Maximum Velocity MySQL

Jay Pipes
Community Relations Manager, North America
MySQL Inc.
A Quick Survey

- So, how many are using...
  - 3.23? 4.0? 4.1? 5.0? 5.1?
  - MyISAM? InnoDB? Other?
  - Replication? Cluster? Partitioning?
  - Enterprise? Community?
  - PostgreSQL? Oracle? SQL Server? Other?
Get Your Learn On

- Profiling, benchmarking, EXPLAIN command
- Schema and indexing guidelines
- Black-belt SQL coding
- Tuning server settings
Benchmarking, Profiling, and EXPLAIN Command
Benchmarking Concepts

- Allows you to track changes in your application and server environment over time
- Isolate to a single variable
- Record everything
  - Configuration files, OS/Hardware changes, SQL changes
- Shut off unnecessary programs and network
- Disable query cache in MySQL
Your Benchmarking Toolbox

- **ApacheBench (ab)**
  - Excellent for simple web application benchmarks

- **Sysbench, mysqlslap (5.1+), supersmack**
  - MySQL-specific benchmarking tools

- **Frameworks/harnesses reduce repetitive work**
  - MyBench
  - JMeter/Ant
  - Custom scripting (most common)
Example simple benchmark script
benchmark-no-cache.sh

#!/bin/sh
# Restart Apache to ensure cleared buffers
sudo apache2ctl restart

# Restart MySQL
sudo /etc/init.d/mysql restart

# Kill any cached files
sudo rm -rf /var/www/apache2-default/benchmark/cache/*

# Warm up Apache with a simple page
ab -c 300 -n 2000 http://localhost/apache2-default/php-info.php >& /dev/null

echo "NO CACHE BENCHMARK RUN:" > no-cache-benchmark.results

# Reset the query cache and
# flush tables and status counters
mysql --skip-column-names --user=root < setup-benchmark.sql >> no-cache-benchmark.results

# Run the benchmark on the warmed-up server
ab -n 2000 -c 300 \nhttp://localhost/apache2-default/benchmark/test-no-cache.php >> no-cache-benchmark.results

# Run the post-benchmark status script
mysql --skip-column-names --user=root < post-benchmark.sql >> no-cache-benchmark.results
Example simple benchmark result
benchmark-no-cache.result

**SCALE5X**

<table>
<thead>
<tr>
<th>Connection Times (ms)</th>
<th>min</th>
<th>mean[+/-sd]</th>
<th>median</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect:</td>
<td>0</td>
<td>157 669.5</td>
<td>0</td>
<td>3004</td>
</tr>
<tr>
<td>Processing:</td>
<td>113</td>
<td>434 316.3</td>
<td>436</td>
<td>4560</td>
</tr>
<tr>
<td>Waiting:</td>
<td>112</td>
<td>434 316.3</td>
<td>436</td>
<td>4560</td>
</tr>
<tr>
<td>Total:</td>
<td>113</td>
<td>592 730.9</td>
<td>439</td>
<td>5916</td>
</tr>
</tbody>
</table>

Percentage of the requests served within a certain time (ms)
- 50%   439
- 66%   447
- 75%   457
- 80%   466
- 90%   502
- 95%   3193
- 98%   3382
- 99%   3439
- 100%  5916 (longest request)
Profiling Concepts

• Diagnose a running system for bottlenecks
  – I/O
  – Memory
  – CPU
  – Operating System (e.g. file descriptor usage)
  – Network

• Look for the big bottlenecks; don't waste time over-optimizing for microseconds
Slow Query Log

• Enable in the `my.cnf` configuration file:

  ```
  log_slow_queries=/var/lib/mysql/slow-queries.log  # location of log file
  long_query_time=2  # number of seconds for MySQL to consider it slow
  log_long_format  # log any query not using an index (full table scans)
  ```

• 5.1+ does not require a server restart
• Use `mysqldumpslow` program to parse
The EXPLAIN Command

