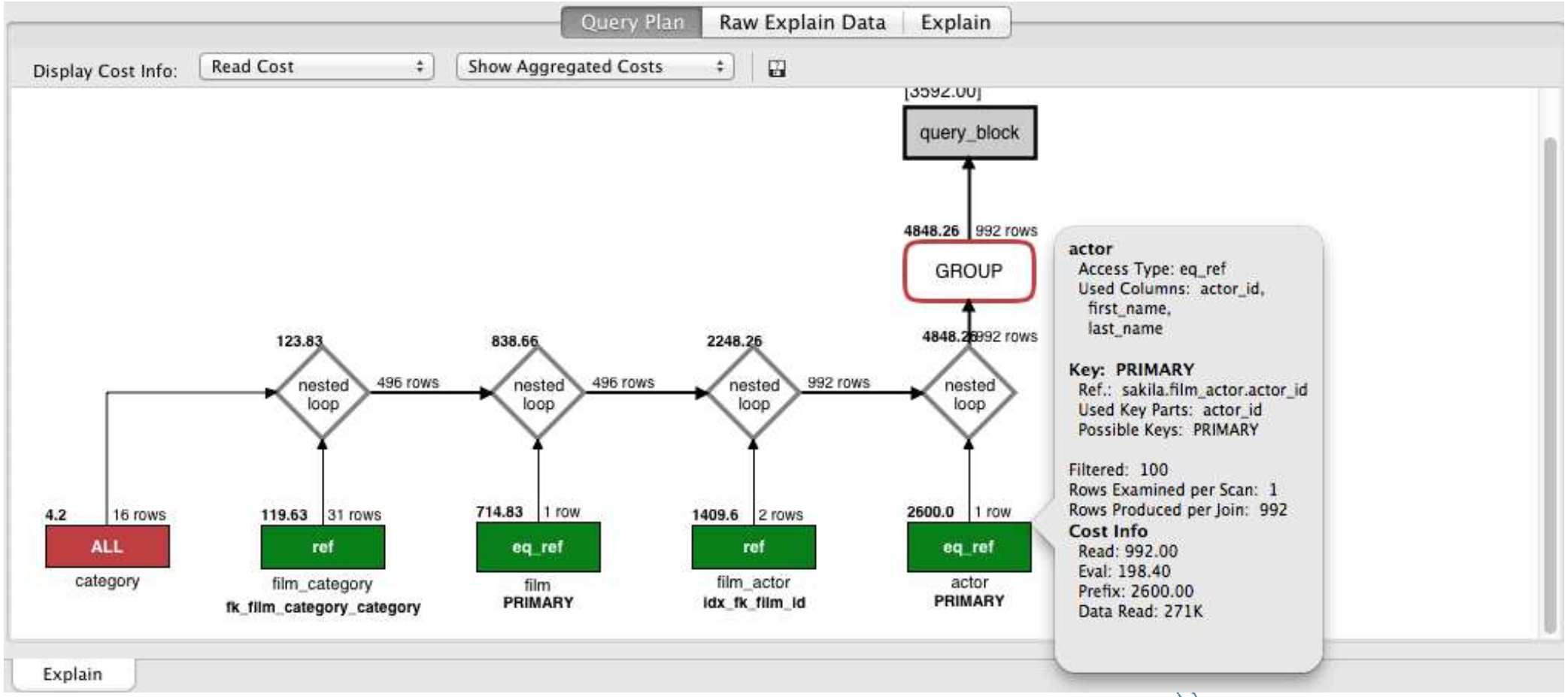
explain format=JSON delete from person where postalcode = '10000'\G



```
EXPLAIN: {
  "query_block": {
    "select_id": 1,
    "table": {
      "delete": true,
      "table_name": "person",
      "access_type": "range",
      "possible_keys": [
        "postalcode"
      ], "key": "postalcode",
      "used_key_parts": [
        "postalcode"],
      "key_length": "5",
          "ref": [
            "const"
          ],
          "rows": 1,
          "filtered": 100,
          "attached_condition": “
(`test_icp`.`person`.`postalcode`
 = '10000')"
        }
    }
}
```

Visual explain in MySQL WorkBench

The JSON format can be displayed by GUI tools



Optimizer traces

New in 5.6, useful for support and troubleshooting

- Typical usage :

```
# Turn tracing on (it's off by default):
SET optimizer_trace="enabled=on,end_marker=on";
SET optimizer_trace_max_mem_size=1000000;
SELECT ...; # your query here
SELECT * FROM INFORMATION_SCHEMA.OPTIMIZER_TRACE;
# possibly more queries...
# When done with tracing, disable it:
SET optimizer_trace="enabled=off";
```

Optimizer traces

What is traced ?

- Contains more information on the optimizer decision
- Execution plans at runtime !
- Select, update, delete, replace
- CALL (stored procedure), SET using selects
- The replication SQL thread is not traceable
- 20 % overhead
- **SELECT TRACE INTO DUMPFILE <filename> FROM INFORMATION_SCHEMA.OPTIMIZER_TRACE;**

Optimizer trace : demo (1)

Creating a stored procedure

```
use test_icp;
drop procedure if exists test_proc;
delimiter //
create procedure test_proc() begin
set @s = "";
select 'select count(*) from person' into @s;
prepare stmt from @s;
execute stmt;
end
//
delimiter ;
```

Optimizer trace : demo (2)

Test script

```
SET OPTIMIZER_TRACE="enabled=on",END_MARKERS_IN_JSON=on;
set optimizer_trace_limit = 100;
set optimizer_trace_offset=-100;
SET optimizer_trace_max_mem_size=1000000;
SELECT * FROM city c join person p WHERE p.postalcode >= 90000 AND
  p.postalcode < 95000 and c.postalcode = p.postalcode;
call test_proc();
SELECT * FROM INFORMATION_SCHEMA.OPTIMIZER_TRACE;
SET optimizer_trace="enabled=off";
```

Optimizer trace : demo (3)

