Performance Tuning Best Practices

Jay Pipes
Community Relations Manager, North America
(jay@mysql.com)

TELECONFERENCE: please dial a number to hear the audio portion of this presentation.

Toll-free US/Canada: 866-469-3239	Direct US/Canada: 650-429-3300 :
0800-295-240 Austria	0800-71083 Belgium
80-884912 Denmark	0-800-1-12585 Finland
0800-90-5571 France	0800-101-6943 Germany
00800-12-6759 Greece	1-800-882019 Ireland
800-780-632 Italy	0-800-9214652 Israel
800-2498 Luxembourg	0800-022-6826 Netherlands
800-15888 Norway	900-97-1417 Spain
020-79-7251 Sweden	0800-561-201 Switzerland
0800-028-8023 UK	1800-093-897 Australia
800-90-3575 Hong Kong	00531-12-1688 Japan

EVENT NUMBER/ACCESS CODE: 921155876

MySQL: The World's Most Popular Open Source Database

Founded in 1995; operations in 23 countries

Fastest growing relational database

Over 8,000,000 installations; 40,000 downloads / day

Dramatically reduces Total Cost of Ownership (TCO)

Used by leading IT organizations and ISVs

Second Generation Open Source

- **MySQL AB is a profitable company**
  - Develops the software in-house; community helps test it
  - Owns source code, copyrights and trademarks
  - Targets the “commoditized” market for databases
- **“Quid Pro Quo” dual licensing for OEM market**
  - Open source GPL license for open source projects
  - Cost-effective commercial licenses for commercial use
- **Annual MySQL Network subscription for Enterprise and Web**
  - Per server annual subscription
  - Includes support, alert and update advisors, Knowledge Base, Certified/Optimized Binaries
- **MySQL supports it users**
  - Worldwide 24 x 7 support
  - Training and certification
  - Consulting

> “Reasoning's inspection study shows that the code quality of MySQL was six times better than that of comparable proprietary code. ”
>
> **Reasoning Inc.**

Overview

- Profiling and Benchmarking Concepts
- Sources of Problems
- Indexing Guidelines
- Schema Guidelines
- Coding Guidelines
- Server Parameters

Benchmarking Concepts

- Provides a track record of changes
  - Baseline is the starting point
  - Testing done iteratively
  - Deltas between tests show difference that the change(s) made
- Stress/Load testing of application and/or database
- Harness or framework useful to automate many benchmark tasks

Benchmarking Tips

- Always give yourself a target
- Record *everything*
  - ✓ Schema dump
  - ✓ my.cnf files
  - ✓ hardware/os configuration files as needed
- Isolate the problem
  - ✓ Shut down unnecessary programs
  - ✓ Stop network traffic to machine
  - ✓ Disable the query cache
  - ✓ Change one thing at a time

Benchmarking Toolbox

- SysBench
  - http://sysbench.sourceforge.net/
- mysqlslap (5.1+)
  - http://dev.mysql.com/doc/refman/5.1/en/mysqlslap.html
- Apache Bench (ab)
- supersmack
  - http://www.vegan.net/tony/supersmack/
- MyBench
  - http://jeremy.zawodny.com/mysql/mybench/

Profiling Concepts

- Diagnose a running system
- Low hanging fruit
  - Diminishing returns
  - Be careful not to over-optimize
- Identify performance bottlenecks in
  - Memory
  - CPU
  - I/O (Disk)
  - Network and OS

Profiling Toolbox

- SHOW Commands
  - `SHOW PROCESSLIST | STATUS | INNODB STATUS`
  - http://dev.mysql.com/show
- EXPLAIN
  - http://dev.mysql.com/explain
- MyTop
  - http://jeremy.zawodny.com/mysql/mytop/
- Whole host of Linux power tools
  - gprof / oprofile
  - vmstat / ps / top / mpstat / procinfo
- apd for PHP developers
  - http://pecl.php.net/package/apd

Slow Query Log

- Slow Query Log
  - `log_slow_queries=/var/lib/mysql/slow-queries.log`
  - `long_query_time=2`
  - Use mysqldumpslow
  - (5.1+) Can log directly to a table, plus does not require restart of server
    - `SET GLOBAL SLOW_QUERY_LOG = { ON | OFF }`
    - http://dev.mysql.com/doc/refman/5.1/en/log-tables.html