• Simply append EXPLAIN before any SELECT statement
• Returns the execution plan chosen
• Each row in output is a set of information
  – A real schema table
  – A “virtual” table (a subquery in the FROM clause)
  – A subquery in the SELECT or WHERE clause
  – A UNIONed resultset
Example EXPLAIN output

```
mysql> EXPLAIN SELECT f.film_id, f.title, c.name
    > FROM film f INNER JOIN film_category fc
    > ON f.film_id=fc.film_id INNER JOIN category c
    > ON fc.category_id=c.category_id WHERE f.title LIKE 'T%' \G
```

---

**1. row**  
*select_type:* SIMPLE  
*table:* c  
*type:* ALL  
*possible_keys:* PRIMARY  
*key:* NULL  
*key_len:* NULL  
*ref:* NULL  
*rows:* 16

Extra:

---

**2. row**  
*select_type:* SIMPLE  
*table:* fc  
*type:* ref  
*possible_keys:* PRIMARY, fk_film_category_category  
*key:* fk_film_category_category  
*key_len:* 1  
*ref:* sakila.c.category_id  
*rows:* 1

Extra: Using index

---

**3. row**  
*select_type:* SIMPLE  
*table:* f  
*type:* eq_ref  
*possible_keys:* PRIMARY, idx_title  
*key:* PRIMARY  
*key_len:* 2  
*ref:* sakila.fc.film_id  
*rows:* 1  
Extra: Using where

---

An estimate of rows in this set  
The "access strategy" chosen  
The available indexes, and the one(s) chosen  
A covering index is used
**EXPLAIN Tips**

- There is a **huge** difference between “index” in the type column and “Using index” in the Extra column
  - In the type column, it means a full index scan
  - In the Extra column, it means a covering index
- 5.0+ look for the index_merge optimization
  - Prior to 5.0, only one index can be used per table
    - Would have to use a UNION to achieve same results
  - 5.0+ if multiple indexes can be used, can use them
mysql> EXPLAIN SELECT * FROM rental  
    -> WHERE rental_id IN (10,11,12)  
    -> OR rental_date = '2006-02-01'  
*************************** 1. row ***************************  
id: 1
select_type: SIMPLE
table: rental
type: index_merge
possible_keys: PRIMARY,rental_date
    key: rental_date,PRIMARY
key_len: 8,4
    ref: NULL
rows: 4
Extra: Using sort_union(rental_date,PRIMARY);  
Using where
1 row in set (0.04 sec)
Schema and Index Strategies
Poor schema is a great way to shoot yourself in the foot

Use smallest data types possible (esp. InnoDB)
- Do you really need that BIGINT?

Smaller the field structure in table or index row, the more records can fit into a single page (so faster accesses!)

Normalize first, then de-normalize only in extreme cases
Journey to the Center of the Database

Ahh, normalization...

CREATE TABLE Users (  
  user_id INT NOT NULL AUTO_INCREMENT  
  , email VARCHAR(80) NOT NULL  
  , display_name VARCHAR(50) NOT NULL  
  , password CHAR(41) NOT NULL  
  , first_name VARCHAR(25) NOT NULL  
  , last_name VARCHAR(25) NOT NULL  
  , address VARCHAR(80) NOT NULL  
  , city VARCHAR(30) NOT NULL  
  , province CHAR(2) NOT NULL  
  , postcode CHAR(7) NOT NULL  
  , interests TEXT NULL  
  , bio TEXT NULL  
  , signature TEXT NULL  
  , skills TEXT NULL  
  , company TEXT NULL  
  , PRIMARY KEY (user_id)  
  , UNIQUE INDEX (email)  
) ENGINE=InnoDB;
When Horizontal Partitioning Makes Sense

- "Extra" columns are mostly NULL
- "Extra" columns are infrequently accessed
- When space in buffer pool is at a premium
  - Splitting the table allows main records to consume the buffer pages without the extra data taking up space in memory
  - Many more "main" records can fit into a single 16K InnoDB data page
- To use FULLTEXT on your text columns
CREATE TABLE Products (  
p_product_id INT NOT NULL AUTO_INCREMENT  
, name VARCHAR(80) NOT NULL  
, unit_cost DECIMAL(7,2) NOT NULL  
, description TEXT NULL  
, image_path TEXT NULL  
, num_views INT UNSIGNED NOT NULL  
, num_in_stock INT UNSIGNED NOT NULL  
, num_on_order INT UNSIGNED NOT NULL  
, PRIMARY KEY (product_id)  
, INDEX (name(20))  
) ENGINE=InnoDB; // Or MyISAM