Results

QUERY	TRACE
SET optimizer_trace_max_mem_size=1000000	{ "steps": [ ] /* steps */ }
SELECT * FROM city c join person p WHERE p.p...	{ "steps": [{ "join_preparation": { "select#": 1, "steps": [{ "expanded_query": "/* select#1 */ select `c`.`postalcode` A
call test_proc()	{ "steps": [ ] /* steps */ }
SET @s = ''	{ "steps": [ ] /* steps */ }
select 'select count(*) from person' into @s	{ "steps": [{ "join_preparation": { "select#": 1, "steps": [{ "expanded_query": "/* select#1 */ select 'select count(*) fr
select count(*) from person	{ "steps": [{ "join_preparation": { "select#": 1, "steps": [{ "expanded_query": "/* select#1 */ select count(0) AS `cou
select count(*) from person	{ "steps": [{ "join_preparation": { "select#": 1, "steps": [{ "expanded_query": "/* select#1 */ select count(0) AS `cou

Optimizer trace : demo (4)


```
"considered_execution_plans": [
        {
          "plan_prefix": [
          ] /* plan_prefix */,
          "table": "`person`",
          "best_access_path": {
            "considered_access_paths": [
              {
                "access_type": "scan",
                "rows": 131315,
                "cost": 26680,
                "chosen": true
              }
            ] /* considered_access_paths */
          } /* best_access_path */,
          "cost_for_plan": 26680,
          "rows_for_plan": 131315,
          "chosen": true
        }
```

Persistent statistics

- Why ?
- What's new in 5.6 ?
- Where are the statistics stored ?
- Demo

Persistent statistics : why ?

A major improvement for plan stability

- InnoDB statistics are based on random dives
  - to estimate index **cardinalities**
- Before 5.6, index and table statistics are :
  - calculated at runtime
  - transient
- Bad for execution plans
  - different EXPLAIN
  - different EXPLAIN on the master and slaves

Cardinality

An estimate of the number of unique values in the index.(…)The higher the cardinality, the greater the chance that MySQL uses the index when doing joins.

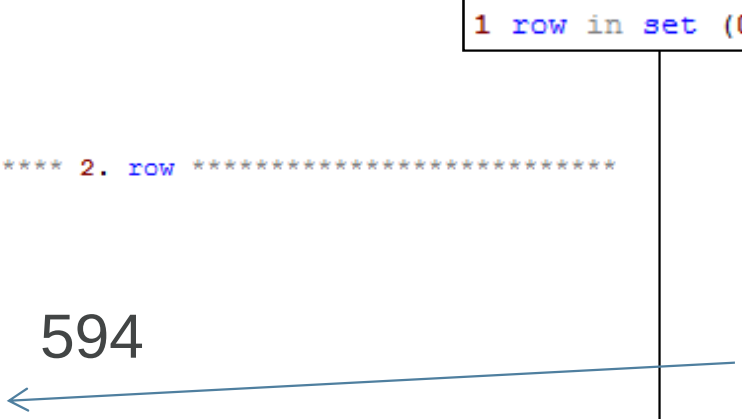
MySQL manual

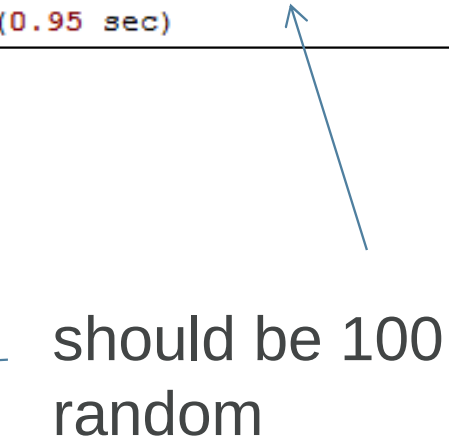
SHOW INDEX Syntax

Cardinality estimate on a big InnoDB table

```
show indexes from t1\G
*************************** 1. row ***************************
        Table: t1
   Non_unique: 0
     Key_name: PRIMARY
 Seq_in_index: 1
  Column_name: id
    Collation: A
  Cardinality: 1249236
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
*************************** 2. row ***************************
        Table: t1
   Non_unique: 1
     Key_name: c1
 Seq_in_index: 1
  Column_name: c1
    Collation: A
  Cardinality: 594
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
```

```
mysql> select count(*), count(distinct c1) from test_persistent.t1;
+----------+--------------------+
| count(*) | count(distinct c1) |
+----------+--------------------+
|  1248578 |                100 |
+----------+--------------------+
1 row in set (0.95 sec)
```





Persistent statistics : example

Big tables with close cardinalities

- 2 identical tables can produce different values due to random dives :

```
create table t2 like t1;
insert into t2 select * from t1 order by id;
analyze table t2;
analyze table t1;
show indexes from t2;
show indexes from t1;
```

Same data

Different cardinalities !

```
*************************** 2. row ***************************
        Table: t1
   Non_unique: 1
     Key_name: c1
 Seq_in_index: 1
  Column_name: c1
    Collation: A
  Cardinality: 594
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
```

594

```
*************************** 2. row ***************************
        Table: t2
   Non_unique: 1
     Key_name: c1
 Seq_in_index: 1
  Column_name: c1
    Collation: A
  Cardinality: 18
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
```

18

What's new in 5.6 ?

Persistent statistics and more

- ***innodb_stats_persistent*** = ON per default
- ***innodb_stats_auto_recalc*** = ON

```
mysql> show variables like 'innodb_stat%';
+---------------------------------------+-------------+
| Variable_name                         | Value       |
+---------------------------------------+-------------+
| innodb_stats_auto_recalc              | ON          |
| innodb_stats_method                   | nulls_equal |
| innodb_stats_on_metadata              | OFF         |
| innodb_stats_persistent               | ON          |
| innodb_stats_persistent_sample_pages  | 20          |
| innodb_stats_sample_pages             | 8           |
| innodb_stats_transient_sample_pages   | 8           |
+---------------------------------------+-------------+
7 rows in set (0.00 sec)
```

Statistics are collected using ANALYZE TABLE or if the rows changed by more than 10%

What's new in 5.6 ?

