



Presented by,
MySQL AB® & O'Reilly Media, Inc.

The Top 20 Design Tips For MySQL Enterprise Data Architects

Ronald Bradford

COO

PrimeBase Technologies

April 2008





1. Know Your Technology Tools

- ❖ Generics are inefficient
- ❖ Product expertise in a different RDBMS is not enough
- ❖ You have chosen MySQL
 - ❖ Maximize it's strengths
 - ❖ Minimize it's weaknesses



Overview

- ❖ Table Structure
- ❖ SQL
- ❖ Indexes
- ❖ Enterprise Approaches



1. Know Your Technology Tools

- ❖ Maximize MySQL strengths
 - ❖ Scale out / HA Options
 - ❖ Different Storage Engines
 - ❖ Query Cache
- ❖ Minimize MySQL weaknesses
 - ❖ No Online Alter
 - ❖ Backup Strategies
 - ❖ Instrumentation



2. Know Your Disk Footprint

Disk = Memory = Performance

- ❖ Every single byte counts
- ❖ Average 25% - 30% saving on engagements
- ❖ Better 60% (200GB System)
- ❖ Best 78% (8GB per master with 12 masters)

Less disk accesses and more data in memory



3. Choose Your Numeric Data Type

- ❖ MySQL has 9 numeric data types
 - ❖ Oracle for example has only 1



3. Choose Your Numeric Data Type

- ❖ Integer: TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT
- ❖ Floating Point: FLOAT, DOUBLE
- ❖ Fixed Point: DECIMAL
- ❖ Other: BIT, (ENUM maybe)



3. Choose Your Numeric Data Type

- ❖ Favorite signs of poor design
 - ❖ INT(1)
 - ❖ BIGINT AUTO_INCREMENT
 - ❖ no UNSIGNED used
 - ❖ DECIMAL(31,0)



3. Choose Your Numeric Data Type

- ❖ INT(1) - 1 does not mean 1 digit
 - ❖ (1) represents client output display format only
 - ❖ INT is 4 Bytes, TINYINT is 1 Byte
 - ❖ TINYINT UNSIGNED can store from 0 – 255
 - ❖ BIT is even better when values are 0 - 1



3. Choose Your Numeric Data Type

- ❖ BIGINT is not needed for AUTO_INCREMENT
- ❖ INT UNSIGNED stores 4.3 billion values
 - ❖ You should be partitioning when at billions of rows
- ❖ BIGINT is applicable for some columns
 - ❖ e.g. summation of values



3. Choose Your Numeric Data Type

Best
Practice

❖ Best Practice

- ❖ All integer columns UNSIGNED unless there is a reason otherwise
- ❖ Adds a level of data integrity for negative values



4. Other Data Type Efficiencies

❖ TIMESTAMP v DATETIME

- ❖ Suitable for EPOCH only values
- ❖ TIMESTAMP is 4 bytes
- ❖ DATETIME is 8 bytes
- ❖ FYI: DATE is 3 bytes, TIME is 3 bytes = 6 Bytes???



4. Other Data Type Efficiencies

- ❖ CHAR(n)
- ❖ Use VARCHAR(n) for variable values
- ❖ e.g. CHAR(128) when storing ~10 bytes



5. Application Data Type Efficiencies

- ❖ Using Codes or ENUM
 - ❖ A description is a presentation layer function
 - ❖ e.g. 'M', 'F' instead of 'Male', 'Female'
 - ❖ e.g. 'A', 'I' instead of 'Active', 'Inactive'
- ❖ BINARY(16/20) v CHAR(32/40)
 - ❖ MD5() or HASH() Hex value with twice the length
- ❖ INT UNSIGNED for IPv4 address
 - ❖ VARCHAR(15) results in average 12 bytes v 4 bytes



6. NOT NULL

- ❖ Saves up to a byte per column per row of data
- ❖ Double benefit for indexed columns
- ❖ Don't use frameworks or tools
- ❖ NOT NULL DEFAULT " is bad design

