Fourth Annual

MySQL®

Users Conference 2006

Performance Tuning Best Practices

Jay Pipes
Community Relations Manager
MySQL, Inc.

Presented by MySQL O'REILLY

Overview

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

Benchmarking Concepts

- Benchmarks provide a track record
- Baseline provides the starting point
- Always give yourself a target
- Change only one thing at a time
- Record *everything*
- Disable query cache

Profiling Concepts

- Diagnostics on running system
- Get familiar with EXPLAIN
- Use the Slow Query Log and mysqldumpslow
- Low hanging fruit and diminishing returns
- Use mytop to catch locking and long-running queries

Sources of Problems

- Poor Indexing Choices
- Inefficient or Bloated Schema Design
- Bad Coding Practices
- Server Variables Not Tuned Properly
- Hardware and/or Network Bottlenecks

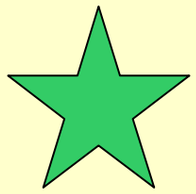
Index Guidelines

- Poor or missing index fastest way to kill a system
- Look for covering index opportunities
- Ensure good selectivity on index fields
- On multi-column indexes, pay attention to order of fields within the index definition
- As database grows, ensure distribution is good
- Remove redundant indexes for faster write performance

Common Index Problem

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

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

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

Only Top Query Uses 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;
```

or...

```
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;
```

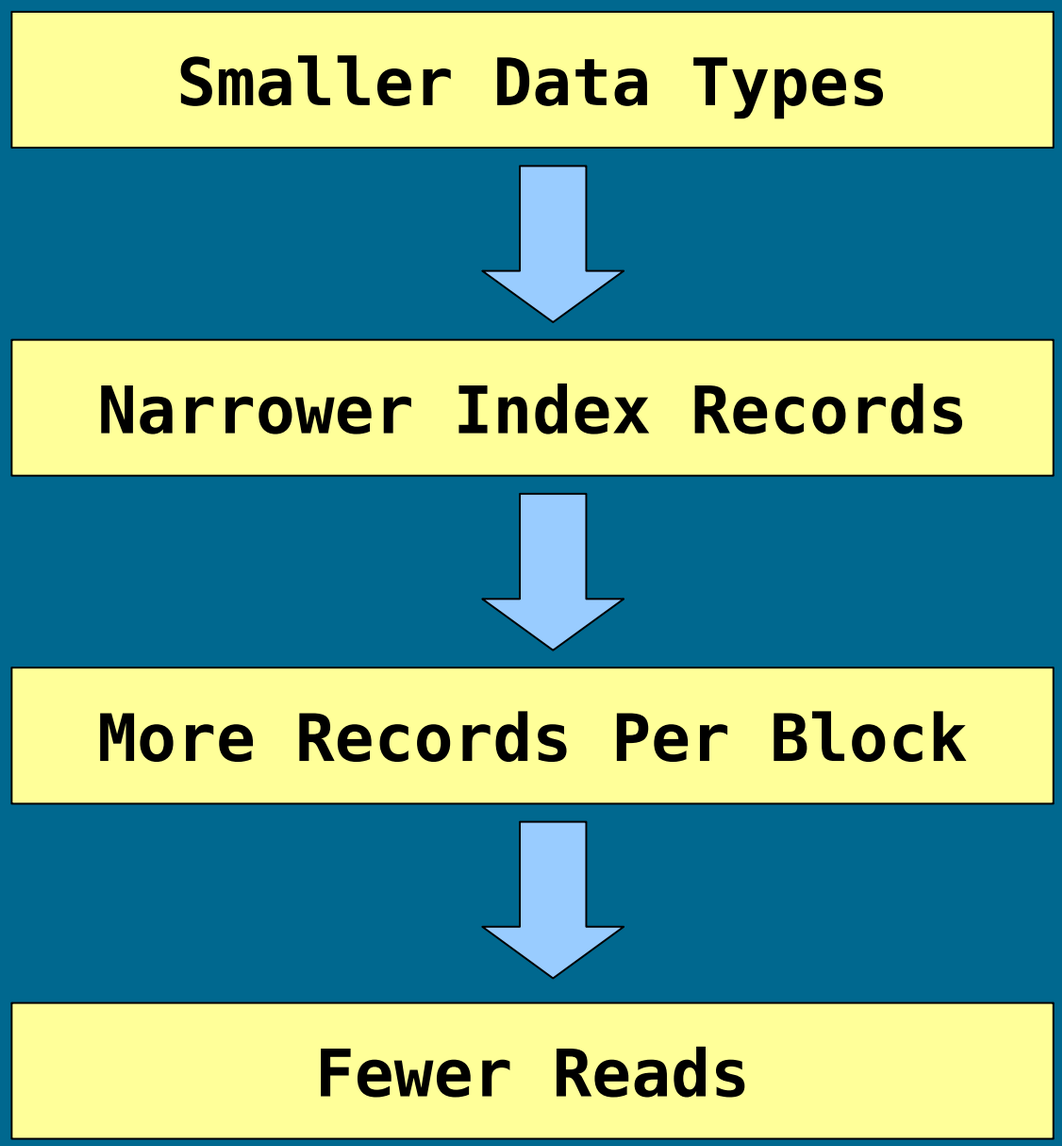
Solving the GROUP BY Problem