CREATE TABLE ProductCounts (  
product_id INT NOT NULL  
, num_views INT UNSIGNED NOT NULL  
, num_in_stock INT UNSIGNED NOT NULL  
, num_on_order INT UNSIGNED NOT NULL  
, PRIMARY KEY (product_id)  
) ENGINE=InnoDB; // Or MyISAM

CREATE TABLE ProductCountSummary (  
total_products INT UNSIGNED NOT NULL  
) ENGINE=MEMORY;

// Getting a simple COUNT of products  
// easy on MyISAM, terrible on InnoDB  
SELECT COUNT(*)  
FROM Products;
When Counter Tables Make Sense

- Mixing static attributes with frequently **updated** fields in a single table?
  - Thrashing occurs with query cache. Each time an update occurs **on any record in the table**, all queries referencing the table are invalidated in the Query Cache.

- Doing COUNT(*) with no WHERE on an indexed field on an **InnoDB** table?
  - Complications with versioning make full record counts very slow.
Identifying Good Field Candidates for Indexes

- Good Selectivity (% distinct values in field)
- Used in WHERE? ON? GROUP BY? ORDER BY?
- How to determine selectivity of current indexes?
  - SHOW INDEX FROM some_table
    - Repeat as needed
  - SELECT COUNT(DISTINCT some_field)/COUNT(*) FROM some_table
    - Repeat as needed
  - or, use the INFORMATION_SCHEMA ...
INFORMATION_SCHEMA is your friend

<table>
<thead>
<tr>
<th>TABLE_SCHEMA</th>
<th>TABLE_NAME</th>
<th>INDEX_NAME</th>
<th>COLUMN_NAME</th>
<th>SEQ_IN_INDEX</th>
<th>COLS_IN_INDEX</th>
<th>CARD</th>
<th>ROWS</th>
<th>SEL %</th>
</tr>
</thead>
<tbody>
<tr>
<td>worklog</td>
<td>amendments</td>
<td>text</td>
<td>text</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>33794</td>
<td>0.00</td>
</tr>
<tr>
<td>planetmysql</td>
<td>entries</td>
<td>categories</td>
<td>categories</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4171</td>
<td>0.02</td>
</tr>
<tr>
<td>planetmysql</td>
<td>entries</td>
<td>categories</td>
<td>title</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4171</td>
<td>0.02</td>
</tr>
<tr>
<td>planetmysql</td>
<td>entries</td>
<td>categories</td>
<td>content</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>4171</td>
<td>0.02</td>
</tr>
<tr>
<td>sakila</td>
<td>inventory</td>
<td>idx_store_id_film_id</td>
<td>store_id</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4673</td>
<td>0.02</td>
</tr>
<tr>
<td>sakila</td>
<td>rental</td>
<td>idx_fk_staff_id</td>
<td>staff_id</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>16291</td>
<td>0.02</td>
</tr>
<tr>
<td>worklog</td>
<td>tasks</td>
<td>title</td>
<td>title</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3567</td>
<td>0.03</td>
</tr>
<tr>
<td>worklog</td>
<td>tasks</td>
<td>title</td>
<td>description</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3567</td>
<td>0.03</td>
</tr>
<tr>
<td>sakila</td>
<td>payment</td>
<td>idx_fk_staff_id</td>
<td>staff_id</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>15422</td>
<td>0.04</td>
</tr>
<tr>
<td>mysqlforge</td>
<td>mw_recentchanges</td>
<td>rc_ip</td>
<td>rc_ip</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>996</td>
<td>0.20</td>
</tr>
</tbody>
</table>
Effects of Column Order in Indexes