Persisted statistics and more

- ***innodb_stats_on_metadata*** = OFF per default (dynamic variable)
- ***innodb_stats_method*** = nulls_equal

```
mysql> show variables like 'innodb_stat%';
+---------------------------------------+-------------+
| Variable_name                         | Value       |
+---------------------------------------+-------------+
| innodb_stats_auto_recalc              | ON          |
| innodb_stats_method                   | nulls_equal |
| innodb_stats_on_metadata              | OFF         |
| innodb_stats_persistent               | ON          |
| innodb_stats_persistent_sample_pages  | 20          |
| innodb_stats_sample_pages             | 8           |
| innodb_stats_transient_sample_pages   | 8           |
+---------------------------------------+-------------+
7 rows in set (0.00 sec)
```

innodb_stats_method impacts cardinality calculation when there are NULL values

What's new in 5.6 ?

Persisted statistics and more

- ***innodb_persistent_sample_pages*** = 20
- ***innodb_transient_sample_pages*** = 8
- = ***innodb_sample_pages*** (deprecated)

```
mysql> show variables like 'innodb_stat%';
+---------------------------------------+-------------+
| Variable_name                         | Value       |
+---------------------------------------+-------------+
| innodb_stats_auto_recalc              | ON          |
| innodb_stats_method                   | nulls_equal |
| innodb_stats_on_metadata              | OFF         |
| innodb_stats_persistent               | ON          |
| innodb_stats_persistent_sample_pages  | 20          |
| innodb_stats_sample_pages             | 8           |
| innodb_stats_transient_sample_pages   | 8           |
+---------------------------------------+-------------+
7 rows in set (0.00 sec)
```

Used by the statistics estimation algorithm during ANALYZE TABLE Or auto-recalc

What's new in 5.6 ?

Persisted statistics options at the table level

CREATE / ALTER table option	Possible values	Description
STATS_PERSISTENT	ON, OFF, DEFAULT	See ***Innodb_stats_persistent***
STATS_AUTO_RECALC	ON, OFF, DEFAULT	See ***innodb_stats_auto_recalc***
STATS_SAMPLE_PAGES	20	See ***Innodb_persistent_sample_pages_***

Where are the statistics stored ?

2 new InnoDB tables in the **mysql** database

```
mysql> use mysql;
Database changed
mysql> show tables like '%innodb%';
+---------------------------+
| Tables_in_mysql (%innodb%) |
+---------------------------+
| innodb_index_stats         |
| innodb_table_stats         |
+---------------------------+
2 rows in set (0.00 sec)

show create table innodb_table_stats\G

 CREATE TABLE innodb_table_stats (
  database_name varchar(64) COLLATE utf8_bin NOT NULL,
  table_name varchar(64) COLLATE utf8_bin NOT NULL,
  last_update timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  n_rows bigint(20) unsigned NOT NULL,
  clustered_index_size bigint(20) unsigned NOT NULL,
  sum_of_other_index_sizes bigint(20) unsigned NOT NULL,
  PRIMARY KEY (database_name,table_name)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin STATS_PERSISTENT=0
```

Where are the statistics stored ?

2 new InnoDB tables in the **mysql** database

```
show create table innodb_index_stats\G

 CREATE TABLE innodb_index_stats (
  database_name varchar(64) COLLATE utf8_bin NOT NULL,
  table_name varchar(64) COLLATE utf8_bin NOT NULL,
  index_name varchar(64) COLLATE utf8_bin NOT NULL,
  last_update timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  stat_name varchar(64) COLLATE utf8_bin NOT NULL,
  stat_value bigint(20) unsigned NOT NULL,
  sample_size bigint(20) unsigned DEFAULT NULL,
  stat_description varchar(1024) COLLATE utf8_bin NOT NULL,
  PRIMARY KEY (database_name,table_name,index_name,stat_name)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin STATS_PERSISTENT=0
```

These tables are replicated by default : it solves the problematic master / slave execution plan differences before 5.6

Managing tablespaces and partitions

- Transportable table spaces
- Managing partitions
- Tablespace location
- Separate undo logs
- Demo

Transportable tablespaces

How is an InnoDB table stored ?

- 3 parts :
  - The FRM file
  - The IBD file containing the data
    - several IBD if partitioned
  - An entry in the internal InnoDB data dictionary
    - Located in the main InnoDB tablespace (ibdata1)

Transportable tablespaces

Copying table spaces?

- Pre-requisite : ***innodb_file_per_table*** = 1, (.ibd file).
- Faster copy than mysqldump
- Data recovery
- Backup (MEB can backup one table)
- Building a test environment

Transportable tablespaces

Copying datafiles (ibd) from a server to another

- Before 5.6,
  - It was possible to restore a data file from the same MySQL server
  - Harder if the InnoDB dictionary part was missing (missing table ID)
- From 5.6, it is possible provided that :
  - You are using the same MySQL series : 5.6 => 5.6
    - Preferably **SAME** version
  - Mixing series can cause serious crashes :

    16292419 - CRASH WHEN IMPORTING A 5.5 TABLESPACE TO 5.6
  - It is only safe with additional metadata available at export (.cfg file)

Transportable tablespaces

How it works

source

FLUSH TABLES t FOR EXPORT;

t.frm, t.ibd, t.cfg

UNLOCK TABLES;

destination

CREATE TABLE
t(c1 INT) engine=InnoDB;
ALTER TABLE t DISCARD
TABLESPACE;