Profiling Tips

- Get *very* familiar with `EXPLAIN`
  - Access types
  - Learn the `type`, `key`, `ref`, `rows`, `Extra` columns
- Low hanging fruit (diminishing returns)
- Use MyTop to catch locking and long-running queries in real-time

Sources of Problems

- Poor or nonexistent indexing
- Inefficient or bloated schema design
- Bad SQL Coding Practices
- Server variables not tuned properly
- Hardware and/or network bottlenecks

Indexing Guidelines

- Poor or missing index fastest way to kill a system
- Ensure good selectivity on field
- Look for covering index opportunities
- On multi-column indexes, pay attention to the order of the fields in the index (example ahead)
- As database grows, examine distribution of values within indexed field
- Remove redundant indexes for faster write performance

Common Index Problem

```
CREATE TABLE Tags (
  tag_id INT NOT NULL AUTO_INCREMENT
, tag_text VARCHAR(50) NOT NULL
, PRIMARY KEY (tag_id)
) ENGINE=MyISAM;

CREATE TABLE Products (
  product_id INT NOT NULL AUTO_INCREMENT
, name VARCHAR(100) NOT NULL
// many more fields...
, PRIMARY KEY (product_id)
) ENGINE=MyISAM;

CREATE TABLE Products2Tags (
  product_id INT NOT NULL
, tag_id INT NOT NULL
, PRIMARY KEY (product_id, tag_id)
) ENGINE=MyISAM;
```

```
// This top query uses the index
// on Products2Tags

SELECT p.name
, COUNT(*) as tags
FROM Products2Tags p2t
INNER JOIN Products p
ON p2t.product_id = p.product_id
GROUP BY p.name;

// This one does not because
// index order prohibits it

SELECT t.tag_text
, COUNT(*) as products
FROM Products2Tags p2t
INNER JOIN Tags t
ON p2t.tag_id = t.tag_id
GROUP BY t.tag_text;
```

Common Index Problem Solved

```
CREATE TABLE Tags (
  tag_id INT NOT NULL AUTO_INCREMENT
, tag_text VARCHAR(50) NOT NULL
, PRIMARY KEY (tag_id)
) ENGINE=MyISAM;

CREATE TABLE Products (
  product_id INT NOT NULL AUTO_INCREMENT
, name VARCHAR(100) NOT NULL
// many more fields...
, PRIMARY KEY (product_id)
) ENGINE=MyISAM;

CREATE TABLE Products2Tags (
  product_id INT NOT NULL
, tag_id INT NOT NULL
, PRIMARY KEY (product_id, tag_id)
) ENGINE=MyISAM;
```

```
CREATE INDEX ix_tag
ON Products2Tags (tag_id);

// or... create a covering index:

CREATE INDEX ix_tag_prod
ON Products2Tags (tag_id, product_id);

// But, only if not InnoDB... why?
```

Schema Guidelines

- Inefficient schema another great way to kill performance
- Use the smallest data types necessary
  - Do you really need that `BIGINT`?
- Normalize first, denormalize only in extreme cases

Schema Tips

- Consider horizontally splitting many-columned tables (example ahead)
- Consider vertically partitioning many-rowed tables
  - Merge tables (MyISAM only)
  - Homegrown
  - Partitioning (5.1+)
- Fewer fields = Narrow rows = More records per block
- Use “counter” tables to mitigate query cache issues (example ahead)
  - Essential for InnoDB

Horizontal Partitioning Example

```
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;
```

```
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
, PRIMARY KEY (user_id)
, UNIQUE INDEX (email)
) ENGINE=InnoDB;

CREATE TABLE UserExtra (
  user_id INT 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)
) ENGINE=InnoDB;
```

Horizontal Partitioning Benefits

- Main table has narrow rows, so...
  - ✓ More records fit into a single data page
  - ✓ Fewer reads from memory/disk to get same number of records
- Less frequently queried data doesn't take up memory
- More possibilities for indexing and different storage engines
  - ➢ Allows targeted multiple MyISAM key caches for hot and cold data

Counter Table Example

```
CREATE TABLE Products (
  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

// Getting a simple COUNT of products
// easy on MyISAM, terrible on InnoDB
SELECT COUNT(*)
FROM Products;
```

```
CREATE TABLE Products (
  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
, 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;

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

Counter Table Benefits

- Critical for InnoDB because of complications of MVCC
- Allows query cache to cache specific data set which will be invalidated only infrequently
- Allows you to target `SQL_NO_CACHE` for `SELECT`s against counter tables, freeing query cache
- Allows MEMORY storage engine for summary counters, since stats can be rebuilt

Schema Tips (cont'd)

- Ensure small clustering key (InnoDB)
- Don't use surrogate keys when a naturally occurring primary key exists
  - Example (of what not to do):