The Tag2Project Table:

```
CREATE TABLE Tag2Project (
  tag INT UNSIGNED NOT NULL,
  project INT UNSIGNED NOT NULL,
  PRIMARY KEY (tag, project)
) ENGINE=MyISAM;
```

```sql
mysql> EXPLAIN SELECT project, COUNT(*) as num_tags
           -> FROM Tag2Project
           -> GROUP BY project;
+-------------+-------+---------+----------------------------------------------+
| table       | type  | key     | Extra                                        |
+-------------+-------+---------+----------------------------------------------+
| Tag2Project | index | PRIMARY | Using index; Using temporary; Using filesort |
+-------------+-------+---------+----------------------------------------------+

mysql> EXPLAIN SELECT tag, COUNT(*) as num_projects
           -> FROM Tag2Project
           -> GROUP BY tag;
+-------------+-------+---------+-------------+
| table       | type  | key     | Extra       |
+-------------+-------+---------+-------------+
| Tag2Project | index | PRIMARY | Using index |
+-------------+-------+---------+-------------+

mysql> CREATE INDEX project ON Tag2Project (project);
Query OK, 701 rows affected (0.01 sec)
Records: 701  Duplicates: 0  Warnings: 0
```

```sql
mysql> EXPLAIN SELECT project, COUNT(*) as num_tags
           -> FROM Tag2Project
           -> GROUP BY project;
+-------------+-------+---------+-------------+
| table       | type  | key     | Extra       |
+-------------+-------+---------+-------------+
| Tag2Project | index | project | Using index |
+-------------+-------+---------+-------------+
```
Maximum Velocity MySQL

Black-belt SQL Coding
SQL Coding Guidelines

• Change the way you think
  – SQL Programming != Procedural Programming

• No more “for” loop thinking
  – Instead, learn to think in “sets”

• KISS (Keep it Simple and Straightforward)
  – “Chunky” coding
  – If it looks too complex, break it down

• Be consistent
  – Helps you and your team
Thinking in Sets

“Show the maximum price that each product was sold, along with the product name for each product”

• Many programmers think:
  – OK, **for each** product, find the maximum price the product was sold and output that with the product's name (**WRONG!**)

• Think instead:
  – OK, I have **2 sets** of data here. One set of product names and another set of maximum sold prices
Not everything is as it seems...

```sql
mysql> EXPLAIN SELECT
    ->   p.*
    ->   FROM payment p
    ->   WHERE p.payment_date =
    ->     ( SELECT MAX(payment_date)
    ->         FROM payment
    ->         WHERE customer_id=p.customer_id);
```

```
+--------------------+---------+------+---------------------------------+--------------+---------------+-------+-------------+
| select_type        | table   | type | possible_keys                   | key          | ref           | rows  | Extra       |
+--------------------+---------+------+---------------------------------+--------------+---------------+-------+-------------+
| PRIMARY            | p       | ALL  | NULL                            | NULL         | NULL          | 16451 | Using where |
| DEPENDENT SUBQUERY | payment | ref  | idx_fk_customer_id,payment_date | payment_date | p.customer_id |     12 | Using index |
+--------------------+---------+------+---------------------------------+--------------+---------------+-------+-------------+
3 rows in set (0.00 sec)
```

```sql
mysql> EXPLAIN SELECT
    ->   p.*
    ->   FROM ( 
    ->     SELECT customer_id, MAX(payment_date) as last_order
    ->     FROM payment
    ->     GROUP BY customer_id
    ->   ) AS last_orders
    ->   INNER JOIN payment p
    ->   ON p.customer_id = last_orders.customer_id
    ->   AND p.payment_date = last_orders.last_order;
```

```
+-------------+------------+-------+-------------------------+--------------------+--------------------------------+-------+
| select_type | table      | type  | possible_keys                   | key                | ref                    | rows  |
+-------------+------------+-------+---------------------------------+--------------------+------------------------+-------+
| PRIMARY     | <derived2> | ALL   | NULL                            | NULL               | NULL                   |   599 |
| PRIMARY     | p          | ref   | idx_fk_customer_id,payment_date | payment_date       | customer_id,last_order |     1 |
| DERIVED     | payment    | index | NULL                            | idx_fk_customer_id | NULL                   | 16451 |
+-------------+------------+-------+---------------------------------+--------------------+------------------------+-------+
3 rows in set (0.10 sec)
```
...not what you expected?

```sql
mysql> SELECT
    > p.*
    > FROM payment p
    > WHERE p.payment_date =
    > ( SELECT MAX(payment_date)
    >   FROM payment
    >   WHERE customer_id=p.customer_id);

<table>
<thead>
<tr>
<th>payment_id</th>
<th>customer_id</th>
<th>staff_id</th>
<th>rental_id</th>
<th>amount</th>
<th>payment_date</th>
<th>last_update</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

623 rows in set (0.49 sec)
```

```sql
mysql> SELECT
    > p.*
    > FROM ( 
    >  SELECT customer_id, MAX(payment_date) as last_order 
    >  FROM payment 
    >  GROUP BY customer_id 
    > ) AS last_orders 
    > INNER JOIN payment p 
    > ON p.customer_id = last_orders.customer_id 
    > AND p.payment_date = last_orders.last_order;

<table>
<thead>
<tr>
<th>payment_id</th>
<th>customer_id</th>
<th>staff_id</th>
<th>rental_id</th>
<th>amount</th>
<th>payment_date</th>
<th>last_update</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

623 rows in set (0.09 sec)
```
Isolate those Indexed Fields!