Database
directory

ALTER TABLE t IMPORT
TABLESPACE

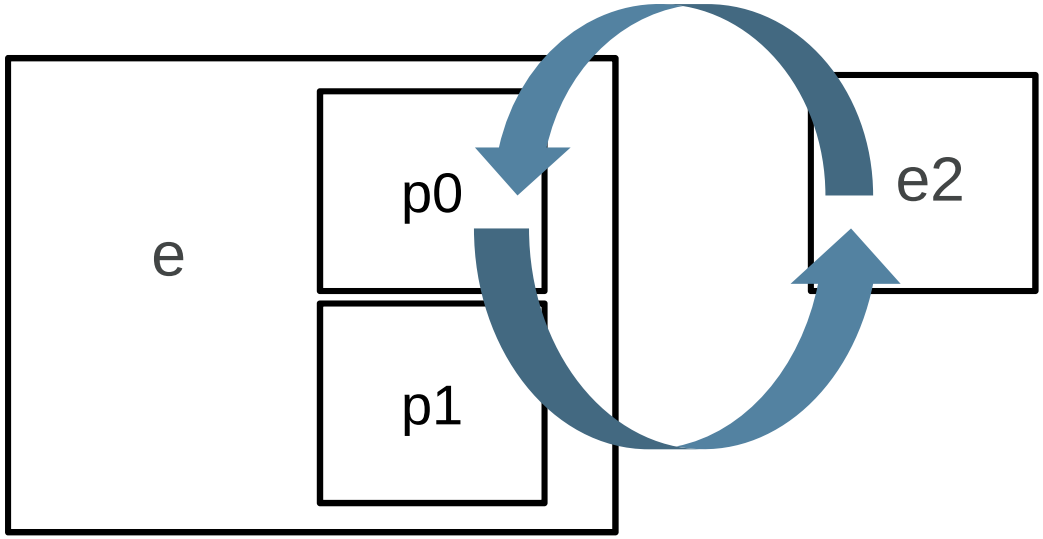
SELECT * from t

Managing partitions

Selecting and exchanging partitions

- `SELECT * FROM employees PARTITION (p0, p2) WHERE lname LIKE 'S%';`

- `ALTER TABLE e EXCHANGE PARTITION p0 WITH TABLE e2;`



e2 data is checked against the p0 partition definition

Tablespace location

More flexibility

- Before 5.6, one datadir for all ibd files.
- Symlinks
  - only solution
  - problematic on ALTER TABLE
  - not supported for InnoDB
- From 5.6 :

```
CREATE TABLE external
(x int UNSIGNED NOT NULL PRIMARY KEY) DATA DIRECTORY
 = '/volumes/external1/data';
```

Separate undo logs

Smaller ibdata1 !

- Undo logs store uncommitted data during a transaction
- Their size is not limited
- Stored in the ibdata1
- 2 variables :
  - **innodb_undo_tablespaces** : the number of undo logs [0 – 126]
  - **Innodb_undo_directory :** undo log directory (better on fast storage)
- Cannot shrink for now
- Cannot be dropped

Online Schema changes (Online DDL)

- Introduction
- Demo

Online DDL : introduction

A most wanted 5.6 feature !

- DDL = **D**ata **D**efinition **L**anguage, mostly ALTER TABLE
- Before 5.6, ALTER TABLE were blocking (exclusive lock)
- Online DDL is crucial for availability, developer agility
  - scripted solutions : trigger based before 5.6
    - pt-online-schema-change
    - Online Schema Change @ Facebook
- MySQL 5.6 introduces Online DDL
  - for most ALTER
  - not all

Online DDL : ALTER options

2 main parameters were added to ALTER TABLE

- ALGORITHM [=] {DEFAULT|INPLACE|COPY}
  - DEFAULT = defined by the operation
  - either : INPLACE or COPY
- LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
  - DEFAULT = defined by the operation
  - NONE : concurrent DML allowed
  - SHARE : concurrent queries
  - EXCLUSIVE : all statements are waiting

Online DDL : definitions

Some definitions are needed before we go on

- DML = **D**ata **M**anipulation **L**anguage
  - Insert
  - Update
  - Delete
  - Replace
- Select are not part of the DML (standard) !
- Concurrent queries = select

Type of Online Operations

Credits : Calvin Sun

- Metadata only
  - MySQL Server metadata, such as alter column default
  - MySQL Server metadata & InnoDB metadata, such as add/drop foreign key
- Metadata plus w/o rebuilding the table, such as add/drop index
- Metadata plus rebuilding the table, such as add primary index, add column.

Credits : Calvin Sun

How Does It Work - Online Add Index

CREATE INDEX *index_name* ON *table name (c1)*

	Concurrent User	Source (table)	(cluster) Index	Metadata Lock
Pre-Prepare Phase	Concurrent Select, Delete, Insert, Update	Check whether the online DDL is supported		Upgradable Shared Metadata Lock
Prepare Phase	No concurrent DML allowed	Create temp table for new index (if primary)	Create log files; Logging starts	Exclusive Metadata Lock
Build Phase	Concurrent Select, Delete, Insert, Update	Scan clustered index; Extract index entries; Sort / merge index build	DML Logging; Apply logs at the end of create index	Upgradable Shared Metadata Lock
Final Phase	No concurrent DML allowed	Drop old table (if primary)	Update system tables (metadata)	Exclusive Metadata Lock

Credits : Calvin Sun

Considerations for Online Operations (1)

- Plan disk space requirements for online operations
  - Temp tables (if need rebuild)
  - DML log files
  - Redo/undo logs
  - Parameters
    a) innodb_sort_buffer_size
    b) innodb_online_alter_log_max_size

Credits : Calvin Sun

Considerations for Online Operations (2)

- Performance considerations
  - Online operations may be slower than offline operations
  - Understand the impact on DML operations
- Other considerations
  - Foreign keys

Example 1: Add / Drop Index

Credits : Calvin Sun

```
mysql: set old_alter_table=0;
Query OK, 0 rows affected (0.00 sec)

mysql: create index i_dtyp_big on big_table (data_type);
Query OK, 0 rows affected (37.93 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql: drop in
Query OK, 0 ro
Records: 0  Du
```

```
mysql: set old_alter_table=1;
Query OK, 0 rows affected (0.00 sec)

mysql: create index i_dtyp_big on big_table (data_type);
Query OK, 1731584 rows affected (4 min 59.33 sec)
Records: 1731584  Duplicates: 0  Warnings: 0

mysql: drop index i_dtyp_big on big_table;
Query OK, 1731584 rows affected (3 min 55.90 sec)
Records: 1731584  Duplicates: 0  Warnings: 0
```