**Always use NOT NULL unless
there is a reason why not**

Best
Practice



7. Know about character sets

- ❖ Default in MySQL 5 is UTF8
- ❖ Can be defined at database, schema, table or column level
- ❖ Only define columns that need UTF8
 - ❖ e.g. Codes, MD5 Value, web address
- ❖ MySQL internal buffers are fixed width
 - ❖ e.g. VARCHAR(255) utf8 is 765 bytes to store just 1 byte



8. When VARCHAR Is Bad

- ❖ VARCHAR(255)
 - ❖ Poor Design - No understanding of underlying data
 - ❖ Old Design - (4.x limitation, now 3-4 years old)
- ❖ Disk usage may be efficient
- ❖ MySQL internal memory usage is not



8. When VARCHAR is bad

```
CREATE TABLE `XXX` (  
  `orderHandle` varchar(255) NOT NULL default '',  
  `personName` varchar(255) default NULL,  
  `addressLines` varchar(255) default NULL,  
  `city` varchar(255) default NULL,  
  `state` varchar(255) default NULL,  
  `postalCode` varchar(255) default NULL,  
  `countryCode` varchar(255) default NULL,  
  `phone` varchar(255) default NULL,  
  `email` varchar(255) default NULL,  
  `shipMethod` varchar(255) default NULL,  
  `shipTo` varchar(255) default NULL,  
  `receiveByDate` date default NULL,  
  `currency` varchar(3) default NULL,  
  `price` varchar(255) default NULL,  
  `flags` int(11) default '0',  
  `lastUpdateTime` timestamp NOT NULL default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,  
  `creationTime` timestamp NOT NULL default '0000-00-00 00:00:00',  
  PRIMARY KEY (`orderHandle`)  
) ENGINE=MyISAM DEFAULT CHARSET=utf8
```



9. Be Wary of TEXT/BLOB

- ❖ Using SELECT *
 - ❖ MySQL Internal Temporary table will force Temp Disk Table
- ❖ Internal storage (e.g. Innodb)
 - ❖ Stores first 768 bytes, then a separate 16k data page per row per TEXT/BLOB field



10. Know Every SQL Statement

- ❖ Developers don't write proper SQL statements
- ❖ SQL statements will directly affect your performance
- ❖ For Example
 - ❖ Repeating SQL statements for no benefit
 - ❖ 1000 very quick small unnecessary queries is worse than 1 slow query



10. Know Every SQL Statement

- ❖ Data collection options
- ❖ Incomplete Options
 - ❖ Slow Query Log
 - ❖ SHOW PROCESSLIST
 - ❖ Application level logging
- ❖ Impractical Options
 - ❖ General Log



10. Know Every SQL Statement

- ❖ Data collection options
 - ❖ MySQL Proxy
 - ❖ See histogram.lua
 - ❖ Firewall forwarding rules



11. Monitor Every SQL Statement

- ❖ Review Query Execution Plan (QEP)
 - ❖ EXPLAIN
- ❖ Time queries
- ❖ Row Count / Affected rows
- ❖ Result Set Size

Best
Practice

Review over time, things change



12. The Impact Of Indexes

❖ Good

- ❖ Dramatic performance improvements
- ❖ Improves memory usage
- ❖ Data Integrity

❖ Bad

- ❖ Slows performance for writes
- ❖ Wastes disk space for unused, duplicate or ineffective indexes
- ❖ In-effective usage of memory



13. Index Types For Design

- ❖ Concatenated Indexes

 - ❖ (col1, col2)

- ❖ Partial Indexes

 - ❖ (name(20))

- ❖ Covering Indexes

- ❖ Full Text Indexes

- ❖ No function based indexes



14. Minimizing internal MySQL processing

- ❖ Correctly design tables, indexes and SQL to eliminate
 - ❖ Using temporary table
 - ❖ Using filesort



15. Transactions

- ❖ Always design for transactions
- ❖ Always use transactions
- ❖ Use a transactional storage engine