mysql> EXPLAIN SELECT * FROM film WHERE title LIKE 'Tr%'
G
*************************** 1. row ***************************
id: 1
select_type: SIMPLE
table: film
type: range
possible_keys: idx_title
key: idx_title
key_len: 767
ref: NULL
rows: 15
Extra: Using where

Nice. In the top query, we have a fast range access on the indexed field.

mysql> EXPLAIN SELECT * FROM film WHERE LEFT(title,2) = 'Tr'  
G
*************************** 1. row ***************************
id: 1
select_type: SIMPLE
table: film
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 951
Extra: Using where

Oops. In the bottom query, we have a slower full table scan because of the function operating on the indexed field (the LEFT() function).
A Very Common Isolated Index Field Problem

SELECT * FROM Orders
WHERE TO_DAYS(CURRENT_DATE()) - TO_DAYS(order_created) <= 7;

Not a good idea! Lots o' problems with this...

SELECT * FROM Orders
WHERE order_created >= CURRENT_DATE() - INTERVAL 7 DAY;

Better... Now the index on order_created will be used at least. Still a problem, though...

SELECT * FROM Orders
WHERE order_created >= '2007-02-11' - INTERVAL 7 DAY;

Best. Now the query cache can cache this query, and given no updates, only run it once a day...
Calculated Field Example

```sql
CREATE TABLE Customers (  
customer_id INT NOT NULL  ,
email VARCHAR(80) NOT NULL  
) ENGINE=InnoDB;

// Bad idea, can't use index  
// on email field
SELECT *  
FROM Customers  
WHERE email LIKE '%.com';

// So, we enable fast searching on a reversed field  
// value by inserting a calculated field
ALTER TABLE Customers
ADD COLUMN rv_email VARCHAR(80) NOT NULL;

// Now, we update the existing table values
UPDATE Customers SET rv_email = REVERSE(email);

// Then, we create an index on the new field
CREATE INDEX ix_rv_email ON Customers (rv_email);

// Then, we make a trigger to keep our data in sync
DELIMITER ;;
CREATE TRIGGER trg_bi_cust  
BEFORE INSERT ON Customers  
FOR EACH ROW BEGIN  
SET NEW.rv_email = REVERSE(NEW.email);  
END ;;

// same trigger for BEFORE UPDATE...
// Then SELECT on the new field...
WHERE rv_email LIKE CONCAT(REVERSE('com'), '%');
```
Tuning Server Settings
SHOW STATUS and SHOW VARIABLES

• SHOW STATUS
  – Counter variables (lots of `em)
  – Count reads, writes, threads, etc.