Credits : Calvin Sun

Example 2: Rename Column

```
mysql: set old_alter_table=0;
Query OK, 0 rows affected (0.00 sec)

mysql: alter table big_table change `flags` `new_flags`
    ->   varchar(3) character set utf8 not null;
Query OK, 0 rows affected (0.08 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql: set old_alter_table=1;
Query OK, 0 rows affected (0.00 sec)

mysql: alter table big_table change `new_flags` `flags`
    ->   varchar(3) character set utf8 not null;
Query OK, 1731584 rows affected (3 min 31.78 sec)
Records: 1731584  Duplicates: 0  Warnings: 0
```

Performance_schema in practice

- ps_helper
- how to use ?
- Demo

ps_helper

Getting started with the performance_schema

- Developed by **Mark Leith**, a MySQL expert @ Oracle
  - worked at MySQL Support
  - Senior Software Manager in the MEM team
- A collection of views, stored procedure
- Installed in a separate schema (ps_helper)
- The views are self explaining
- Related talk :
  - **Performance Schema and ps_helper [CON4077]**

Installing ps_helper

ps_helper is a fantastic troubleshooting tool

- https://github.com/MarkLeith/dbahelper/archive/master.zip
- unzip dbahelper-master.zip
- mysql -uroot -p --socket=/tmp/mysql.sock < ps_helper_56.sql
- update performance_schema.setup_instruments set enabled = 'YES';
- update performance_schema.setup_consumers set enabled = 'YES';

How to use ?

You can use it to collect data on your load.

- The performance_schema has a small overhead
  - This overhead is getting optimized
- Just use the ps_helper views !
- You can collect data like this :
  1. Call ps_helper.truncate_all(1) or server restart
  2. Enable the required instruments and consumers
  3. Run the load
  4. Dump the ps_helper views to a file

Demo (1)

Troubleshooting WordPress performance

- WordPress uses a lot of queries
- The code is very difficult to understand
- Plugins can run expensive queries
- The search is very inefficient on large databases
- Has scalability issues with a lot of posts

Analysing ps_helper results : SQL

```
mysql> select * from statement_analysis                                  ;
+-------------------------------------------------------------------+-----------+------------+-----------+------------+---------------+
| query                                                             | full_scan | exec_count | err_count | warn_count | total_latency |
+-------------------------------------------------------------------+-----------+------------+-----------+------------+---------------+
| SELECT `t` . * , `tt` . * , `t ... (?) ORDER BY `t` . `name` ASC  | *         |       9999 |         0 |          0 | 32.91 s       |
| SELECT `option_value` FROM `wp ... ERE `option_name` = ? LIMIT ?  |           |     130170 |         0 |          0 | 21.67 s       |
| SELECT `t` . * , `tt` . * FROM ... > ? ORDER BY `t` . `name` ASC  |           |      10000 |         0 |          0 | 16.45 s       |
| SELECT * FROM `wp_comments` JO ... omment_date_gmt` DESC LIMIT ?  |           |      10000 |         0 |          0 | 10.68 s       |
| SELECT `user_id` , `meta_key`  ... rmeta` WHERE `user_id` IN (?)  | *         |      10000 |         0 |          0 | 8.47 s        |
| SELECT `option_name` , `option ... options` WHERE `autoload` = ?  | *         |      10836 |         0 |          0 | 4.96 s        |
```

```
+--------------+-----------+---------------+--------------+------------+-----------------+--------------+-------------------+
| lock_latency | rows_sent | rows_sent_avg | rows_scanned | tmp_tables | tmp_disk_tables | rows_sorted  | sort_merge_passes |
+--------------+-----------+---------------+--------------+------------+-----------------+--------------+-------------------+
| 2.85 s       |     10000 |             1 |        50000 |      10000 |           10000 |        10000 |                 0 |
| 5.97 s       |         4 |             0 |            4 |          0 |               0 |            0 |                 0 |
| 766.78 ms    |     10000 |             1 |        40000 |      10000 |           10000 |        10000 |                 0 |
| 3.74 s       |     10000 |             1 |        20000 |          0 |               0 |            0 |                 0 |
| 3.95 s       |    160000 |            16 |       160000 |          0 |               0 |            0 |                 0 |
| 904.78 ms    |   1191950 |           110 |      1256966 |          0 |               0 |            0 |                 0 |
```


Analysing ps_helper results : noSQL

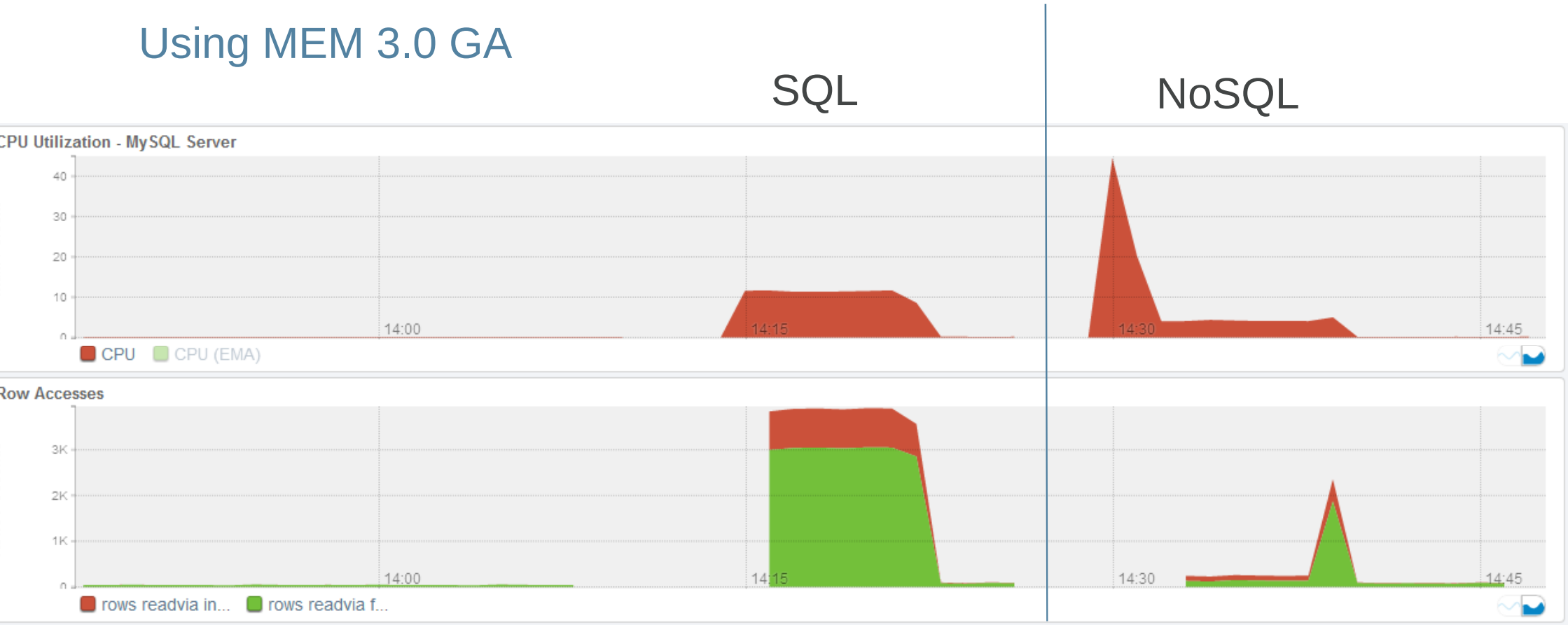
```
mysql> select * from statement_analysis                                        ;
```