```
CREATE TABLE Products2Tags (
  record_id INT UNSIGNED NOT NULL AUTO_INCREMENT
, product_id INT UNSIGNED NOT NULL
, tag_id INT UNSIGNED NOT NULL
, PRIMARY KEY (record_id)
, UNIQUE INDEX (product_id, tag_id)
) ENGINE=InnoDB;
```

Coding Guidelines

- Use “chunky” coding habits (KISS)
- Use stored procedures for a performance boost (5.0+)
- Isolate indexed fields on one side of equation (example ahead)
- Use calculated fields if necessary (example ahead)
- Learn to use joins (!)
  - Eliminate correlated subqueries using standard joins (examples ahead)
- Don't try to outthink the optimizer
  - Sergey, Timour and Igor are really, really smart...

Isolating Indexed Fields Example

- Task: get the Order ID, date of order, and Customer ID for all orders in the last 7 days

```
// Bad idea
SELECT *
FROM Orders
WHERE
TO_DAYS(order_created) –
TO_DAYS(CURRENT_DATE()) >= 7;
```

```
// Better idea
SELECT *
FROM Orders
WHERE
order_created >= CURRENT_DATE() – INTERVAL 7 DAY;

// Best idea is to factor out the CURRENT_DATE
// non-deterministic function in your application
// code and replace the function with a constant.
// Now, query cache can actually cache the query!
SELECT order_id, order_created, customer_id
FROM Orders
WHERE order_created >= '2006-05-24' – INTERVAL 7 DAY;
```

Calculated Fields Example

- Task: search for top-level domain in email addresses

```
// Initial schema
CREATE TABLE Customers (
  customer_id INT NOT NULL
, email VARCHAR(80) NOT NULL
// more fields
, PRIMARY KEY (customer_id)
, INDEX (email(40))
) 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 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'), '%');
```

Correlated Subquery Conversion Example

- Task: convert a correlated subquery in the `SELECT` clause to a standard join

```
// Bad practice
SELECT p.name
,   (SELECT MAX(price)
     FROM OrderItems
     WHERE product_id = p.product_id)
AS max_sold_price
FROM Products p;
```

```
// Good practice
SELECT p.name
, MAX(oi.price) AS max_sold_price
FROM Products p
 INNER JOIN OrderItems oi
  ON p.product_id = oi.product_id
GROUP BY p.name;
```

Derived Table Example

- ✓ Task: convert a correlated subquery in the `WHERE` clause to a standard join on a derived table

```
// Bad performance
SELECT
c.company
, o.*
FROM Customers c
 INNER JOIN Orders o
  ON c.customer_id = o.customer_id
WHERE order_date = (
 SELECT MAX(order_date)
 FROM Orders
 WHERE customer = o.customer
)
GROUP BY c.company;
```

```
// Good performance
SELECT
c.company
, o.*
FROM Customers c
 INNER JOIN (
  SELECT
     customer_id
   , MAX(order_date) as max_order
  FROM Orders
  GROUP BY customer_id
 ) AS m
  ON c.customer_id = m.customer_id
 INNER JOIN Orders o
  ON c.customer_id = o.customer_id
  AND o.order_date = m.max_order
GROUP BY c.company;
```

Server Parameters

- 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_size`
  - Also, remember `mysql` system database is MyISAM
- Memory is cheapest, fastest, easiest way to increase performance

Additional Resources

- http://www.mysqlperformanceblog.com/
  - Peter Zaitsev's blog – Excellent material
- *Optimizing Linux Performance*
  - Philip Ezolt (HP Press)
- http://dev.mysql.com/tech-resources/articles/pro-mysql-ch6.pdf
  - *Pro MySQL* (Apress) chapter on profiling (`EXPLAIN`)
- *Advanced PHP Programming*
  - George Schlossnagle (Developer's Library)

THANK YOU!

- Please email me:
  - Success stories
  - War stories
  - Inventive uses of MySQL
- Feedback on webinar
- Other webinar or article topics you'd like to hear about
- Gripes :)
- Anything else you feel like talking about!

Jay Pipes

(jay@mysql.com)
MySQL, Inc.