• SHOW VARIABLES
  – Your configuration variables

• Both take a LIKE clause, for example:

```sql
SHOW STATUS LIKE 'Created_tmp%';
```

<table>
<thead>
<tr>
<th>Variable_name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Created_tmp_disk_tables</td>
<td>499</td>
</tr>
<tr>
<td>Created_tmp_files</td>
<td>5</td>
</tr>
<tr>
<td>Created_tmp_tables</td>
<td>1933</td>
</tr>
</tbody>
</table>

```
Server Variable Guidelines

- Be aware of what is global vs per thread
- Make small changes, then test
- Often provide a quick solution, but temporary
- Query Cache is not a panacea
- `key_buffer_size` != `innodb_buffer_pool_size`
- Remember mysql system database is MyISAM
- Memory is cheapest, fastest, easiest way to increase performance
MyISAM

• **key_buffer_size**
  - Main MyISAM key cache (blocks of size 1K)
  - Watch for **Key_blocks_unused** approaching 0

• **table_cache** (InnoDB too...)
  - Number of simultaneously open file descriptors
    • < 5.1 contains meta data about tables and file descriptor
    • >= 5.1 Split into table_open_cache

• **myisam_sort_buffer_size**
  - Building indexes, set this as high as possible
• Examine **Handler_read_rnd_next/Handler_read_rnd** for average size of table scans

```
mysql> SHOW STATUS LIKE 'Handler_read_rnd%';
+-----------------------+--------+
| Variable_name         | Value  |
+-----------------------+--------+
| Handler_read_rnd      | 2188   |
| Handler_read_rnd_next | 217247 |
+-----------------------+--------+
```

• Examine **Key_read_requests/Key_reads** for your key_cache hit ratio

```
mysql> SHOW STATUS LIKE 'Key_read%';
+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| Key_read_requests | 10063 |
| Key_reads         | 98    |
+-------------------+-------+
```
InnoDB

- **innodb_buffer_pool_size**
  - Main InnoDB cache for both data and index pages (16K page)
  - If you have InnoDB-only system, set to 60-80% of total memory
  - Watch for `Innodb_buffer_pool_pages_free` approaching 0

- **innodb_log_file_size**
  - Size of the actual log file
  - Set to 40-50% of `innodb_buffer_pool_size`
InnoDB (cont'd)

- **innodb_log_buffer_size**
  - Size of double-write log buffer
  - Set < 16M (recommend 1M to 8M)
- **innodb_flush_method**
  - Determines how InnoDB flushes data and logs
  - defaults to fsync()
  - If getting lots of **Innodb_data_pending_fsyncs**
    - Consider O_DIRECT (Linux only)
  - Other ideas
  - Get a battery-backed disk controller with a write-back cache
  - Set **innodb_flush_log_at_trx_commit=2** (Risky)
Examining Hit Rates (InnoDB and Query Cache)

- Examine `Innodb_buffer_pool_reads` vs `Innodb_buffer_pool_read_requests` for the cache hit ratio

```
mysql> SHOW STATUS LIKE 'Innodb_buffer_pool_read%';
+-----------------------------------+---------+
| Variable_name                     | Value   |
+-----------------------------------+---------+
| Innodb_buffer_pool_read_requests  | 5415365 |
| Innodb_buffer_pool_reads          | 34260   |
+-----------------------------------+---------+
```

- Examine `Qcache_hits/Questions` for the query cache hit ratio

```
mysql> SHOW STATUS LIKE 'Qc%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Qcache_free_blocks      | 1     |
| Qcache_hits             | 6     |
| Qcache_inserts          | 12    |
| Qcache_not_cached       | 41    |
| Qcache_lowmem_prunes    | 0     |
| Questions               | 241   |
+-------------------------+-------+
```

- Ensure `Qcache_lowmem_prunes` is low
- Ensure `Qcache_free_blocks > 0`
Further Reading

- **Optimizing Linux Performance**
  - Philip Ezolt, HP Press
- [http://www.mysqlperformanceblog.com/](http://www.mysqlperformanceblog.com/)
  - Peter Zaitsev
- [http://xaprb.com](http://xaprb.com)
  - Baron Schwartz
- [http://planetmysql.org](http://planetmysql.org)
  - Planet MySQL

- **Advanced PHP Programming**
  - George Schlossnagle, Developer's Library
Thanks!

Come to the Users Conference in April!

http://mysqlconf.com