query	full_scan	exec_count	err_count	warn_count	total_latency
SELECT `option_name` , `option ... options` WHERE `autoload` = ?	*	83925	0	0	33.44 s
SELECT `t` . * , `tt` . * FROM ... > ? ORDER BY `t` . `name` ASC		9998	0	0	14.53 s
SELECT `t` . * , `tt` . * FROM ... (?) ORDER BY `t` . `name` ASC		2042	0	0	3.97 s
SELECT SQL_CALC_FOUND_ROWS `wp ... `post_date` DESC LIMIT ?, ...		9998	0	0	2.36 s
SELECT `t` . * , `tt` . * , `t ... (?) ORDER BY `t` . `name` ASC	*	657	0	0	1.78 s

avg_latency	lock_latency	rows_sent	rows_sent_avg	rows_scanned	tmp_tables	tmp_disk_tables	rows_sorted
398.39 us	3.88 s	9232080	110	9735648	0	0	0
1.45 ms	1.68 s	9999	1	39996	9999	9999	10000
1.94 ms	231.54 ms	694	0	3470	2042	2042	694
236.35 us	993.80 ms	9997	1	9997	0	0	0
2.71 ms	90.30 ms	657	1	3285	657	657	657
160.00 us	293.82 ms	4663	1	4663	0	0	0