```
CREATE TABLE Products2Tags (
  product_id INT UNSIGNED NOT NULL
, tag_id INT UNSIGNED 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);
```

Schema Guidelines

- Inefficient schema another great way to kill your performance
- Use the smallest data types needed
- Consider horizontally splitting many-columned tables
- Consider vertically splitting many-rowed tables using partitioning or Merge tables
- Always remember...fewer reads = faster results

A Pattern For Faster Performance



Schema Guidelines (cont'd)

- InnoDB: Choose a small clustering key since it is appended to each secondary index record
- Don't use surrogate keys when a naturally occuring primary key exists. *BAD* Example:

```
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=MyISAM;
```

Coding Guidelines

- “Chunky” Coding Habits (KISS)
- Use stored procedures for a big performance boost
- InnoDB: Use counter tables
- Isolate indexed fields on one side of equation (example ahead)
- Use calculated fields if needed (example ahead)

Isolating indexed fields

- Always try to isolate indexed fields on one side of condition. Example: Getting Last 7 Days Orders

```
// Bad idea:
WHERE TO_DAYS(order_created) – TO_DAYS(CURRENT_DATE()) >= 7

// Better idea:
WHERE order_created >= CURRENT_DATE() - INTERVAL 7 DAY

// Best idea: in other words, what's wrong with
// the better idea above?
???
```

Using Calculated Fields

- Example: Searching on TLD

```
// Bad idea:
WHERE email_address LIKE '%.com';
// Better idea:
ALTER TABLE Customers
ADD COLUMN rv_email_address VARCHAR(80) NOT NULL;
UPDATE Customers SET rv_email_address = REVERSE(email_address);
CREATE INDEX ix_rv_email ON Customers (rv_email_address(20));

DELIMITER ;;
CREATE TRIGGER trg_bi_cust BEFORE INSERT ON Customers
FOR EACH ROW BEGIN
 SET NEW.rv_email_address = REVERSE(NEW.email_address);
END ;;

// same trigger for BEFORE UPDATE...
// Then SELECT on the new field...
WHERE rv_email_address LIKE CONCAT(REVERSE('.com'), '%');
```

Coding Guidelines (cont'd)

- Learn to use joins
- Eliminate correlated subqueries (examples ahead)
- Don't try to outhink the optimizer (hey, Igor, Sergey and Timour are pretty smart...)

Correlated Subqueries

- Convert correlated subqueries to a standard join. Example:

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

```
//Better 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;
```

Using Derived Tables

```
// Correlated subquery
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;
```

```
// Derived Table
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 what is global vs. per thread
- Make small changes, test
- Often provide a quick solution, but temporary
- Query Cache (not a panacea)
- key_buffer_size != innodb_buffer_pool_size (different storage engines for different scenarios)
- Memory is cheapest, fastest, easiest way to better performance