16. Data Integrity is Key

- ❖ MySQL historically has been very lax
- ❖ Warnings (e.g. Truncations) are rarely every caught
- ❖ SQL_MODE=STRICT_ALL_TABLES
- ❖ Within Schema
 - ❖ NOT NULL
 - ❖ ENUM
 - ❖ UNSIGNED



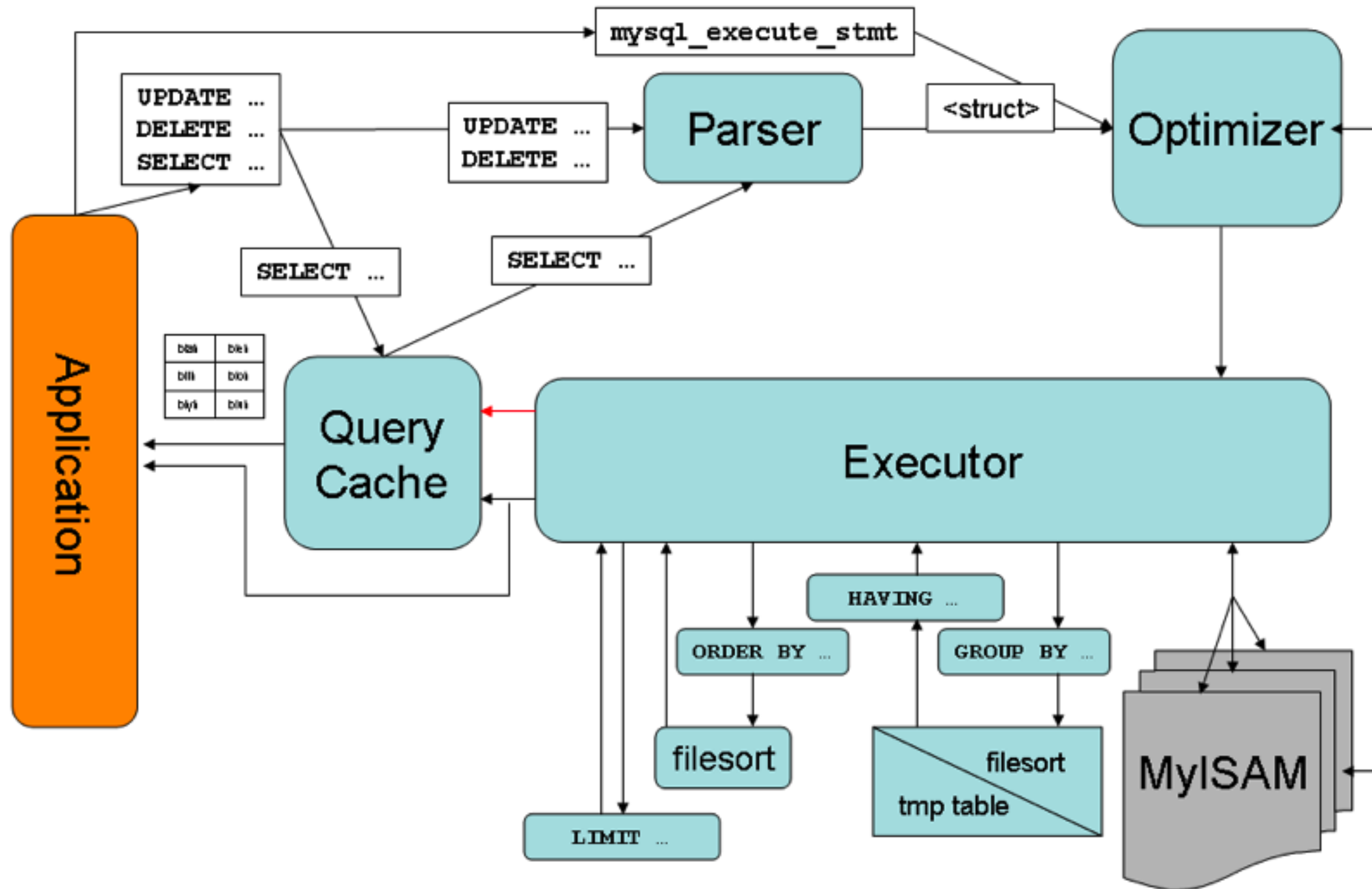
17. Leverage The Query Cache

- ❖ Query Cache can be a great benefit
- ❖ Deterministic v Non Deterministic SQL

Best
Practice

**MySQL Query Cache is not the only type
of caching you should consider**

17. Leverage The Query Cache





17. Leverage The Query Cache

- ❖ SHOW PROFILE Path
- ❖ Simple SELECT
 - ❖ No Query Cache 17 steps
 - ❖ With Query Cache 5 steps
 - ❖ Does not perform parse
 - ❖ Does not perform optimize

The Top 20 Design Tips



Status	Duration	Source_function	Source_file	Source_line
(initialization)	0.000014	send_result_to_client	sql_cache.cc	1143
checking query cache for query	0.000042	open_tables	sql_base.cc	2652
Opening tables	0.000015	mysql_lock_tables	lock.cc	153
System lock	0.000009	mysql_lock_tables	lock.cc	163
Table lock	0.000034	mysql_select	sql_select.cc	2273
init	0.000041	optimize	sql_select.cc	765
optimizing	0.000008	optimize	sql_select.cc	924
statistics	0.000016	optimize	sql_select.cc	934
preparing	0.000012	exec	sql_select.cc	1594
executing	0.000008	exec	sql_select.cc	2114
Sending data	0.000163	mysql_select	sql_select.cc	2318
end	0.000021	mysql_execute_command	sql_parse.cc	5141
query end	0.000007	query_cache_end_of_result	sql_cache.cc	735
storing result in query cache	0.000007	mysql_parse	sql_parse.cc	6142
freeing items	0.000018	dispatch_command	sql_parse.cc	2146
closing tables	0.000009	log_slow_statement	sql_parse.cc	2204
logging slow query	0.000006	dispatch_command	sql_parse.cc	2169

17 rows in set (0.00 sec)