Demo

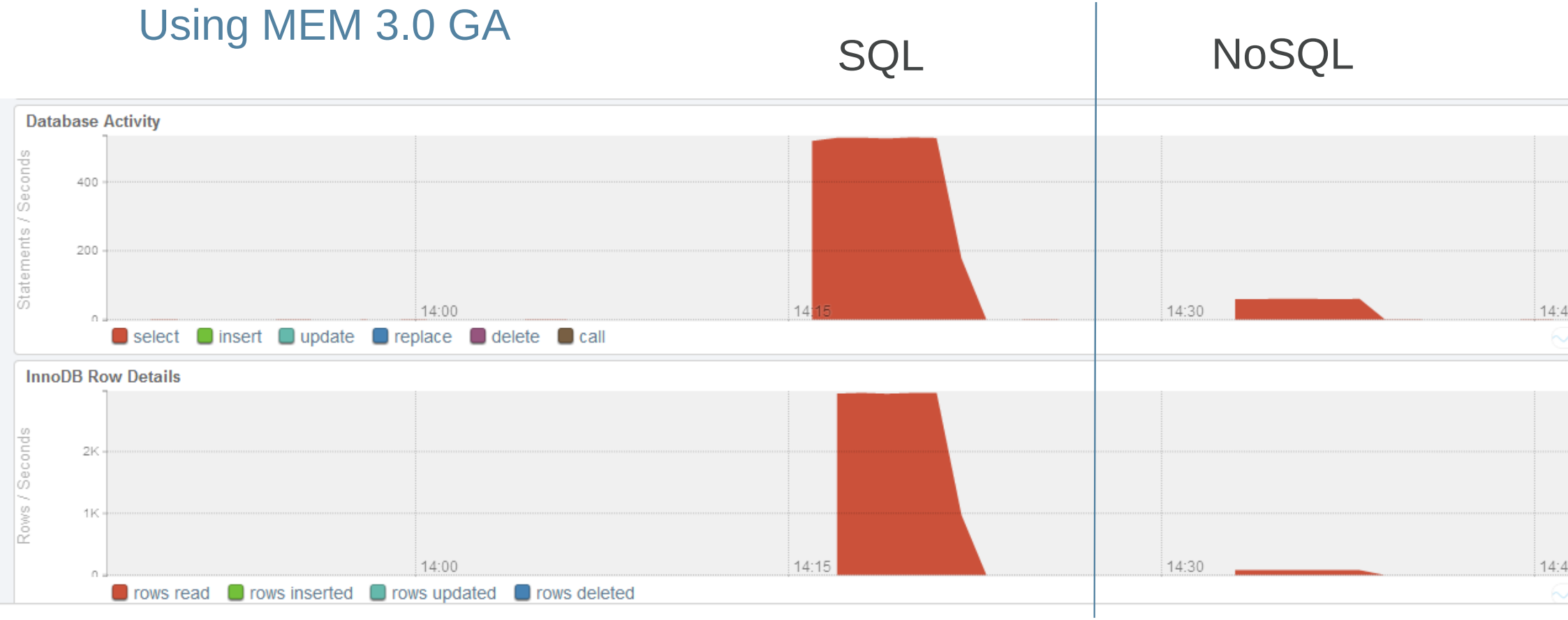
Using MEM 3.0 GA



MySQL Connect ORACLE

Demo

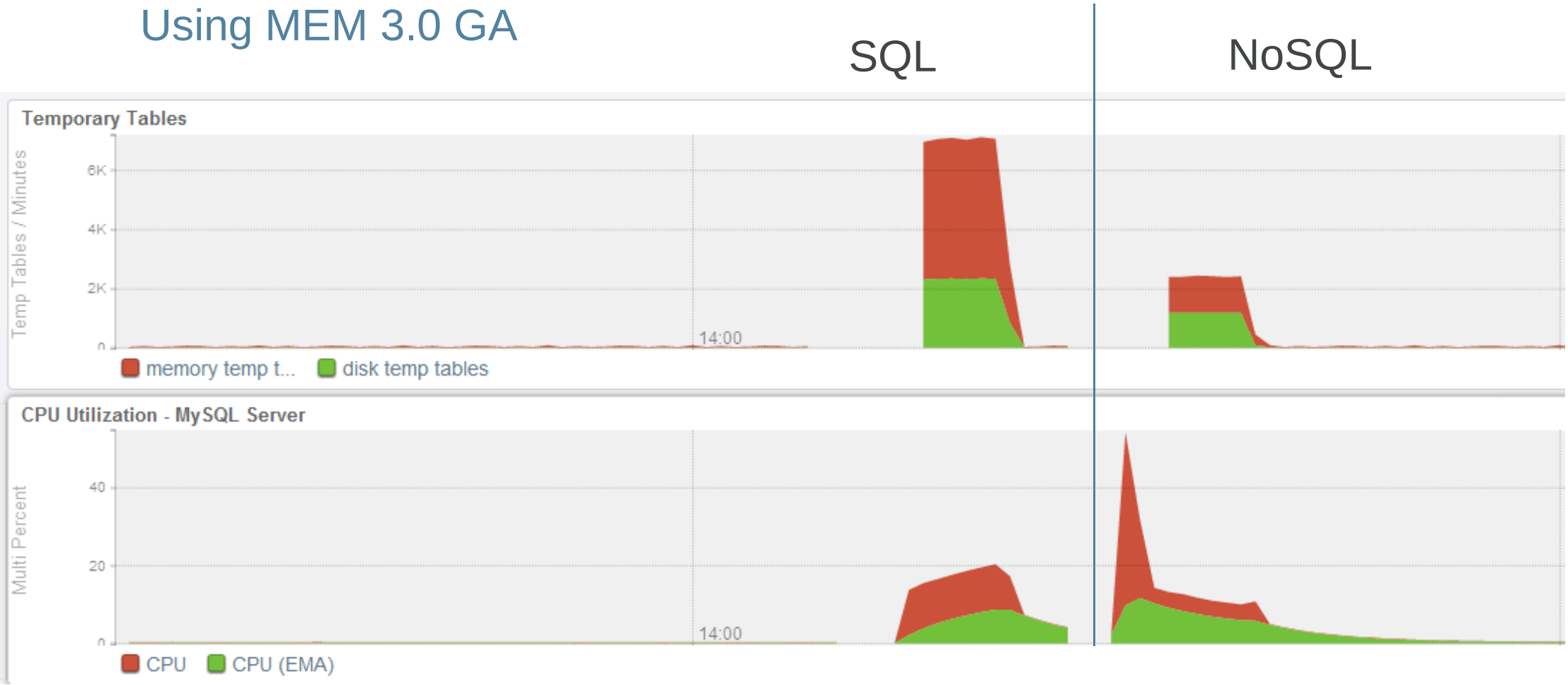
Using MEM 3.0 GA



MySQL Connect ORACLE

Demo

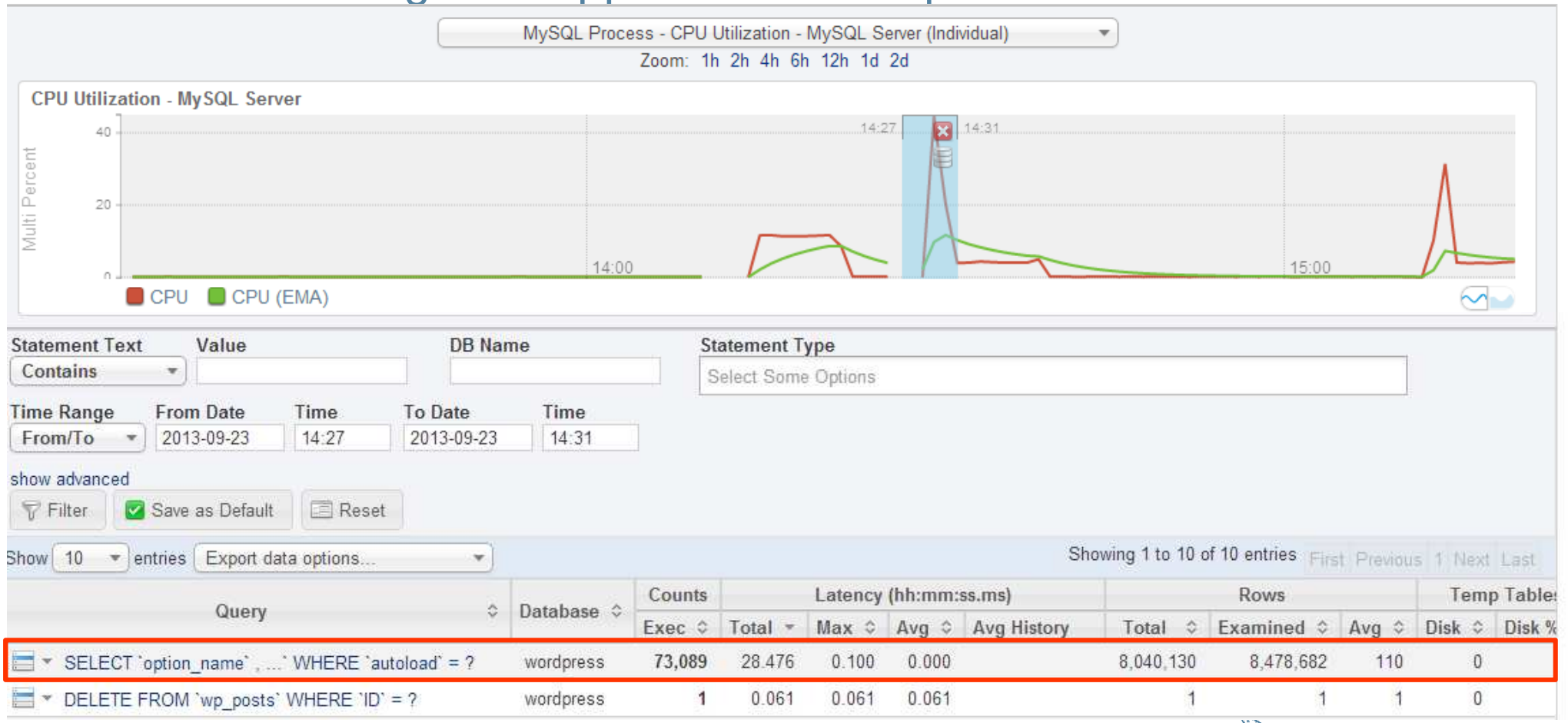
Using MEM 3.0 GA



MySQL Connect ORACLE

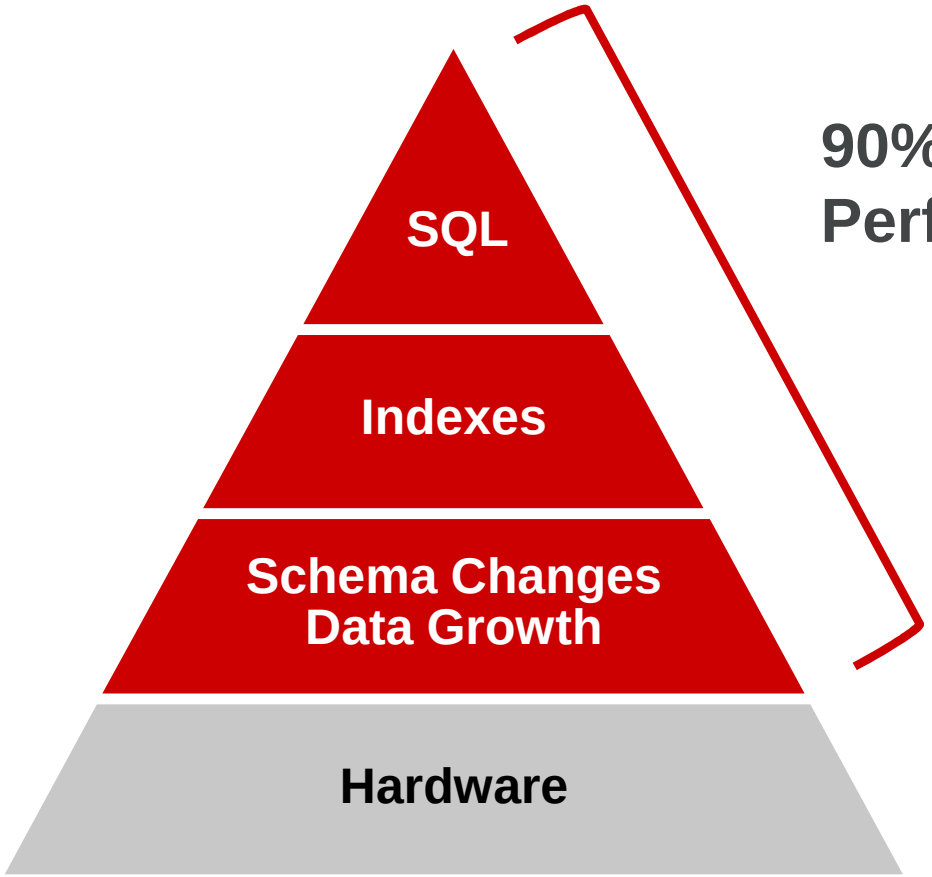
Demo : query analysis

Troubleshooting : the application has a problem !



MySQL Connect ORACLE

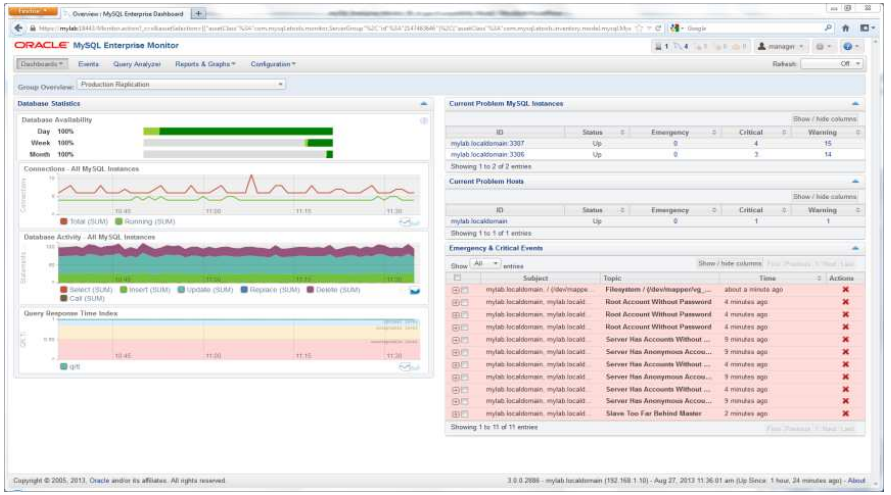
Source of Database Performance Problems



90% of Performance Problems

MySQL Enterprise Monitor 3.0 is GA !

- **SLA monitoring**
- **Real-time performance monitoring**
- **Alerts & notifications**
- **MySQL best practice advisors**



"The MySQL Enterprise Monitor is an absolute must for any DBA who takes his work seriously."

- Adrian Baumann, System Specialist
Federal Office of Information Technology & Telecommunications

MySQL Connect ORACLE

Questions & Answers


MySQL™ Connect