Status	Duration	Source_function	Source_file	Source_line
(initialization)	0.000012	send_result_to_client	sql_cache.cc	1143
checking query cache for query	0.000001	send_result_to_client	sql_cache.cc	1224
checking privileges on cached	0.000007	send_result_to_client	sql_cache.cc	1317
sending cached result to clien	0.000025	log_slow_statement	sql_parse.cc	2204
logging slow query	0.000007	dispatch_command	sql_parse.cc	2169

5 rows in set (0.00 sec)



18. Create Objects Appropriately

- ❖ Using 1 table instead of 'n' for same column structure
 - ❖ e.g. Code table for each type of code
- ❖ Splitting tables for optimal storage
 - ❖ e.g. Placing optional TEXT/BLOB columns in second table
- ❖ Use permanent tables instead of TEMPORARY tables



19. Naming Standards

- ❖ Name all Primary Key's Uniquely
 - ❖ e.g. customer_id, order_id not id
- ❖ Use Data Dictionary SQL to verify data types
 - ❖ Data Types & Lengths
- ❖ Be Descriptive
 - ❖ e.g. invoice_date not just date
- ❖ Avoid Reserved Words
 - ❖ e.g. date, time, timestamp



20. Testing, Testing, Testing

- ❖ You must have a testing environment
- ❖ Testing on a Production server is not an option

Best
Practice

**The goal of a testing environment
is not to test your software,
it is to break your software.**



Executive Summary

- ❖ Learn and know MySQL specifics
- ❖ Disk = Memory = Performance
- ❖ If you don't know your SQL you don't know your application. Log, Review & Monitor all SQL
- ❖ Know all benefits of different indexes
- ❖ You must test to failure in a dedicated test environment



Professional Help is Available

❖ PrimeBase Technologies

- ❖ Technology Experts
- ❖ Solution Experts

❖ 2 decades Expertise & Experience in Enterprise RDBMS Data Architecture

❖ 9 years in MySQL

PRIMEBASE

TECHNOLOGIES